Automatic Algorithm Configuration Methods, Applications, and Perspectives

Thomas Stützle

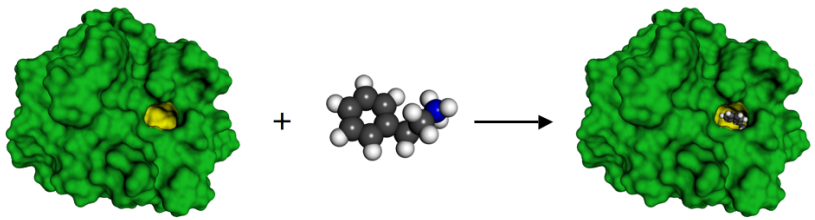
IRIDIA, CoDE, Université Libre de Bruxelles (ULB)
Brussels, Belgium

stuetzle@ulb.ac.be
iridia.ulb.ac.be/~stuetzle

Outline

1. Context
2. Automatic algorithm configuration
3. Automatic configuration methods
4. Applications
5. Concluding remarks

Optimization problems arise everywhere!



Most such problems are computationally very hard (NP-hard!)

The algorithmic solution of hard optimization problems is one of the OR/CS success stories!

- Exact (systematic search) algorithms
  - Branch&Bound, Branch&Cut, constraint programming, ...
  - guarantees on optimality but often time/memory consuming
- Approximate algorithms
  - heuristics, local search, metaheuristics, hyperheuristics ...
  - rarely provable guarantees but often fast and accurate

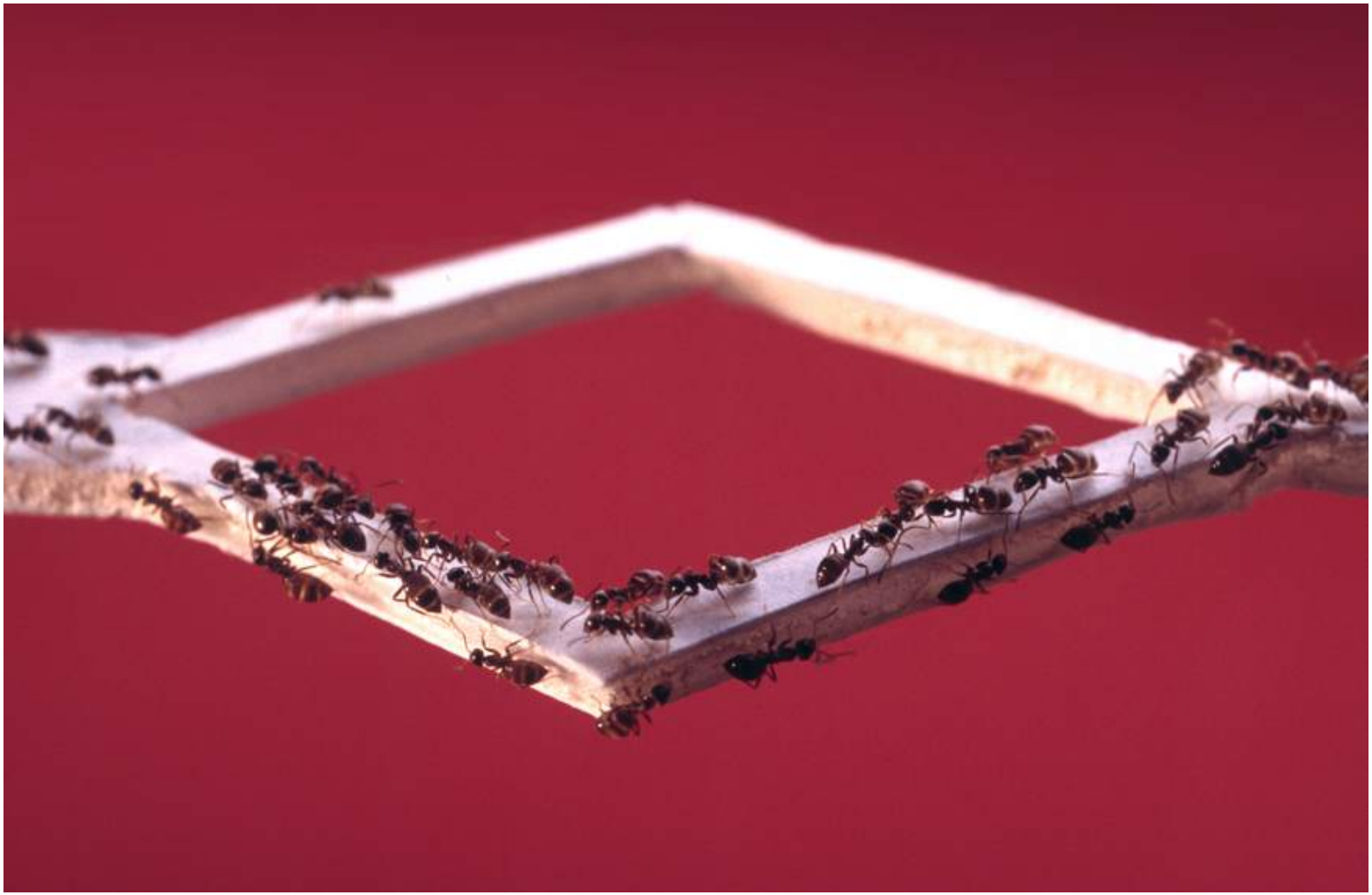
Much active research on hybrids between
exact and approximate algorithms!

Design choices and parameters everywhere

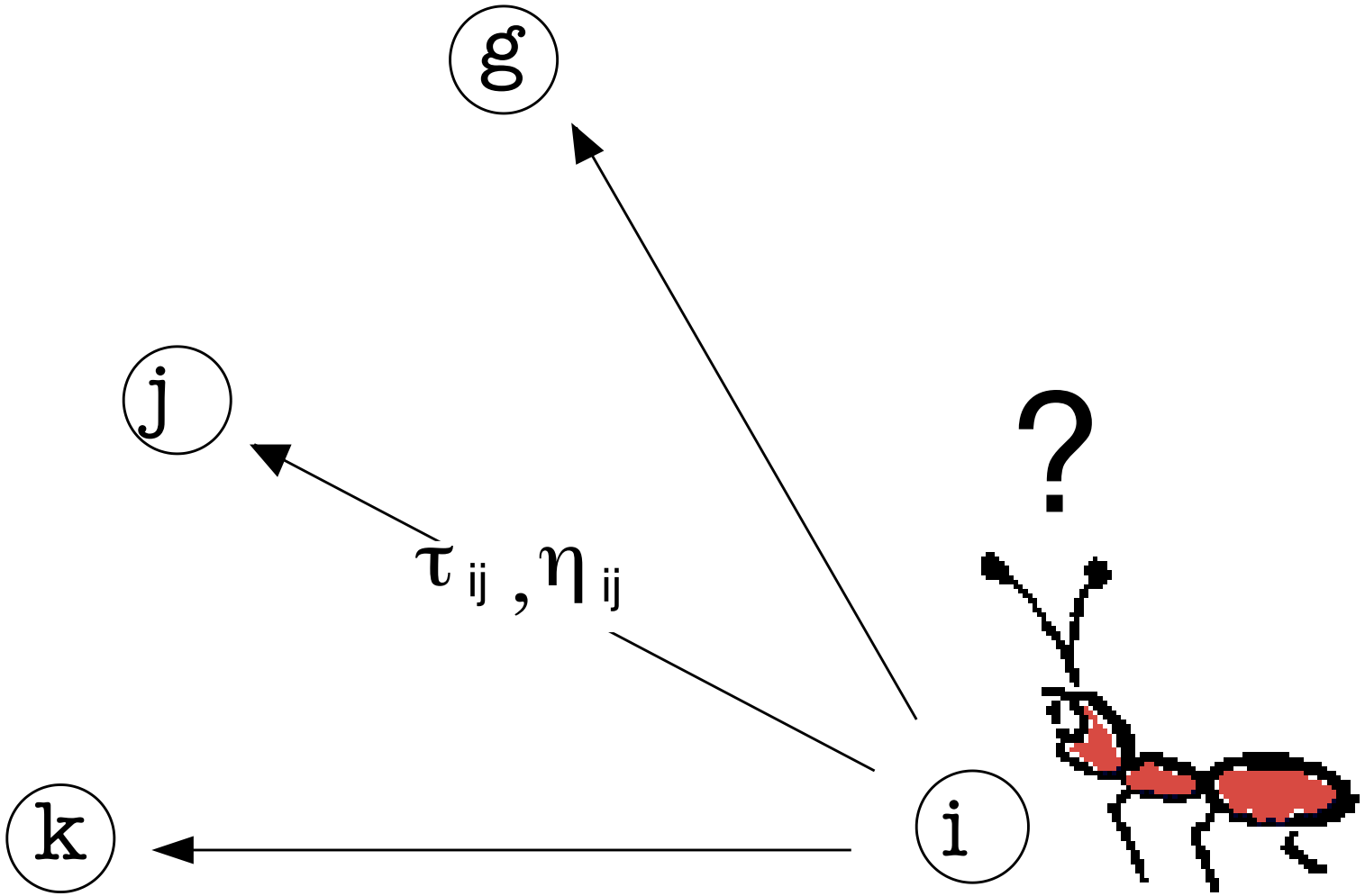
Todays high-performance optimizers involve a large number of design choices and parameter settings

- exact solvers
  - design choices include alternative models, pre-processing, variable selection, value selection, branching rules ...
  - many design choices have associated numerical parameters
  - example: SCIP 3.0.1 solver (fastest non-commercial MIP solver) has more than 200 relevant parameters that influence the solver's search mechanism
- approximate algorithms
  - design choices include solution representation, operators, neighborhoods, pre-processing, strategies, ...
  - many design choices have associated numerical parameters
  - example: multi-objective ACO algorithms with 22 parameters (plus several still hidden ones)

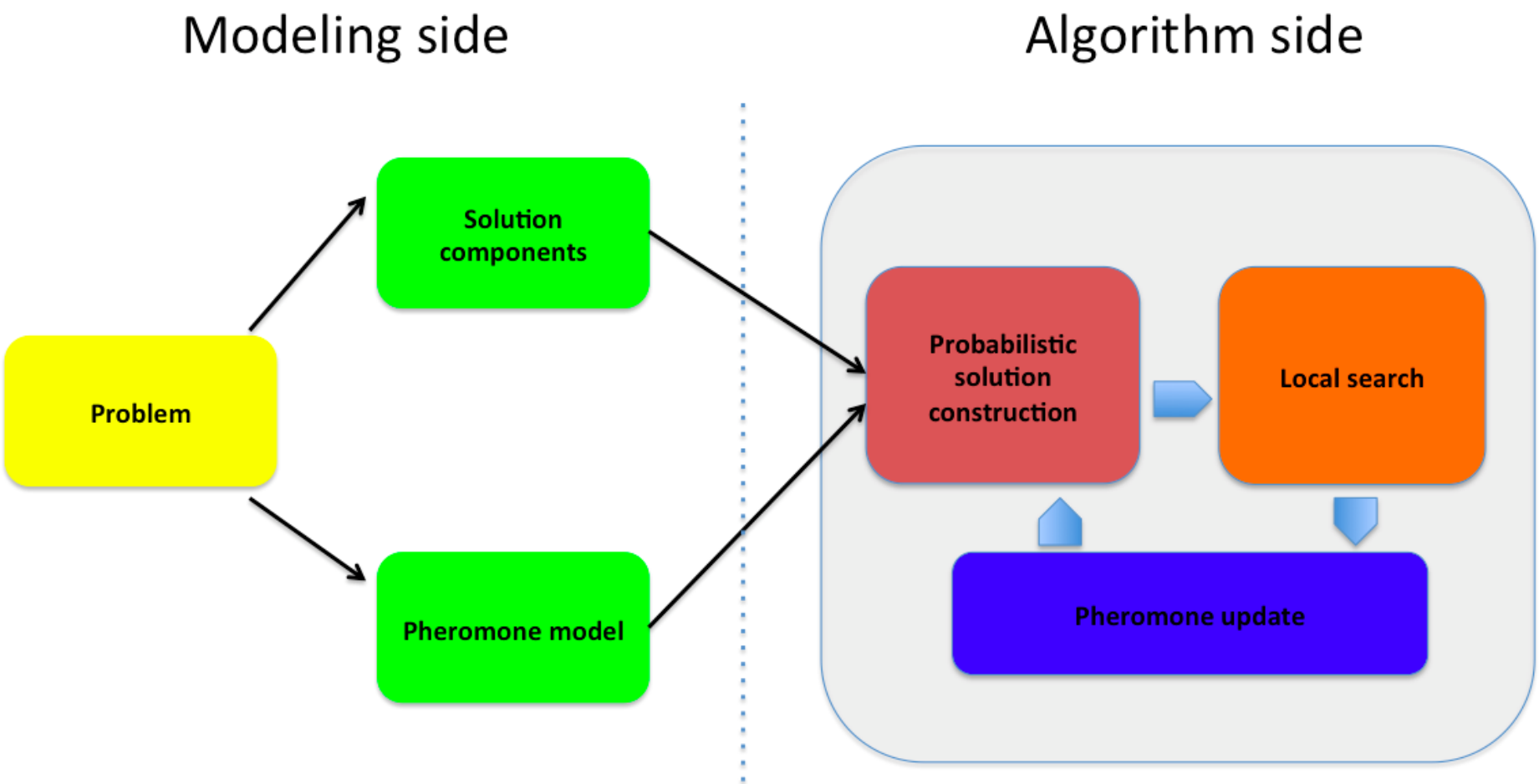
Example: Ant Colony Optimization



ACO, Probabilistic solution construction



Applying Ant Colony Optimization



ACO design choices and numerical parameters

- solution construction
  - choice of constructive procedure
  - choice of pheromone model
  - choice of heuristic information
  - numerical parameters
    - α, β influence the weight of pheromone and heuristic information, respectively
    - q_0 determines greediness of construction procedure
    - m, the number of ants
- pheromone update
  - which ants deposit pheromone and how much?
  - numerical parameters
    - ρ: evaporation rate
    - τ_0: initial pheromone level
- local search
  - . . . many more . . .

