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. Automatic design of algorithms from frameworks
5. Concluding remarks

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, ...
  - powerful general-purpose software available
  - guarantees on optimality but often time/memory consuming
- Approximate algorithms
  - heuristics, local search, metaheuristics, hyperheuristics ...
  - typically special-purpose software
  - rarely provable guarantees but often fast and accurate

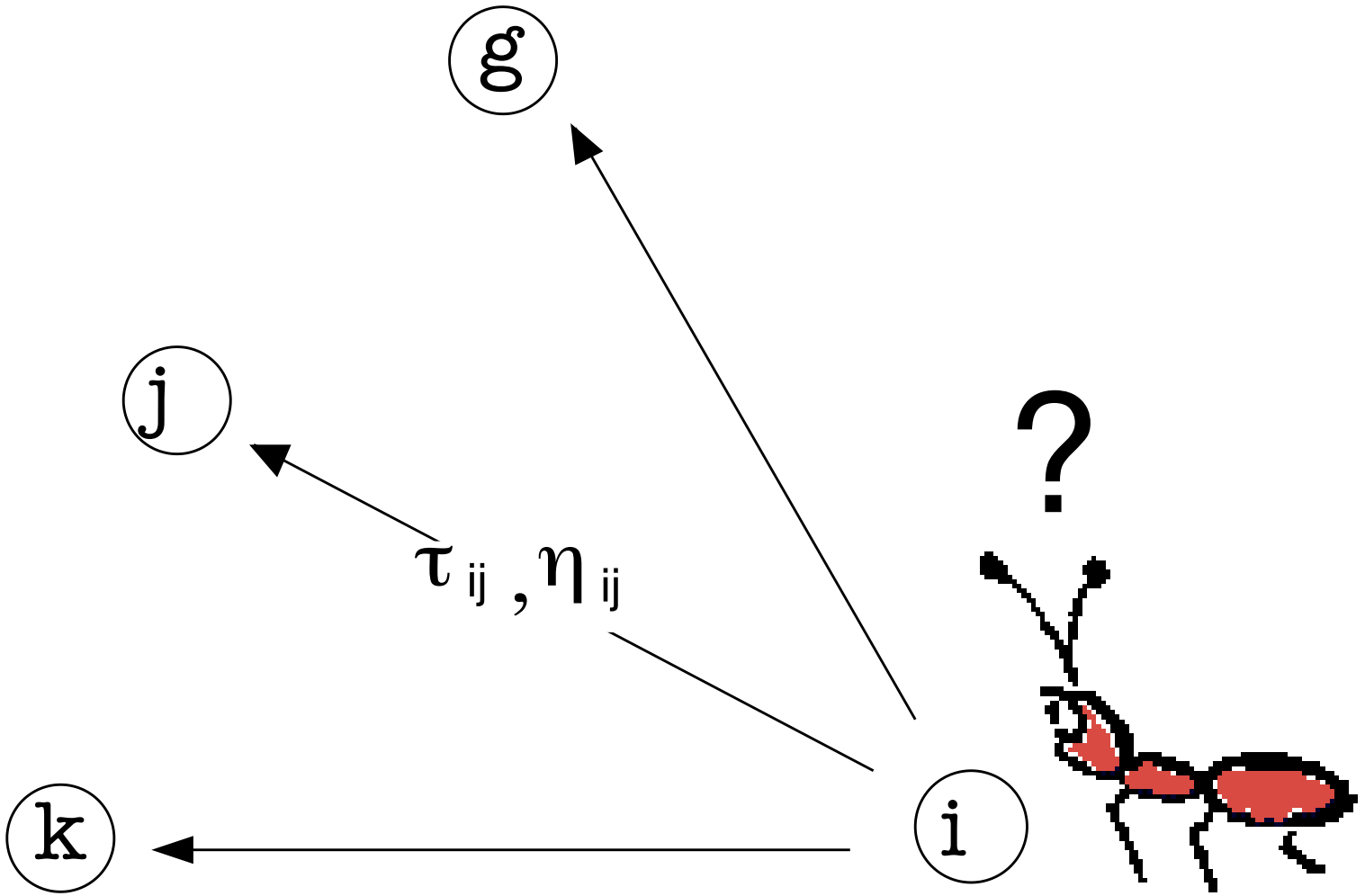
Much active research on hybrids between
exact and approximate algorithms!

Design choices and parameters everywhere