Parameter types

- *categorical* parameters **design**
  - choice of constructive procedure, choice of recombination operator, choice of branching strategy,...
- *ordinal* parameters **design**
  - neighborhoods, lower bounds, ...
- *numerical* parameters **tuning, calibration**
  - integer or real-valued parameters
  - weighting factors, population sizes, temperature, hidden constants, ...
  - numerical parameters may be *conditional* to specific values of categorical or ordinal parameters

Design and configuration of algorithms involves setting categorical, ordinal, **and** *numerical parameters*

Designing optimization algorithms

Challenges

- many alternative design choices
- nonlinear interactions among algorithm components and/or parameters
- performance assessment is difficult

Traditional design approach

- trial–and–error design guided by expertise/intuition
  $\rightsquigarrow$ prone to over-generalizations, implicit independence assumptions, limited exploration of design alternatives

Can we make this approach more principled and automatic?

Towards automatic algorithm configuration

Automated algorithm configuration

- apply powerful search techniques to design algorithms
- use computation power to explore design spaces
- assist algorithm designer in the design process
- free human creativity for higher level tasks

Offline configuration and online parameter control

Offline configuration

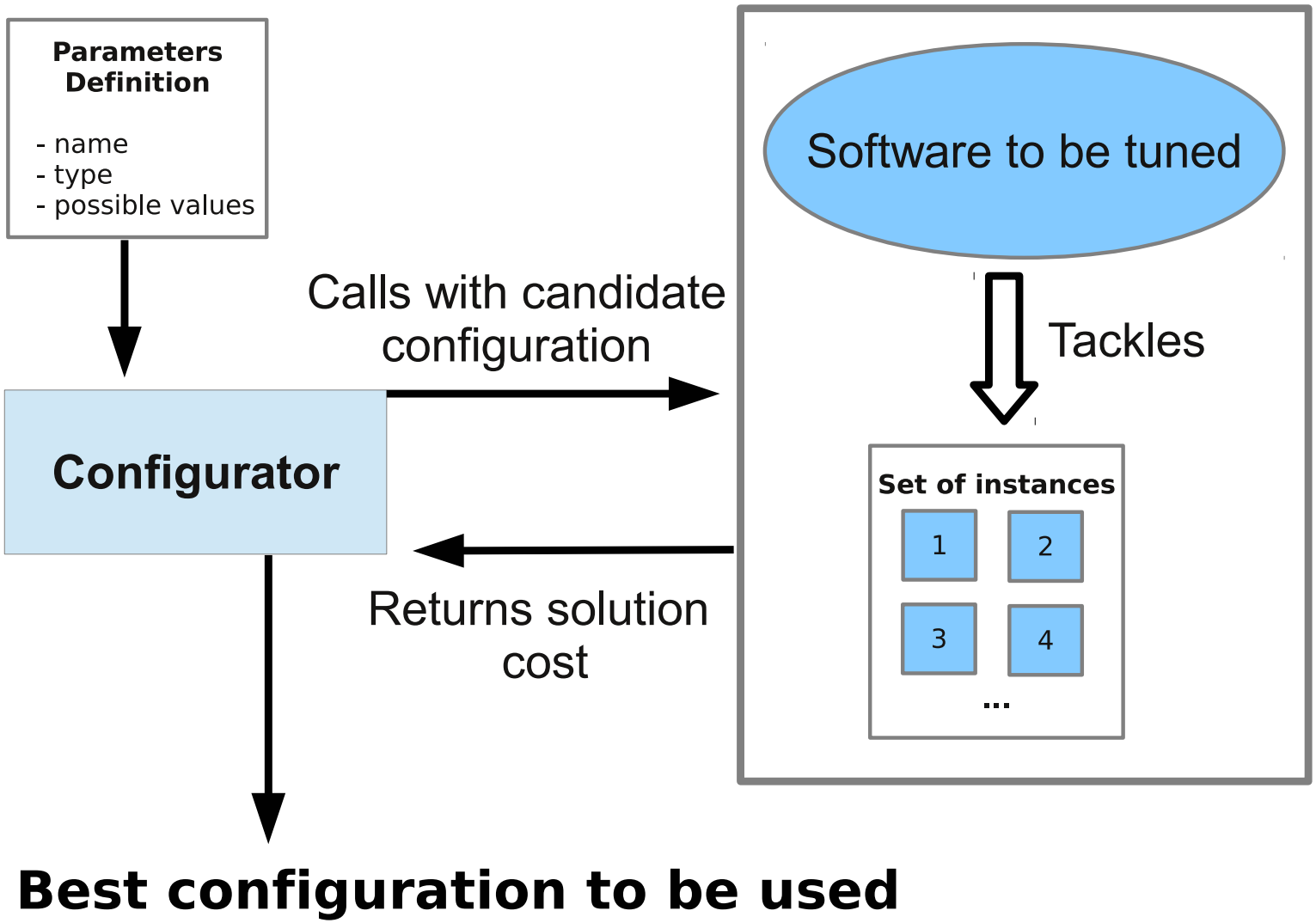
- configure algorithm before deploying it
- configuration on training instances
- related to algorithm design

Online parameter control

- adapt parameter setting while solving an instance
- typically limited to a set of known crucial algorithm parameters
- related to parameter calibration

Offline configuration techniques can be helpful to configure (online) parameter control strategies

Offline configuration



Configurators

Approaches to configuration

- experimental design techniques
  - e.g. CALIBRA [Adenso–Díaz, Laguna, 2006], [Ridge&Kudenko, 2007], [Coy et al., 2001], [Ruiz, Stützle, 2005]
- numerical optimization techniques
  - e.g. MADS [Audet&Orban, 2006], various [Yuan et al., 2012]
- heuristic search methods
  - e.g. meta-GA [Grefenstette, 1985], ParamILS [Hutter et al., 2007, 2009], gender-based GA [Ansótegui at al., 2009], linear GP [Oltean, 2005], REVAC(++) [Eiben et al., 2007, 2009, 2010] ...
- model-based optimization approaches
  - e.g. SPO [Bartz-Beielstein et al., 2005, 2006, ..], SMAC [Hutter et al., 2011, ..], GGA++ [Ansótegui, 2015]
- sequential statistical testing
  - e.g. F-race, iterated F-race [Birattari et al, 2002, 2007, ...]

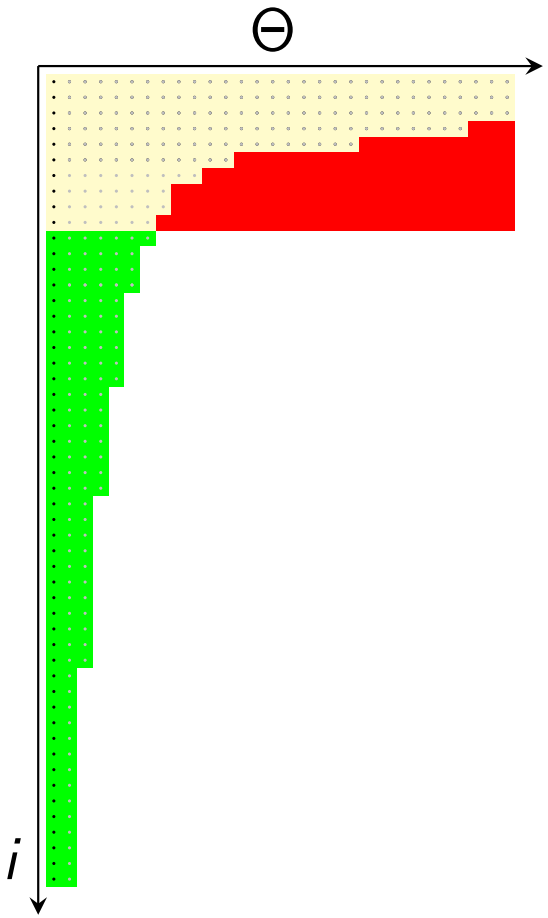
General, domain-independent methods required: (i) applicable to all variable types, (ii) multiple training instances, (iii) high performance, (iv) scalable

Approaches to configuration

- experimental design techniques
  - e.g. CALIBRA [Adenso–Díaz, Laguna, 2006], [Ridge&Kudenko, 2007], [Coy et al., 2001], [Ruiz, Stützle, 2005]
- numerical optimization techniques
  - e.g. MADS [Audet&Orban, 2006], various [Yuan et al., 2012]
- heuristic search methods
  - e.g. meta-GA [Grefenstette, 1985], *ParamILS* [Hutter et al., 2007, 2009], *gender-based GA* [Ansótegui at al., 2009], linear GP [Oltean, 2005], REVAC(++) [Eiben et al., 2007, 2009, 2010] ...
- model-based optimization approaches
  - e.g. SPO [Bartz-Beielstein et al., 2005, 2006, ..], SMAC [Hutter et al., 2011, ..], GGA++ [Ansótegui, 2015]
- sequential statistical testing
  - e.g. F-race, *iterated F-race* [Birattari et al, 2002, 2007, ...]

General, domain-independent methods required: (i) applicable to all variable types, (ii) multiple training instances, (iii) high performance, (iv) scalable

The racing approach



- start with a set of initial candidates
- consider a *stream* of instances
- sequentially evaluate candidates
- discard inferior candidates

  as sufficient evidence is gathered against them
- . . . repeat until a winner is selected

  or until computation time expires

The F-Race algorithm

Statistical testing

1. family-wise tests for differences among configurations
   - Friedman two-way analysis of variance by ranks
2. if Friedman rejects H_0, perform pairwise comparisons to best configuration
   - apply Friedman post-test




Some applications of F-race

International time-tabling competition

- winning algorithm configured by F-race [Chiarandini et al., 2006]
- interactive injection of new configurations

Vehicle routing and scheduling problem

- first industrial application
- improved commerialized algorithm [Becker et al., 2005]

F-race in stochastic optimization

- evaluate "neighbours" using F-race
  (solution cost is a random variable!)
- good performance if variance of solution cost is high
  [Birattari et al., 2006]

Iterated race

Racing is a method for the *selection of the best* configuration and independent of the way the set of configurations is sampled

Iterated race

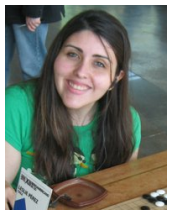
sample configurations from initial distribution

While not terminate()

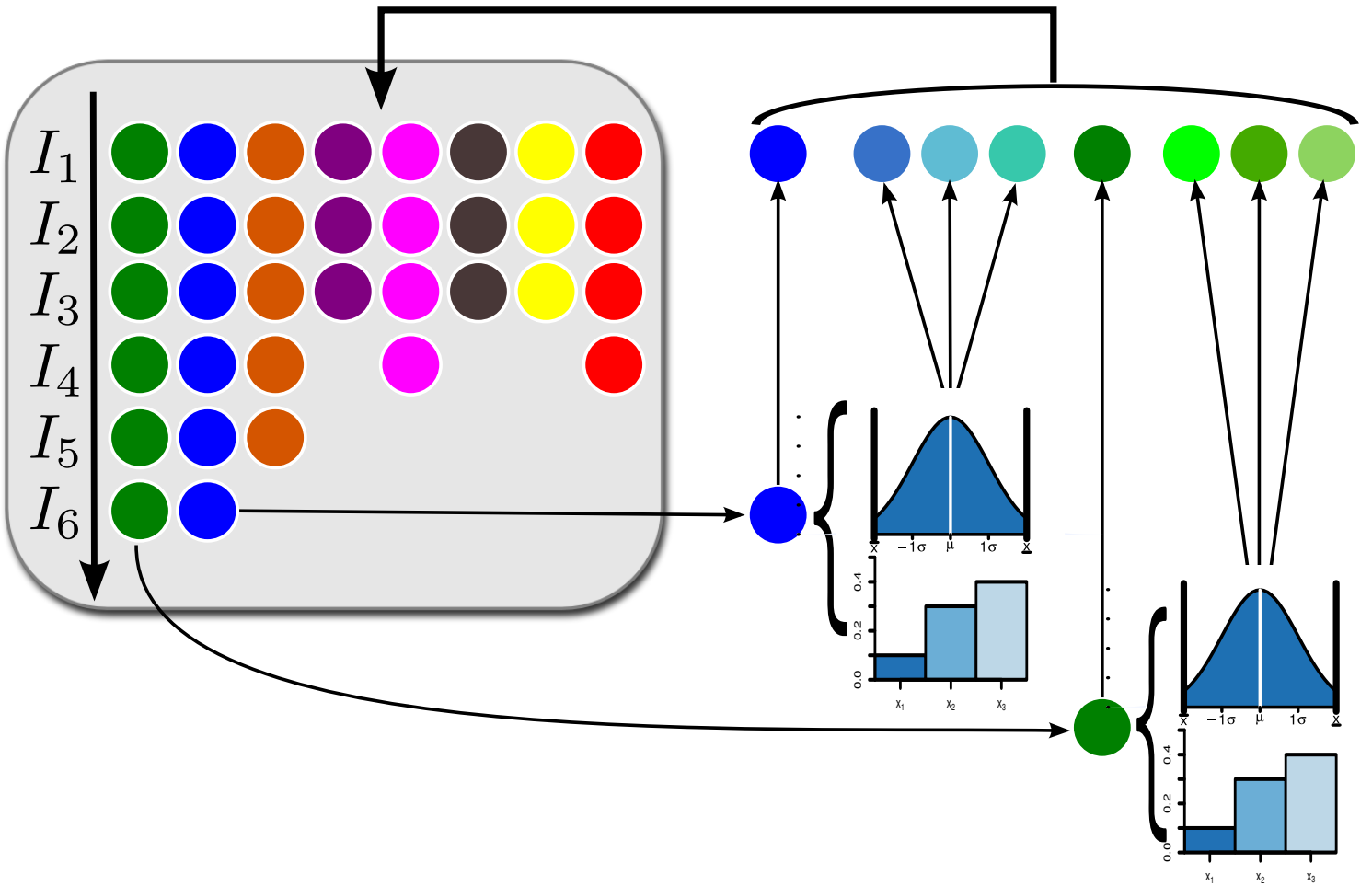
apply race
modify sampling distribution
sample configurations






The irace package: sampling



Iterated racing: sampling distributions

Numerical parameter $X_d \in [\underline{x}_d, \overline{x}_d]$

$\Rightarrow$ *Truncated normal distribution*

$$\mathcal{N}(\mu_d^z, \sigma_d^i) \in [\underline{x}_d, \overline{x}_d]$$

μ_d^z = value of parameter d in elite configuration z

σ_d^i = decreases with the number of iterations

Categorical parameter $X_d \in \{x_1, x_2, \ldots, x_{n_d}\}$

$\Rightarrow$ *Discrete probability distribution*

$$\Pr^z\{X_d = x_j\} =$$

x_1	x_2	$\ldots$	x_{n_d}
0.1	0.3	$\ldots$	0.4

- Updated by increasing probability of parameter value in elite configuration
- Other probabilities are reduced

The irace package

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. **The irace package, Iterated Race for Automatic Algorithm Configuration.** *Technical Report TR/IRIDIA/2011-004*, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

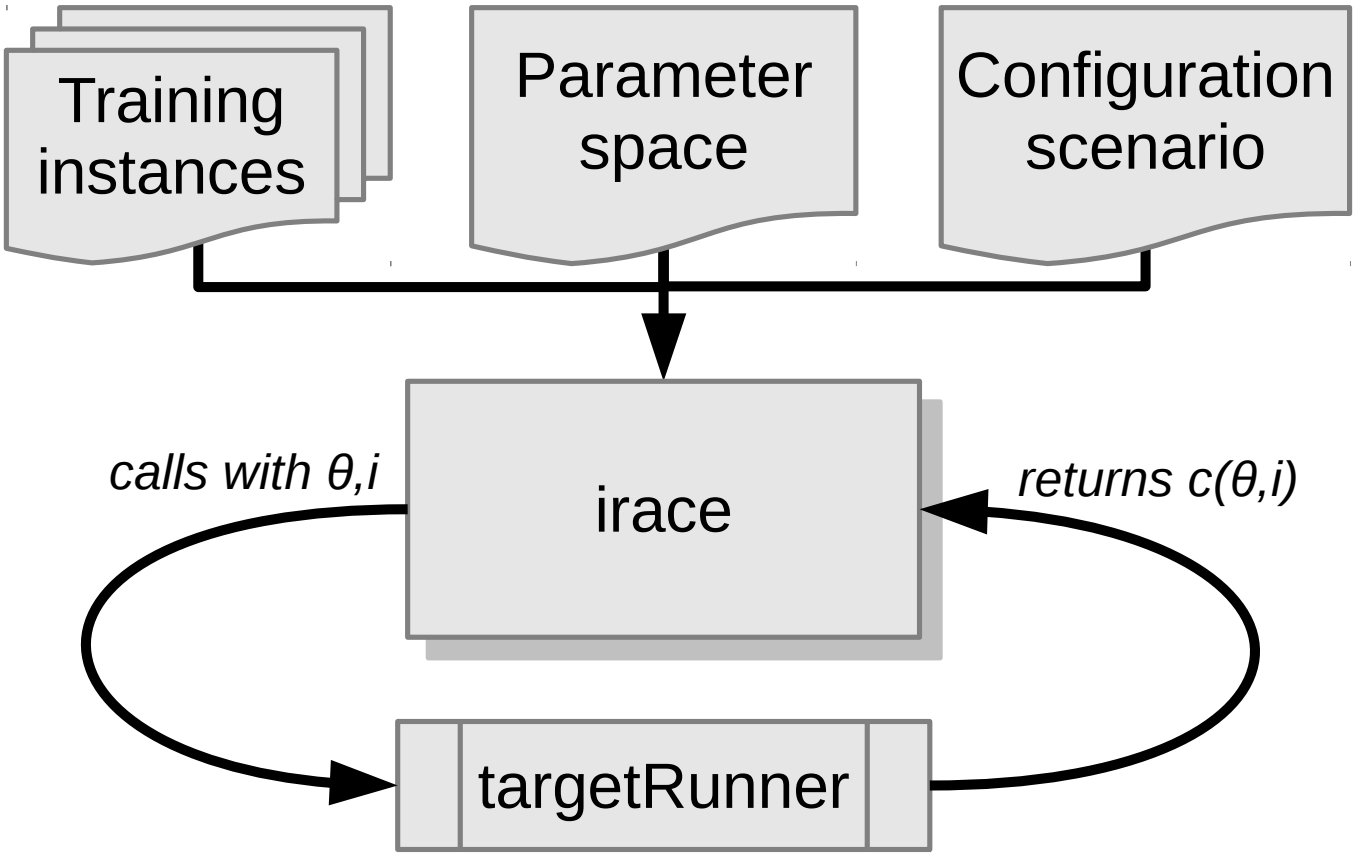
The irace Package: User Guide, 2016, *Technical Report TR/IRIDIA/2016-004*

http://iridia.ulb.ac.be/irace

- implementation of Iterated Racing in R
  - Goal 1: flexible
  - Goal 2: easy to use
- but no knowledge of R necessary
- parallel evaluation (MPI, multi-cores, grid engine ..)
- initial candidates
- forbidden configurations

irace has shown to be effective for configuration tasks with several hundred of variables

The irace package: usage



Example application of irace: ACOTSP

- Thomas Stützle. **ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem**, 2002.
  `http://www.aco-metaheuristic.org/aco-code/`

- ACOTSP: ant colony optimization algorithms for the TSP

  Command-line program:

```
$ ./acotsp -i instance -t 20 --mmas --ants 10 --rho 0.95 ...
```

Goal: find best parameter settings of ACOTSP for solving random Euclidean TSP instances with $n \in [500, 5000]$ within 20 CPU-seconds

Example application of irace: ACOTSP

```
$ cat parameters-acotsp.txt

# name       switch              type  values     conditions
algorithm    "--"                c (as,mmas,eas,ras,acs)
localsearch  "--localsearch "    c (0, 1, 2, 3)
alpha        "--alpha "          r (0.00, 5.00)
beta         "--beta "           r (0.00, 10.00)
rho          "--rho  "           r (0.01, 1.00)
ants         "--ants "           i (5, 100)
q0           "--q0 "             r (0.0, 1.0) | algorithm == "acs"
rasrank      "--rasranks "       i (1, 100)   | algorithm == "ras"
elitistants  "--elitistants "    i (1, 750)   | algorithm == "eas"
nnls         "--nnls "           i (5, 50)    | localsearch %in% c(1,2,3)
dlb          "--dlb "            c (0, 1)     | localsearch %in% c(1,2,3)
```

Example application of irace: ACOTSP

```
$ cat targetRunner
```

```
#!/bin/bash
INSTANCE=$1
CANDIDATENUM=$2
CAND_PARAMS=$*
STDOUT="c${CANDIDATENUM}.stdout"
FIXED_PARAMS=" --time 1 --tries 1 --quiet "
acotsp $FIXED_PARAMS -i $INSTANCE $CAND_PARAMS 1> $STDOUT
COST=$(grep -oE 'Best [-+0-9.e]+' $STDOUT |cut -d' ' -f2)
echo "${COST}"
exit 0
```

Example application of irace: ACOTSP

```
$ ls Instances/
$ cat tune-conf

instanceDir = "./Instances"
maxExperiments = 1000
digits = 2
```

✔ Good to go:

```
$ irace --parallel 2 --debug-level 1
```

- `--parallel` to execute in parallel
- `--debug-level` to see what irace is executing

Example application of irace: ACOTSP and more

- Initial configurations:

  $ cat default.txt

```
algorithm localsearch alpha beta rho  ants nnls dlb q0
as        0           1.0   1.0  0.95 10   NA   NA  NA
```

- Logical expressions that forbid configurations:

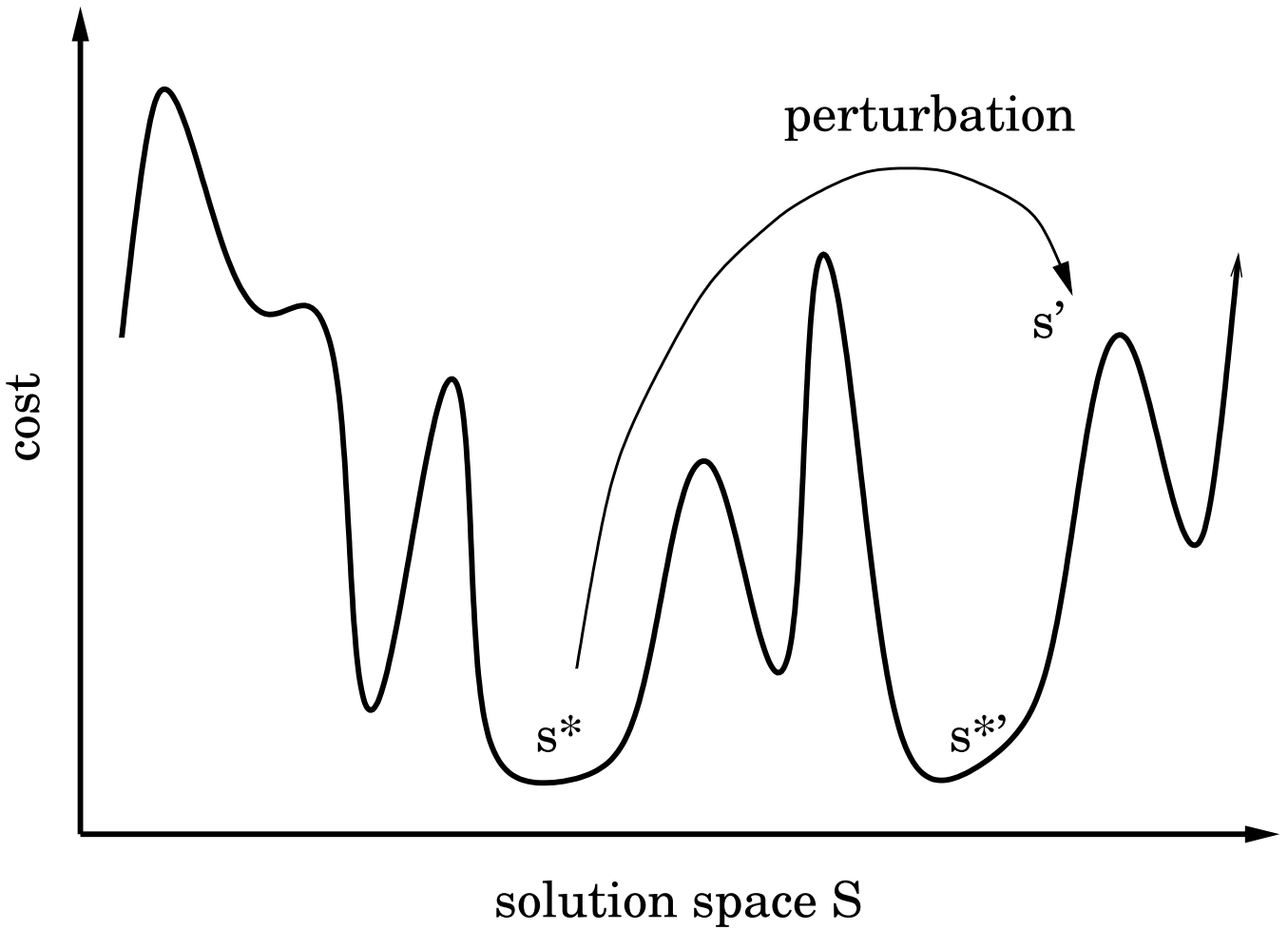
  $ cat forbidden.txt

```
(alpha == 0.0) & (beta == 0.0)
```

Other configurators: ParamILS, SMAC

ParamILS Framework

ParamILS is an iterated local search method that works in the parameter space



Main design choices for ParamILS

Parameter encoding

- only categorical parameters, numerical parameters need to be discretized

Initialization

- select best configuration among default and several random configurations

Local search

- 1-exchange neighborhood search in random order

Perturbation

- change several randomly chosen parameters

Acceptance criterion

- always select the better configuration

Main design choices for ParamILS

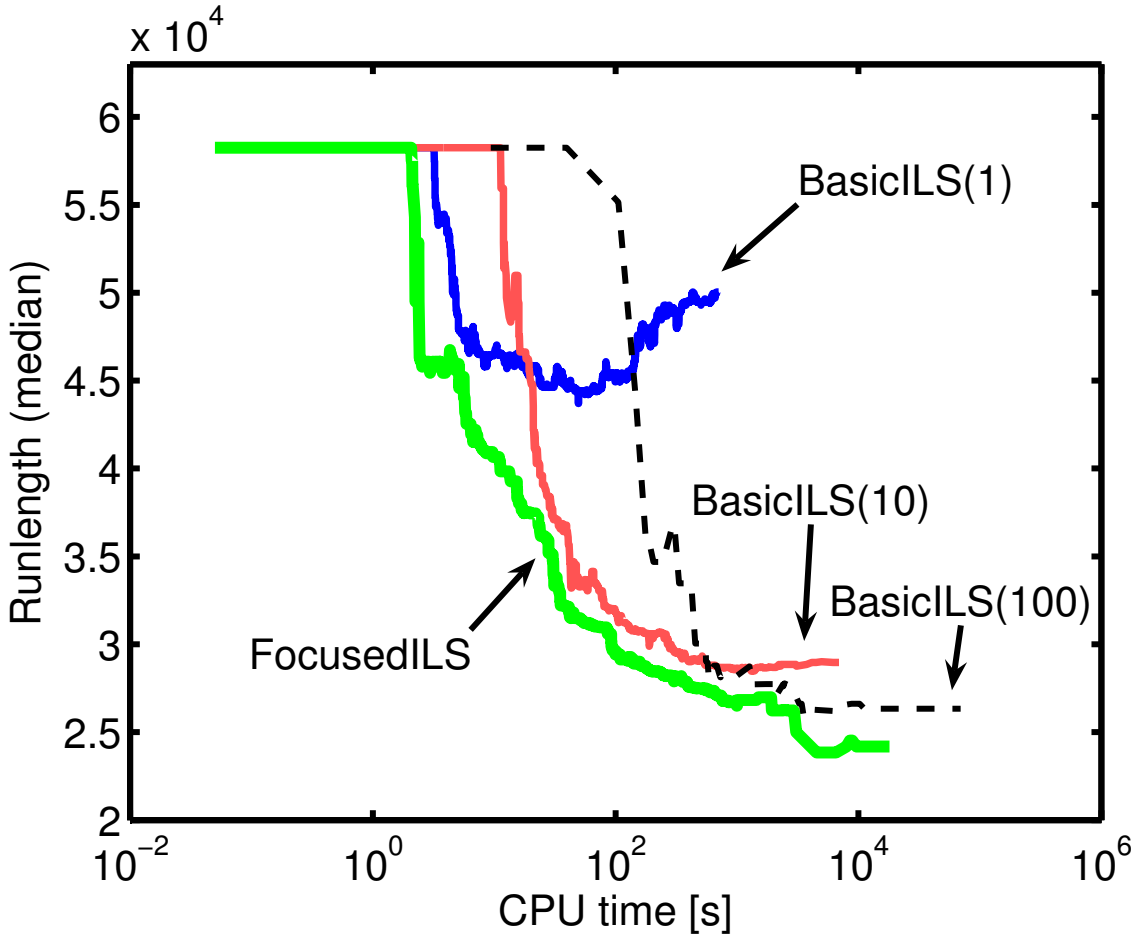
Evaluation of incumbent

- *BasicILS*: each configuration is evaluated on the same number of N instances
- *FocusedILS*: the number of instances on which the best configuration is evaluated increases at run time (intensification)

Adaptive capping

- mechanism for early pruning the evaluation of poor candidate configurations
- particularly effective when configuring algorithms for minimization of computation time

ParamILS: BasicILS vs. FocusedILS



example: comparison of BasicILS and FocusedILS for configuring the SAPS solver for SAT-encoded quasi-group with holes, taken from [Hutter et al., 2007]

Model-based Approaches (SPOT, SMAC)

Idea: Use surrogate model $\mathcal{M}$ to predict performance of configurations

Algorithmic scheme

generate and evaluate initial set of configurations Θ_0

choose best-so-far configuration $\theta^* \in \Theta_0$

while tuning budget available

- learn surrogate model $\mathcal{M}: \Theta \mapsto R$
- use model $\mathcal{M}$ to generate promising configurations Θ_p
- evaluate configurations in Θ_p
- $\Theta_0 := \Theta_0 \cup \Theta_p$
- update $\theta^* \in \Theta_0$

end

output: θ^*

Sequential model-based algorithm configuration (SMAC)

[Hutter et al., 2011]

SMAC extends surrogate model-based configuration to complex algorithm configuration tasks and across multiple instances

Main design decisions

- Random forests for $\mathcal{M} \Rightarrow$ categorical & numerical parameters
- Aggregate predictions from $\mathcal{M}_i$ for each instance i
- Local search on the surrogate model surface (EIC)
  $\Rightarrow$ promising configurations
- Instance features $\Rightarrow$ improve performance predictions
- Intensification mechanism (inspired by FocusedILS)
- Further extensions $\Rightarrow$ capping

Applications

Applications of automatic configuration tools

- configuration of "black-box" solvers
  - e.g. mixed integer programming solvers, continuous optimizers
- supporting tool in algorithm engineering
  - e.g. metaheuristics for probabilistic TSP, re-engineering PSO
- bottom-up generation of heuristic algorithms
  - e.g. heuristics for SAT, FSP, etc.; metaheuristic framework
- design configurable algorithm frameworks
  - e.g. Satenstein, MOACO, UACOR, MOEAs

Example, configuration of "black-box" solvers

Mixed-integer programming solvers



Mixed integer programming (MIP) solvers

[Hutter, Hoos, Leyton-Brown, Stützle, 2009, Hutter, Hoos Leyton-Brown, 2010]

- powerful commercial (e.g. CPLEX) and non-commercial (e.g. SCIP) solvers available
- large number of parameters (tens to hundreds)

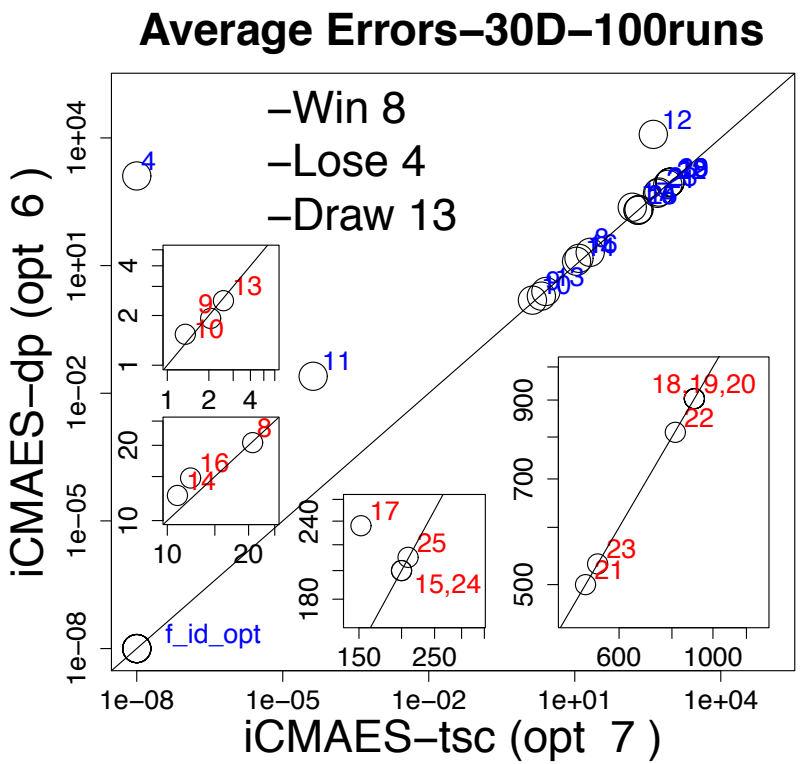
- default configurations not necessarily best for specific problems

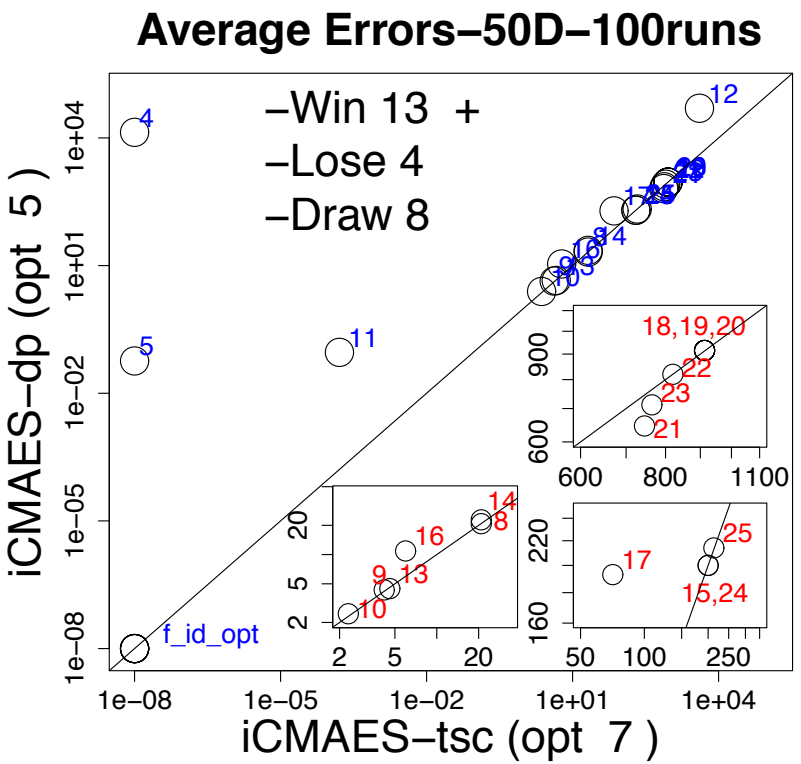
Benchmark set	Default	Configured	Speedup
`Regions200`	72	10.5 (11.4 ± 0.9)	*6.8*
`Conic.SCH`	5.37	2.14 (2.4 ± 0.29)	*2.51*
`CLS`	712	23.4 (327 ± 860)	*30.43*
`MIK`	64.8	1.19 (301 ± 948)	*54.54*
`QP`	969	525 (827 ± 306)	*1.85*

FocusedILS tuning CPLEX, 10 runs, 2 CPU days, 63 parameters

Tune known algorithms; example IPOP-CMAES

- IPOP-CMAES is state-of-the-art continuous optimizer
- configuration done on benchmark problems (instances) distinct from test set (CEC'05 benchmark function set) using seven numerical parameters





Example, supporting tool in algorithm engineering

Tuning in-the-loop (re)design of continuous optimizers



Tuning in-the-loop (re)design of continuous optimizers

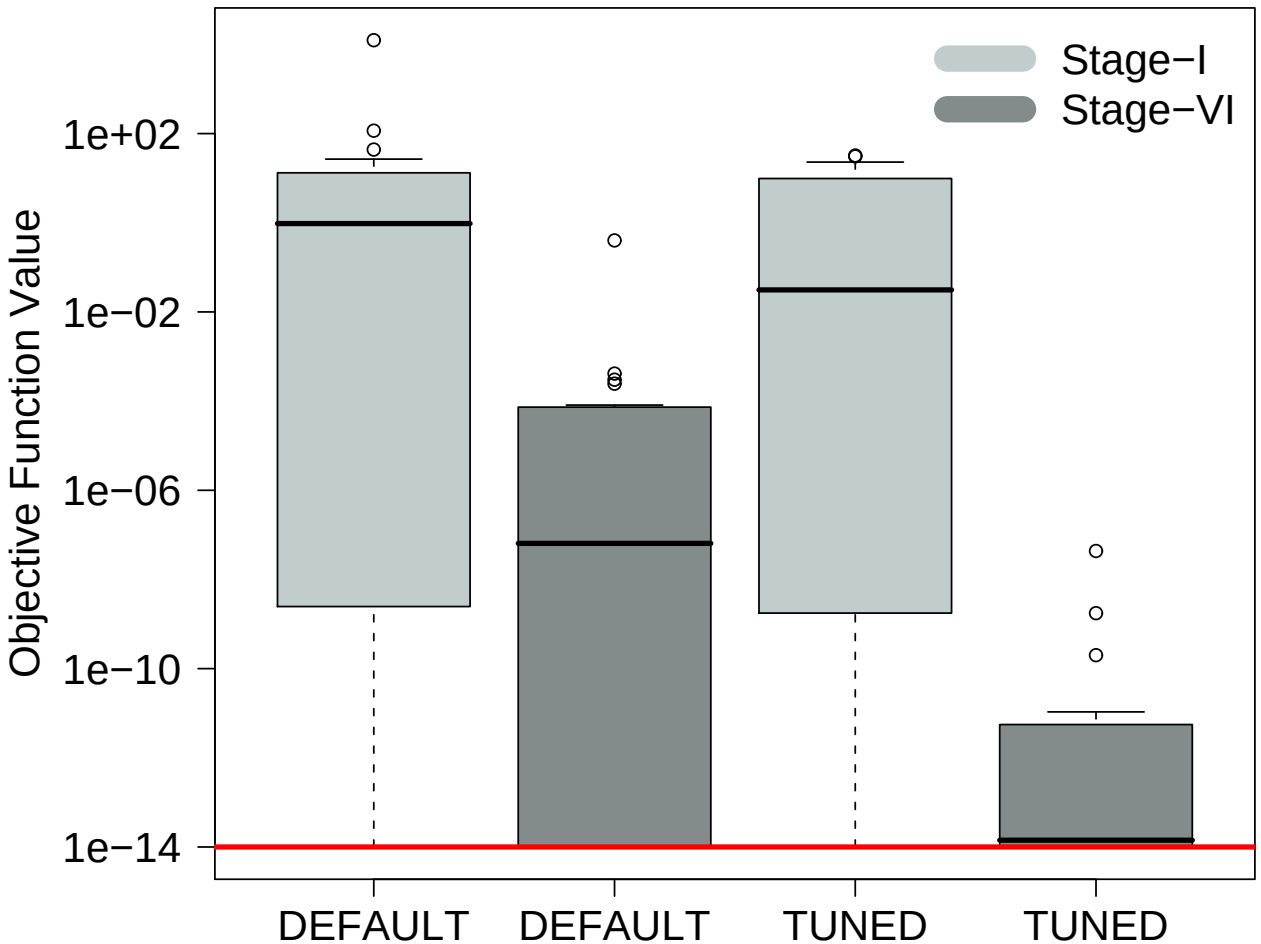
[Montes de Oca, Aydın, Stützle, 2011]

- re-design of an incremental PSO algorithm for large-scale continuous optimization
- six steps
  - local search, call and control strategy of LS, PSO rules, bound constraint handling, stagnation handling, restarts
- iterated F-race used at each step to configure up to 10 parameters
- configuration done on 19 functions of dimension 10
- scaling examined until dimension 1000

configuration results can help designer to gain insight useful for further development

Tuning in-the-loop (re)design of continuous optimizers

[Montes de Oca, Aydın, Stützle, 2011]



Example, bottom-up generation of algorithms

Automatic design of hybrid SLS algorithms

Automatic design of hybrid SLS algorithms

[Marmion, Mascia, Lópes-Ibáñez, Stützle, 2013]

Approach

- decompose single-point SLS methods into components
- derive generalized metaheuristic structure
- component-wise implementation of metaheuristic part

Implementation

- present possible algorithm compositions by a grammar
- instantiate grammer using a parametric representation
  - allows use of standard automatic configuration tools
  - shows good performance when compared to, e.g., grammatical evolution [Mascia, Lópes-Ibáñez, Dubois-Lacoste, Stützle, 2014]

General Local Search Structure: ILS

s_0 :=initSolution
s^* := ls(s_0)
  repeat
    s' :=perturb(s^*,*history*)
    $s^{*\prime}$:=ls(s')
    s^* :=accept(s^*,$s^{*\prime}$,*history*)
  until termination criterion met

- many SLS methods instantiable from this structure
- abilities
  - hybridization
  - recursion
  - problem specific implementation at low-level
  - separation of generic and problem-specific components

Example instantiations of some metaheuristics

	perturb	*ls*	*accept*
SA	random move	$\emptyset$	Metropolis
PII	random move	$\emptyset$	Metropolis, fixed T
TS	$\emptyset$	TS	$\emptyset$
ILS	any	any	any
IG	destruct/construct	any	any
GRASP	rand. greedy sol.	any	$\emptyset$

Grammar

```
     <algorithm> ::= <initialization> <ils>
<initialization> ::= random | <pbs_initialization>
           <ils> ::= ILS(<perturb>, <ls>, <accept>, <stop>)

<perturb> ::= none | <initialization> | <pbs_perturb>
     <ls> ::= <ils> | <descent> | <sa> | <rii> | <pii> | <vns> | <ig> | <pbs_ls>
 <accept> ::= alwaysAccept | improvingAccept  <comparator>
            | prob(<value_prob_accept>) | probRandom | <metropolis>
            | threshold(<value_threshold_accept>) | <pbs_accept>

<descent> ::= bestDescent(<comparator>, <stop>)
            | firstImprDescent(<comparator>, <stop>)
    <sa>  ::= ILS(<pbs_move>, no_ls,  <metropolis>, <stop>)
    <rii> ::= ILS(<pbs_move>, no_ls, probRandom, <stop>)
    <pii> ::= ILS(<pbs_move>, no_ls, prob(<value_prob_accept>), <stop>)
    <vns> ::= ILS(<pbs_variable_move>, firstImprDescent(improvingStrictly),
                  improvingAccept(improvingStrictly), <stop>)
     <ig> ::= ILS(<deconst-construct_perturb>, <ls>, <accept>, <stop>)

<comparator> ::= improvingStrictly | improving
<value_prob_accept> ::= [0, 1]
<value_threshold_accept> ::= [0, 1]
<metropolis> ::= metropolisAccept(<init_temperature>, <final_temperature>,
                                  <decreasing_temperature_ratio>, <span>)
 <init_temperature> ::= {1, 2,..., 10000}
<final_temperature> ::= {1, 2,..., 100}
<decreasing_temperature_ratio> ::= [0, 1]
<span> ::= {1, 2,..., 10000}
```

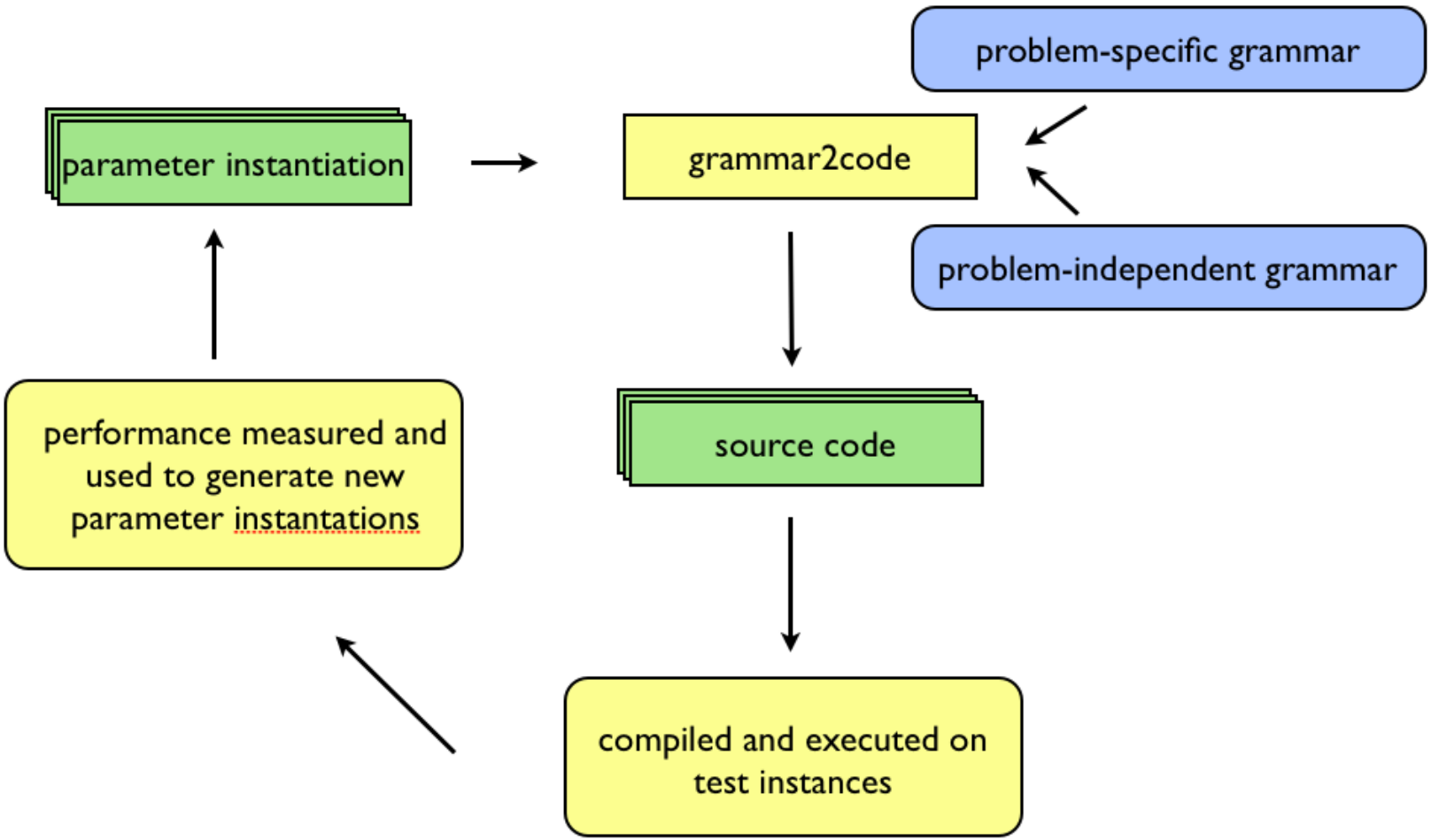
Grammar

```
     <algorithm> ::= <initialization> <ils>
<initialization> ::= random | <pbs_initialization>
           <ils> ::= ILS(<perturb>, <ls>, <accept>, <stop>)

<perturb> ::= none | <initialization> | <pbs_perturb>
     <ls> ::= <ils> | <descent> | <sa> | <rii> | <pii> | <vns> | <ig> | <pbs_ls>
 <accept> ::= alwaysAccept | improvingAccept  <comparator>
            | prob(<value_prob_accept>) | probRandom | <metropolis>
            | threshold(<value_threshold_accept>) | <pbs_accept>

<descent> ::= bestDescent(<comparator>, <stop>)
            | firstImprDescent(<comparator>, <stop>)
    <sa>  ::= ILS(<pbs_move>, no_ls,  <metropolis>, <stop>)
    <rii> ::= ILS(<pbs_move>, no_ls, probRandom, <stop>)
    <pii> ::= ILS(<pbs_move>, no_ls, prob(<value_prob_accept>), <stop>)
    <vns> ::= ILS(<pbs_variable_move>, firstImprDescent(improvingStrictly),
                  improvingAccept(improvingStrictly), <stop>)
     <ig> ::= ILS(<deconst-construct_perturb>, <ls>, <accept>, <stop>)

<comparator> ::= improvingStrictly | improving
<value_prob_accept> ::= [0, 1]
<value_threshold_accept> ::= [0, 1]
<metropolis> ::= metropolisAccept(<init_temperature>, <final_temperature>,
                                  <decreasing_temperature_ratio>, <span>)
 <init_temperature> ::= {1, 2,..., 10000}
<final_temperature> ::= {1, 2,..., 100}
<decreasing_temperature_ratio> ::= [0, 1]
<span> ::= {1, 2,..., 10000}
```

Grammar

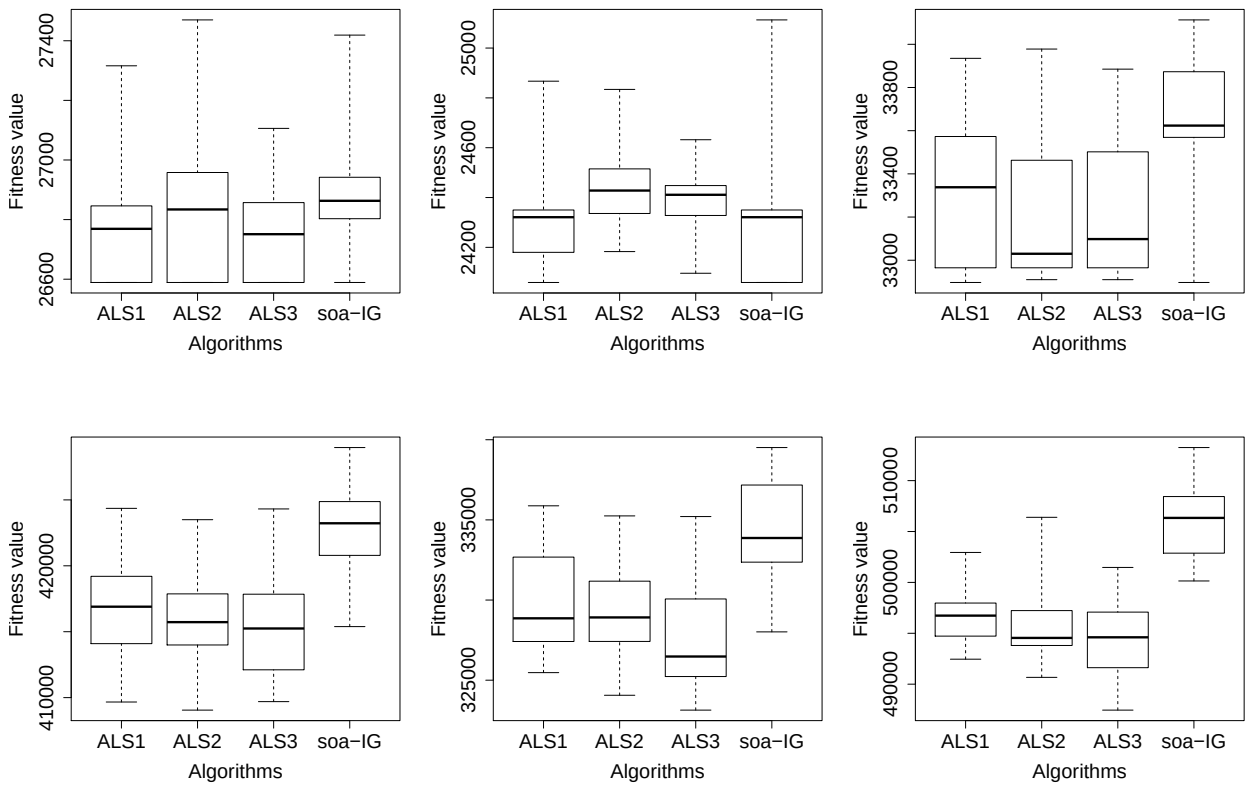
```
     <algorithm> ::= <initialization> <ils>
<initialization> ::= random | <pbs_initialization>
          <ils> ::= ILS(<perturb>, <ls>, <accept>, <stop>)
<perturb> ::= none | <initialization> | <pbs_perturb>
     <ls> ::= <ils> | <descent> | <sa> | <rii> | <pii> | <vns> | <ig> | <pbs_ls
 <accept> ::= alwaysAccept | improvingAccept  <comparator>
            | prob(<value_prob_accept>) | probRandom | <metropolis>
            | threshold(<value_threshold_accept>) | <pbs_accept>

<descent> ::= bestDescent(<comparator>, <stop>)
            | firstImprDescent(<comparator>, <stop>)
    <sa>  ::= ILS(<pbs_move>, no_ls,  <metropolis>, <stop>)
    <rii> ::= ILS(<pbs_move>, no_ls, probRandom, <stop>)
    <pii> ::= ILS(<pbs_move>, no_ls, prob(<value_prob_accept>), <stop>)
    <vns> ::= ILS(<pbs_variable_move>, firstImprDescent(improvingStrictly),
                  improvingAccept(improvingStrictly), <stop>)
     <ig> ::= ILS(<deconst-construct_perturb>, <ls>, <accept>, <stop>)

<comparator> ::= improvingStrictly | improving
<value_prob_accept> ::= [0, 1]
<value_threshold_accept> ::= [0, 1]
<metropolis> ::= metropolisAccept(<init_temperature>, <final_temperature>,
                                  <decreasing_temperature_ratio>, <span>)
 <init_temperature> ::= {1, 2,..., 10000}
<final_temperature> ::= {1, 2,..., 100}
<decreasing_temperature_ratio> ::= [0, 1]
<span> ::= {1, 2,..., 10000}
```

System overview



Flow-shop problem with weighted tardiness

- Automatic configuration:
  - 1, 2 or 3 levels of recursion (r)
  - 80, 127, and 174 parameters, respectively
  - budget: r $\times$ 10 000 trials each of 30 seconds



results are competitive or superior to state-of-the-art algorithm

Summary

Contributions

- approach to automate design and analysis of (hybrid) metaheuristics
- not a silver bullet, but needs right components, especially low-level problem-specific ones
- better or equal performance to state-of-the-art for PFSP-WT, UBQP, TSP-TW
- directly extendible for unbiased comparisons of metaheuristics

Future work

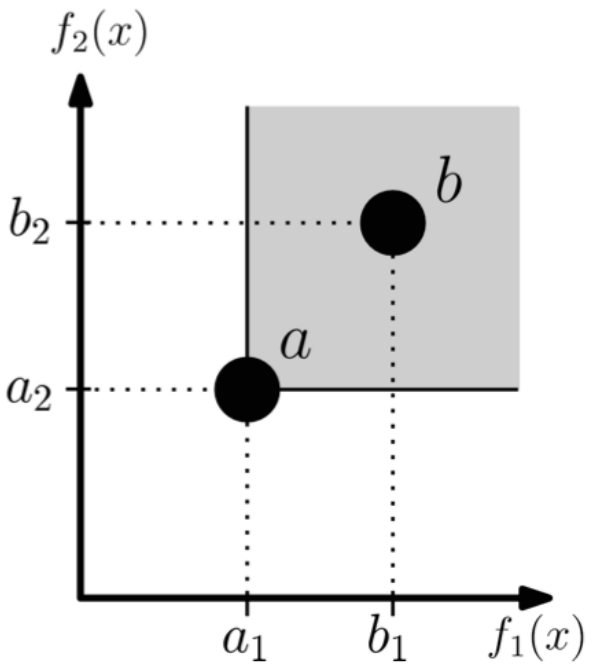
- extensions to other methods and templates
- dealing with complexity of hybrid algorithms
- increase generality to tackle full problem classes

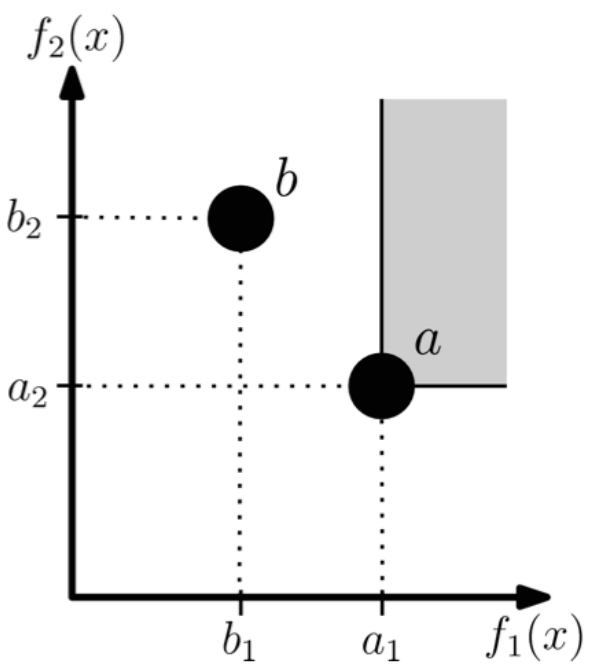
Example, design configurable algorithm framework

Multi-objective ant colony optimization (MOACO)


Multi-objective Optimization

- many real-life problems are multiobjective
- no *a priori* knowledge $\rightsquigarrow$ Pareto-optimality





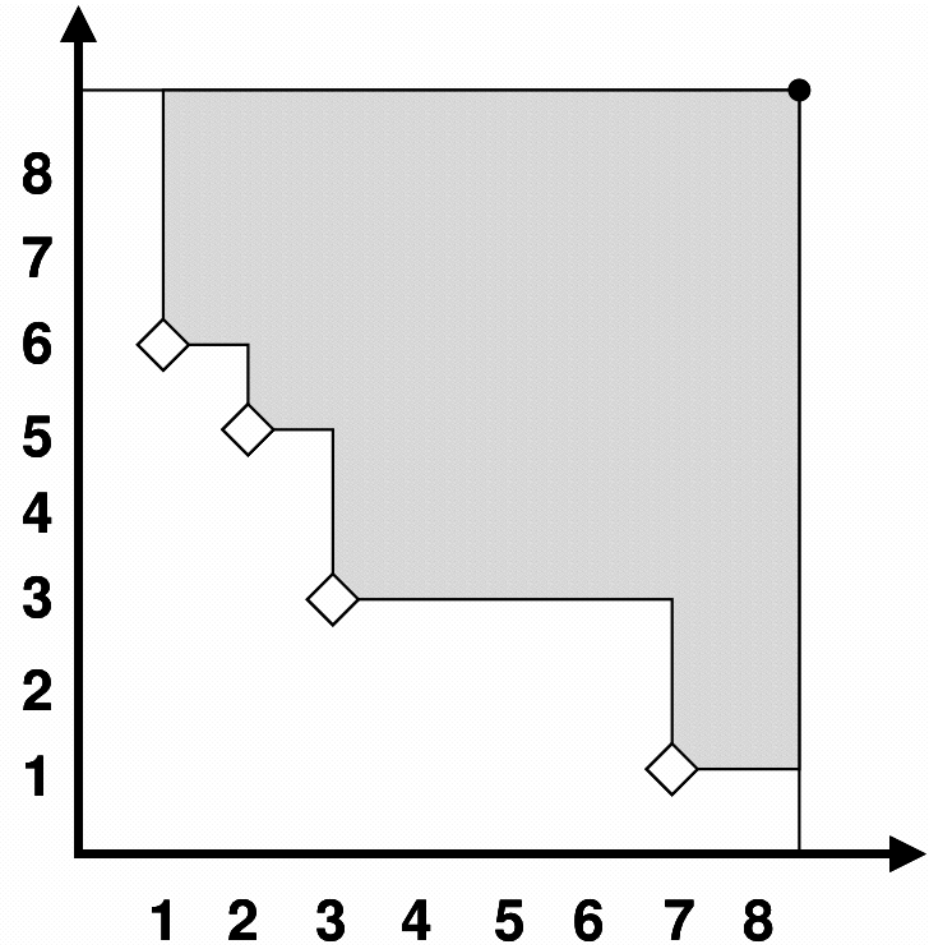
MOACO framework

López-Ibáñez, Stützle, 2012

- algorithm framework for multi-objective ACO algorithms
- can instantiate MOACO algorithms from literature
- 10 parameters control the multi-objective part
- 12 parameters control the underlying pure "ACO" part

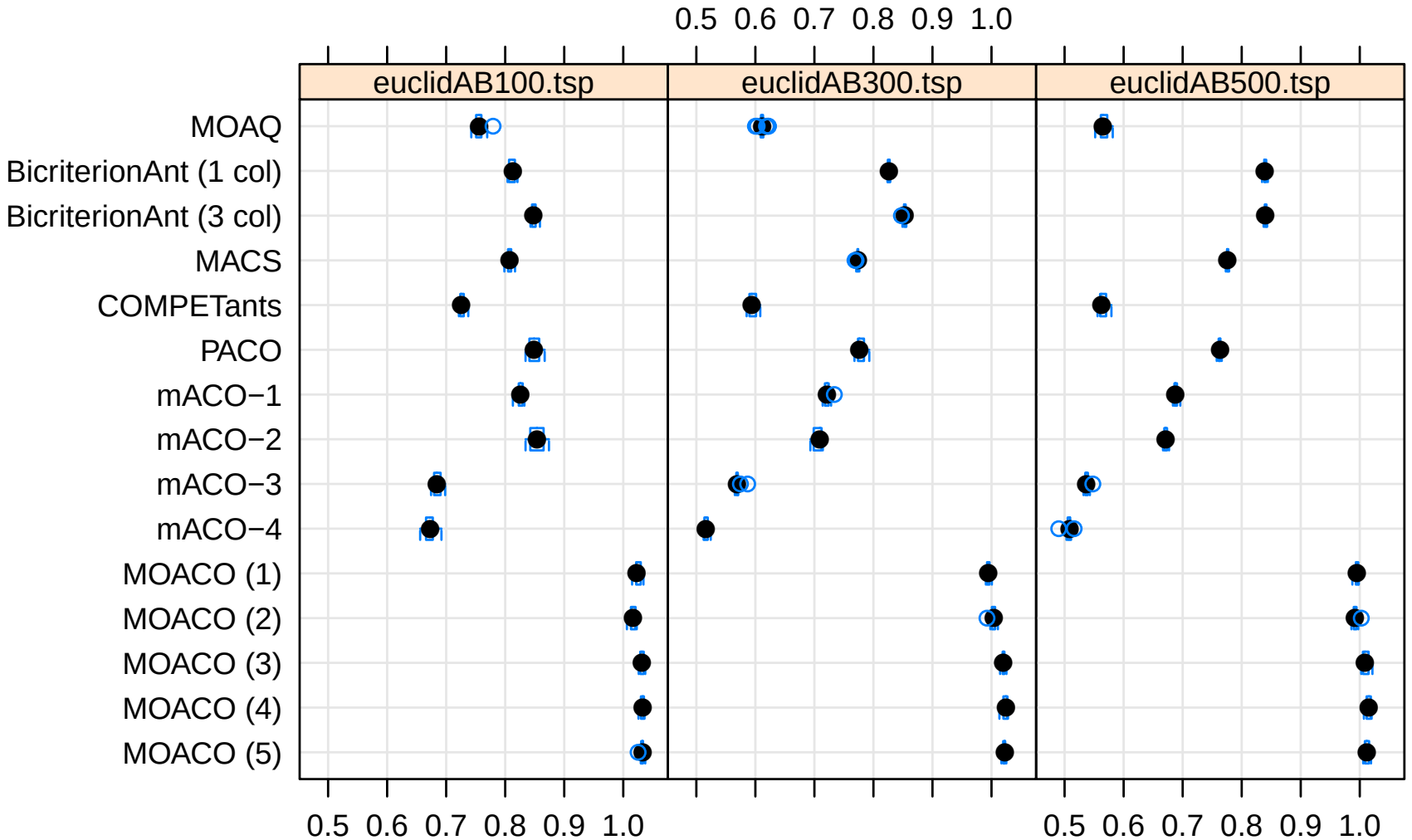
Example of a top-down approach to algorithm configuration

MOACO framework

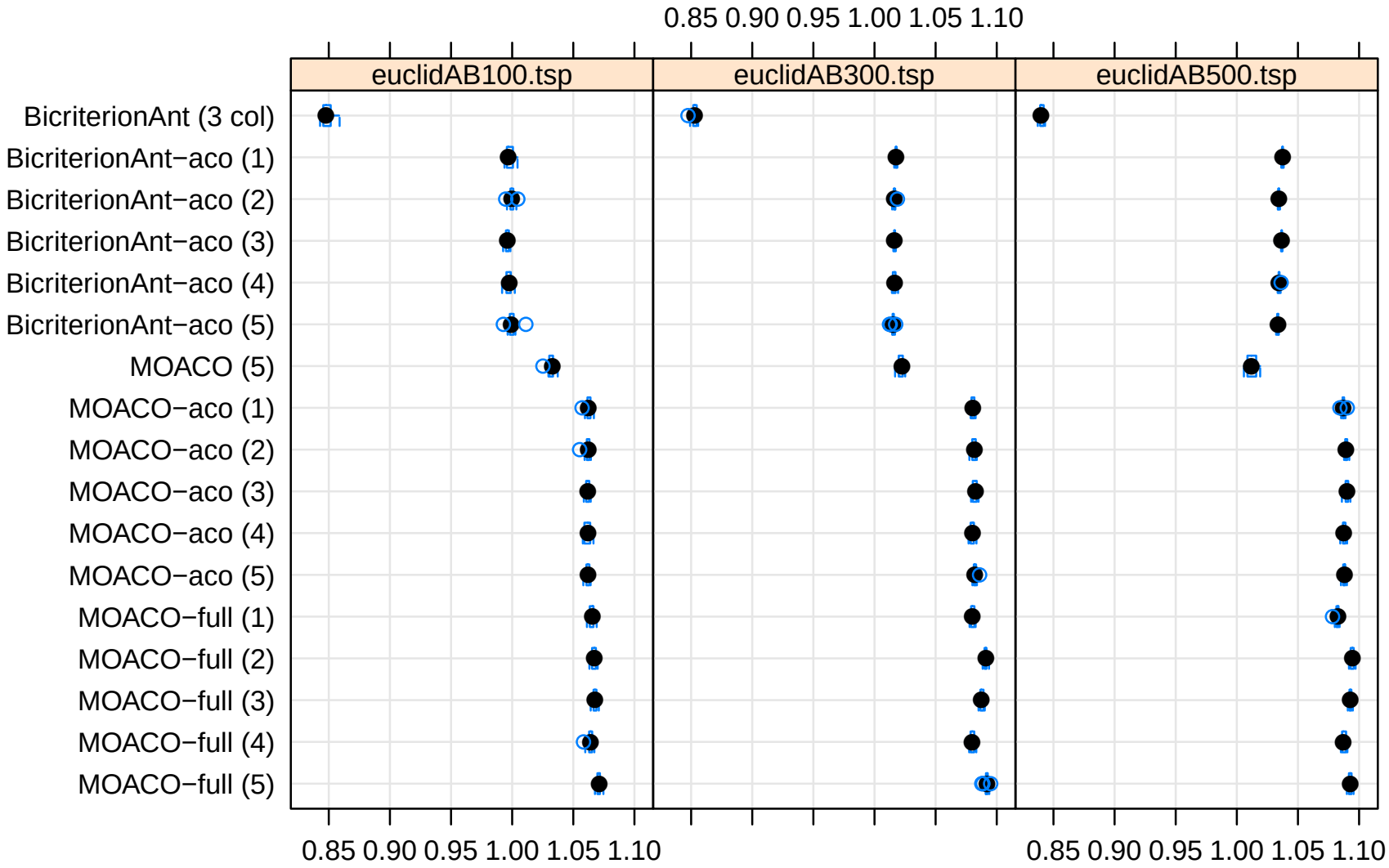


irace + hypervolume = automatic configuration of multi-objective solvers!

Automatic configuration multi-objective ACO



Automatic configuration multi-objective ACO



Summary

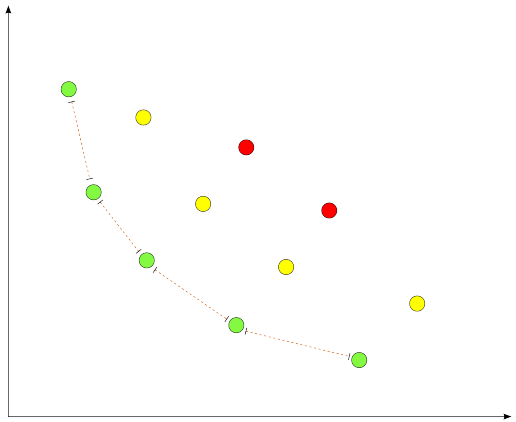
- ~~We propose a new MOACO algorithm that...~~
- We propose an approach to automatically design MOACO algorithms:
  1. Synthesize state-of-the-art knowledge into a flexible MOACO framework
  2. Explore the space of potential designs automatically using irace
- Other examples:
  - Single-objective frameworks for MIP: CPLEX, SCIP
  - Single-objective framework for SAT, SATenstein
  - Multi-objective algorithm frameworks (TP+PLS, MOEA)

Example, new applications

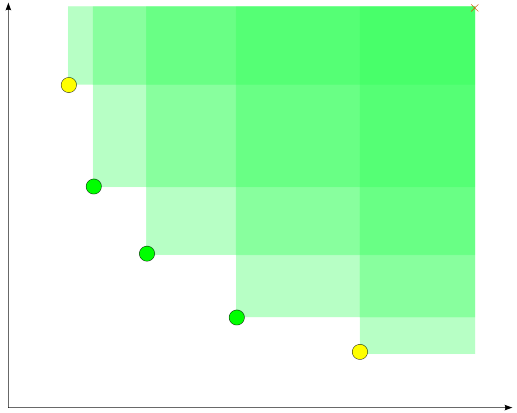
Multi-objective evolutionary algorithms (MOEA)



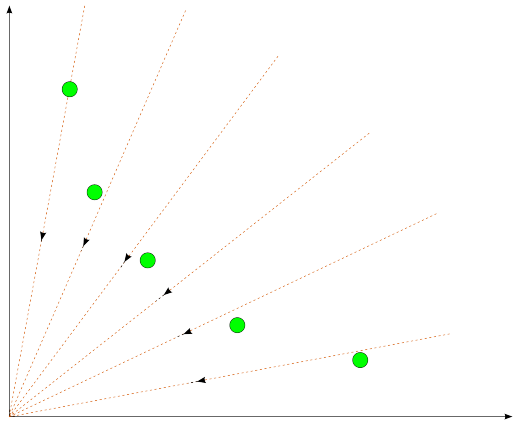
Multi-objective evolutionary algorithms



Pareto based
(NSGA-II, SPEA2)



Indicator based
(IBEA, SMS-EMOA)



Weight based
(MOGLS, MOEA/D)

We focus on building an automatically configurable component-wise framework for Pareto- and indicator-based MOEAs

MOEA Framework — outline


```
1:  pop ← Initialization ()
2:  if type(pop_ext) != none
3:      pop_ext ← pop
4:  repeat
5:      pool ← BuildMatingPool (pop)
6:      pop_new ← Variation (pool)
7:      pop_new ← Evaluation (pop_new)
8:      pop     ← Replacement (pop, pop_new)
9:      if type(pop_ext) = bounded then
10:         pop_ext ← Replacement_Ext (pop_ext, pop_new)
11:     else if type(pop_ext) = unbounded then
12:         pop_ext ← pop_ext ∪ pop
13: until termination criteria met
14: if type(pop_ext) = none
15:     return pop
16: else
17:     return pop_ext
```

Preference relations in mating / replacement

Component	Parameters
Preference	$\langle$ Set-partitioning, Quality, Diversity $\rangle$
BuildMatingPool	$\langle$ Preference$_{Mat}$, Selection $\rangle$
Replacement	$\langle$ Preference$_{Rep}$, Removal $\rangle$
Replacement$_{Ext}$	$\langle$ Preference$_{Ext}$, Removal$_{Ext}$ $\rangle$

Representing known MOEAs

	BuildMatingPool				Replacement			
Alg.	SetPart	Quality	Diversity	Selection	SetPart	Quality	Diversity	Removal
MOGA	rank	—	niche-sh.	DT	—	—	—	generational
NSGA-II	depth	—	crowding	DT	depth	—	crowding	one-shot
SPEA2	strength	—	kNN	DT	strength	—	kNN	sequential
IBEA	—	binary	—	DT	—	binary	—	one-shot
HypE	—	I_H^h	—	DT	depth	I_H^h	—	sequential
SMS	—	—	—	random	depth-rank	I_H^1	—	—

(All MOEAs above use fixed size population and no external archive;
in addition, SMS-EMOA uses $\lambda = 1$)

Experimental setup

- Benchmarks
  - DTLZ (7) and WFG (9) of 2, 3, and 5 objectives
- Scenarios
  - fixed budget, fixed computation time
- Training / Testing set
  - $D_{training} = \{20, 21, \ldots, 60\} \setminus D_{testing} = \{30, 40, 50\}$
- Configuration setup
  - all compared algorithms fine-tuned
  - tuning budget 25 000 algorithm runs

Experimental results

DTLZ			WFG		
2-obj $\Delta R = 126$	3-obj $\Delta R = 127$	5-obj $\Delta R = 107$	2-obj $\Delta R = 169$	3-obj $\Delta R = 130$	5-obj $\Delta R = 97$
Auto_{D2} (1339)	**Auto_{D3} (1500)**	**Auto_{D5} (1002)**	**Auto_{W2} (1692)**	**Auto_{W3} (1375)**	**Auto_{W5} (1170)**
SPEA2_{D2} (1562)	IBEA_{D3} (1719)	SMS_{D5} (1550)	SPEA2_{W2} (2097)	SMS_{W3} (1796)	SMS_{W5} (1567)
IBEA_{D2} (1940)	SMS_{D3} (1918)	IBEA_{D5} (1867)	$\text{NSGA-II}_{\text{W2}}$ (2542)	IBEA_{W3} (1843)	IBEA_{W5} (1746)
$\text{NSGA-II}_{\text{D2}}$ (2143)	HypE_{D3} (2019)	SPEA2_{D5} (2345)	SMS_{W2} (2621)	SPEA2_{W3} (2600)	SPEA2_{W5} (2747)
HypE_{D2} (2338)	SPEA2_{D3} (2164)	$\text{NSGA-II}_{\text{D5}}$ (2346)	IBEA_{W2} (2777)	$\text{NSGA-II}_{\text{W3}}$ (3315)	$\text{NSGA-II}_{\text{W5}}$ (3029)
SMS_{D2} (2406)	$\text{NSGA-II}_{\text{D3}}$ (2528)	HypE_{D5} (2674)	HypE_{W2} (2851)	HypE_{W3} (3431)	MOGA_{W5} (4268)
MOGA_{D2} (2970)	MOGA_{D3} (2851)	MOGA_{D5} (2915)	MOGA_{W2} (4320)	MOGA_{W3} (4540)	HypE_{W5} (4373)

Additional remarks

- additional results
  - time-constrained scenarios
  - cross-benchmark comparison
  - applications to multi-objective flow-shop scheduling
- extensions
  - more comprehensive benchmarks sets
  - design space analysis (e.g. abalation)
  - extensions of template (weights, local search, etc.)

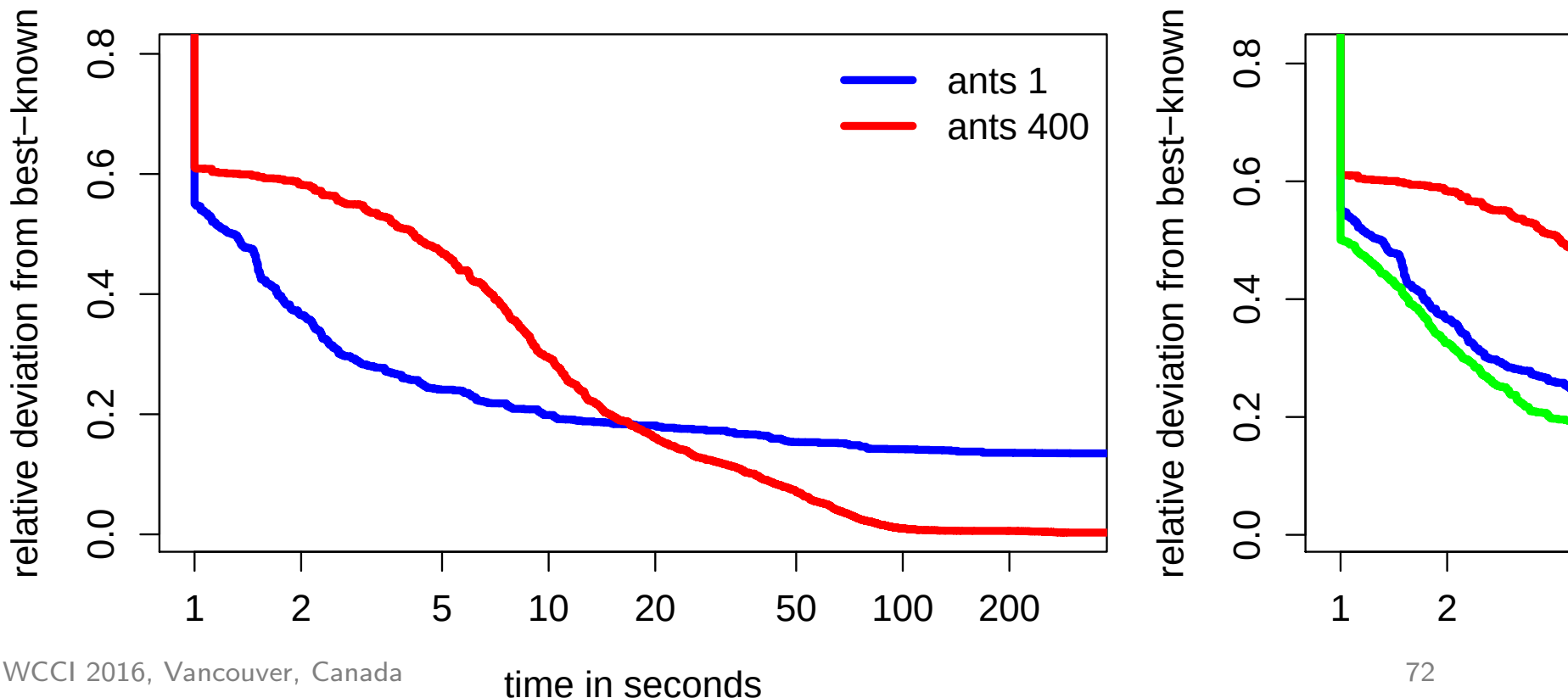
Time has come to automatically configure MOEAs (and other algorithms)

Example, new applications

Improving automatically the anytime behavior of algorithms


"Anytime" Algorithms [Zilberstein, 1996]

"Anytime" algorithms aim to produce as high quality results as possible, independent of the computation time allowed.



Brute-Force Approach

1. Choose *many* parameter settings
2. Run lots of experiments
3. Visually compare SQT plots

After about one year:

+ Strategies for varying *ants*, β, or q_0 that significantly improve the anytime behaviour of MMAS on the TSP.
- Extremely time consuming
- Subjective / Bias

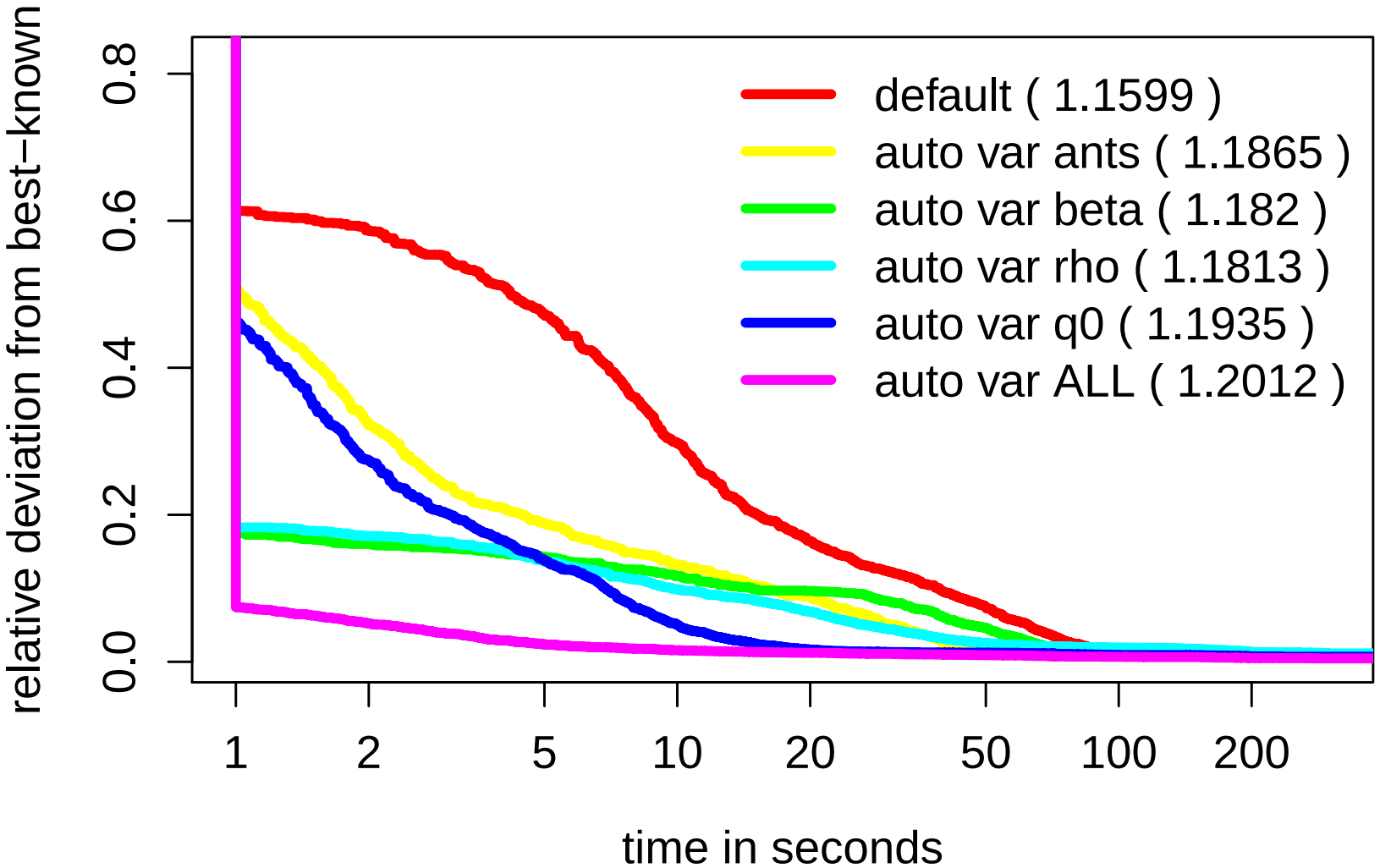
New approach

López–Ibáñez, Stützle, 2011

- multi-objective optimization
  - \+ Objectively defined comparison
  - \+ Performance assessment techniques (hypervolume)

- Automatic configuration
  - \+ Most effort done by the computer
  - \+ Best configurations selected by the computer: *Unbiased*

Experimental comparison



Conclusions on configuring anytime algorithms

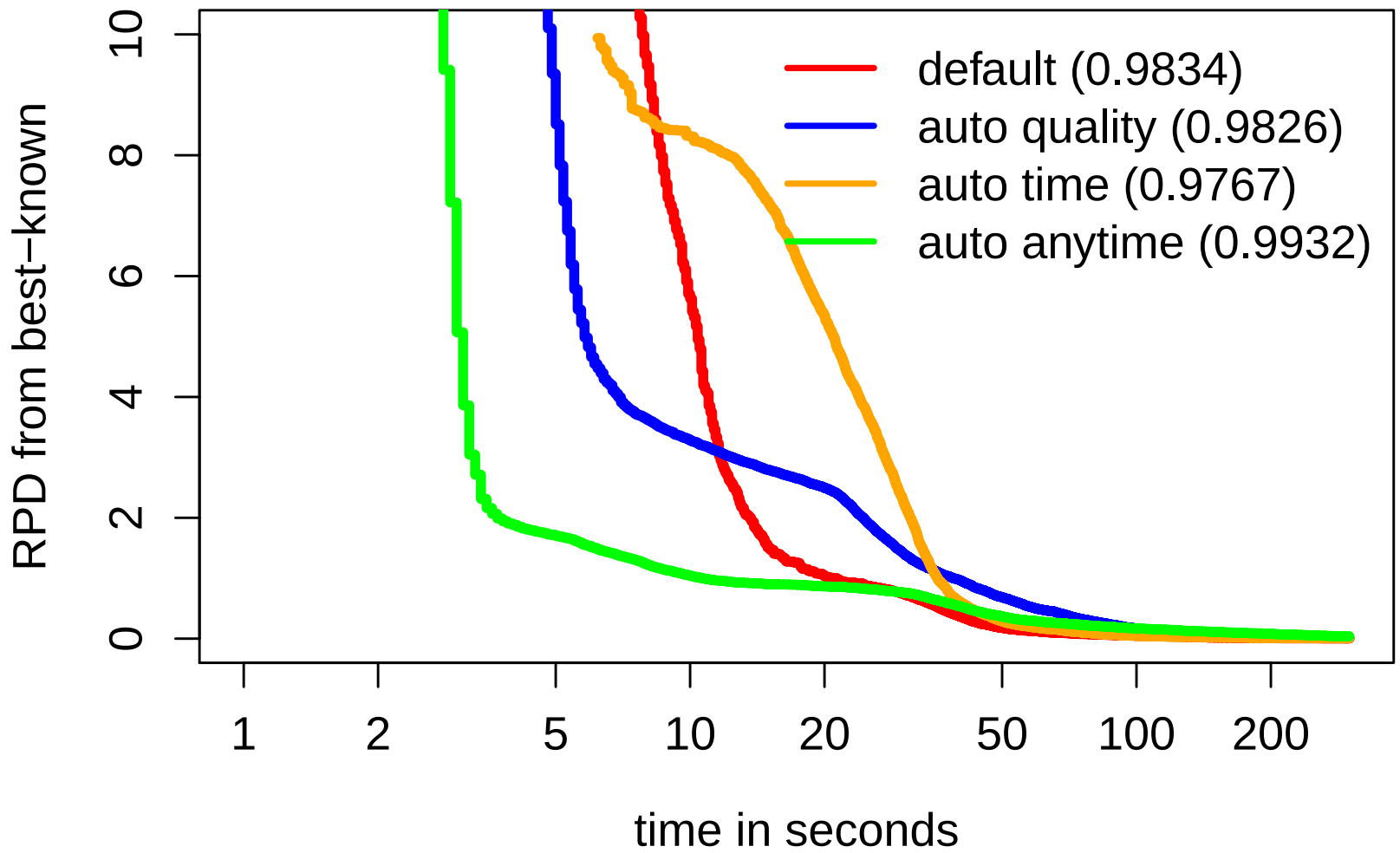
- Less effort: 1 week instead of a year!
- Same or even better results
- Improving the anytime behaviour of metaheuristics becomes *much easier*

We can use offline configuration of online strategies for improving anytime behaviour

1. Implement several online strategies
2. Let offline automatic configuration choose the best strategy for our algorithm / problem

Further work: Improving anytime behavior for SCIP solver v.2.1.0 configuring more than 200 parameters as proof of concept.

Improving anytime behavior of SCIP



Applying SCIP to Winner determination problem for combinatorial auctions; 1000 training, 1000 test instances, 300 secs CPU time; 5000 budget

Few other topics

Scaling to expensive instances

What if my problem instances are too difficult/large?

- Cloud computing / Large computing clusters
- J. Styles and H. H. Hoos. **Automatically Configuring Algorithms for Scaling Performance.** LION, 2012; (extensions also at GECCO 2013)

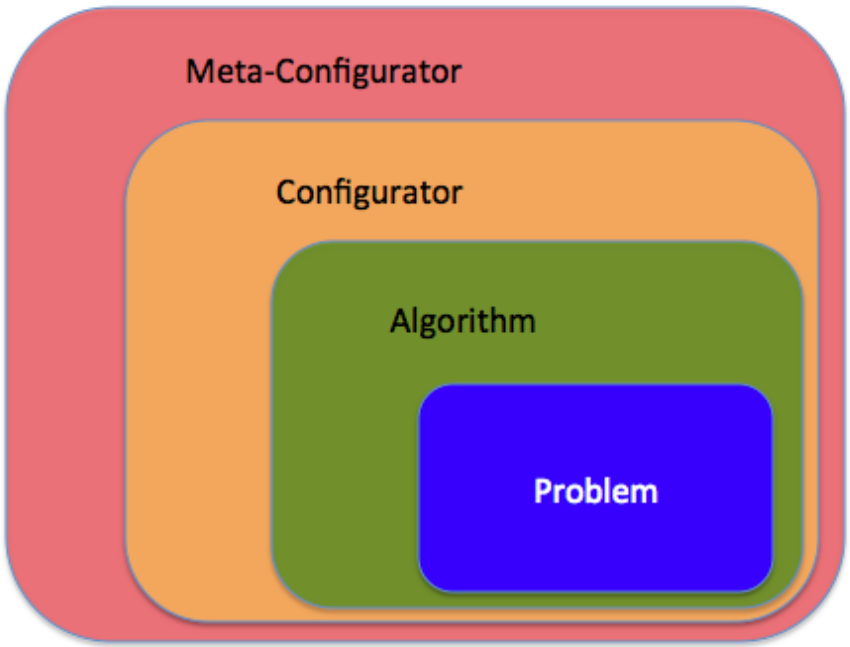
  Tune on small instances,
  then extend to increasingly larger ones

- F. Mascia, M. Birattari, and T. Stützle. **Tuning algorithms for tackling large instances: An experimental protocol.** Learning and Intelligent Optimization, LION 7, 2013.

  Tune on small /medium-size instances,
  then scale parameter values to difficult ones

Configuring configurators

What about configuring automatically the configurator?
. . . and configuring the configurator of the configurator?



✔ it can be done (Hutter et al., 2009) but . . .

✘ it is costly and iterating further leads to diminishing returns

AClib: A Benchmark Library for Algorithm Configuration

F. Hutter, M. Lpez-Ibez, C. Fawcett, M. Lindauer, H. H. Hoos, K. Leyton-Brown and T. Stützle. **AClib: a Benchmark Library for Algorithm Configuration**, Learning and Intelligent Optimization Conference (LION 8), 2014.

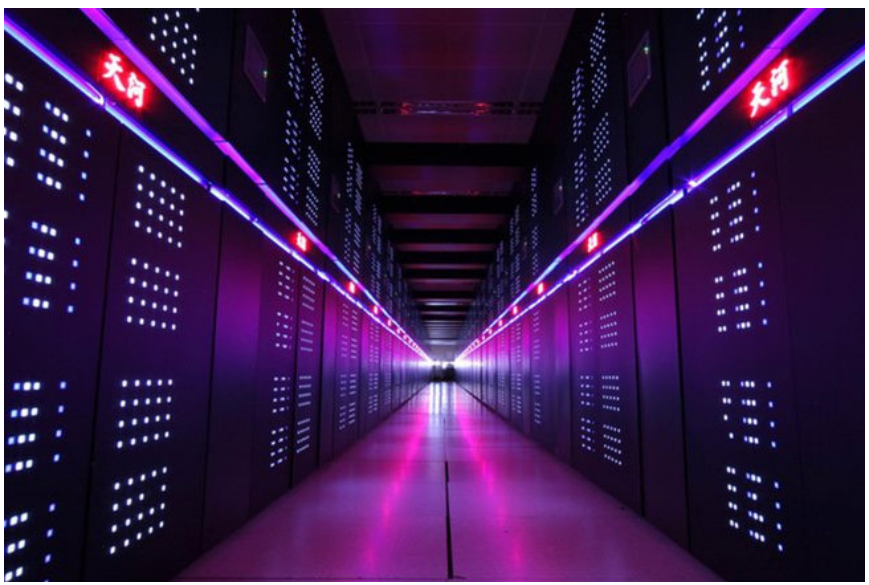
http://www.aclib.net/

- Standard benchmark for experimenting with configurators
- 182 heterogeneous scenarios
- SAT, MIP, ASP, time-tabling, TSP, multi-objective, machine learning
- Extensible $\Rightarrow$ new scenarios welcome !

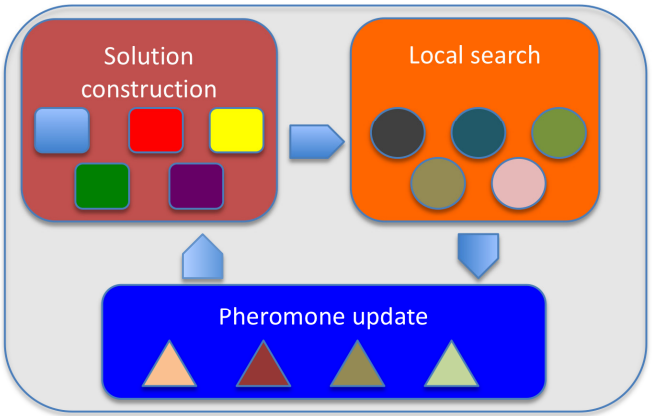
Concluding remarks

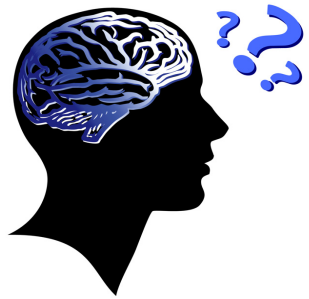
Why automatic algorithm configuration?

- improvement over manual, ad-hoc methods for tuning
- reduction of development time and human intervention
- increase number of considerable degrees of freedom
- empirical studies, comparisons of algorithms
- support for end users of algorithms

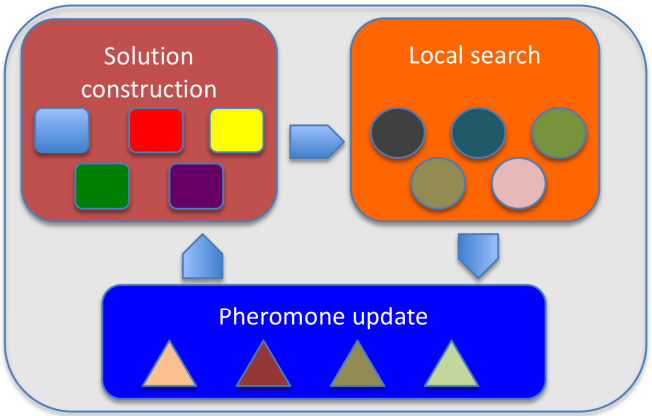


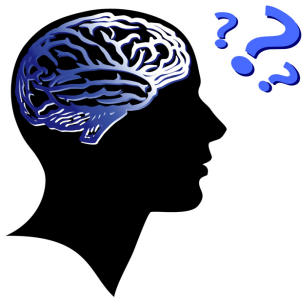
Towards a shift of paradigm in algorithm design

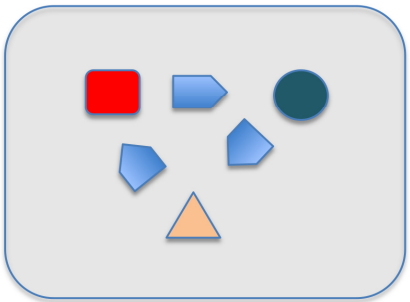




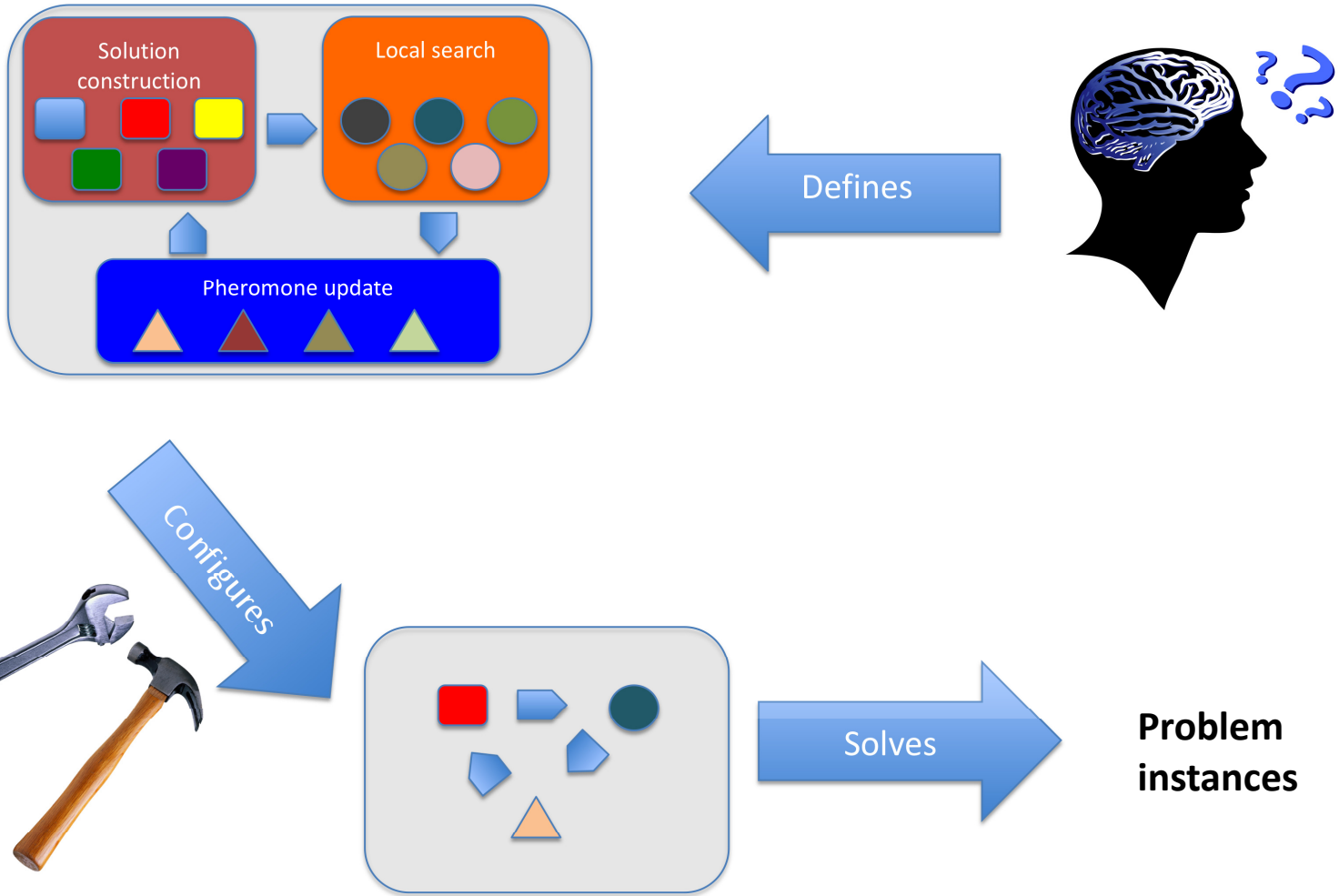
Towards a shift of paradigm in algorithm design







Towards a shift of paradigm in algorithm design



Conclusions

Automatic Configuration

- leverages computing power for software design
- is rewarding w.r.t. development time and algorithm performance

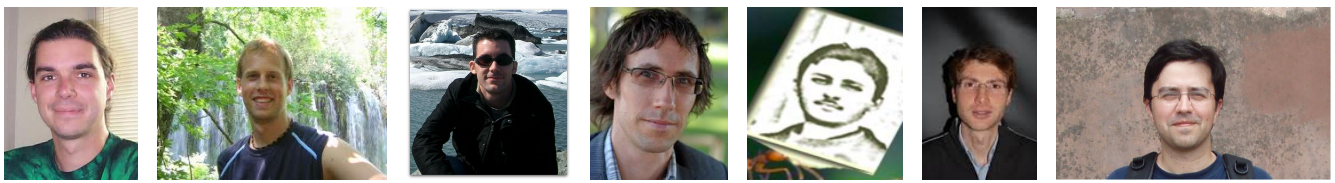
Future work

- more powerful configurators
- more and more complex applications
- paradigm shift in optimization software development

Acknowledgements

IRIDIA

External collaborators



Research funding

F.R.S.-FRNS, Projects ANTS (ARC), Meta-X (ARC), Comex (PAI), MIBISOC (FP7), COLOMBO (FP7), FRFC, Metaheuristics Network (FP5)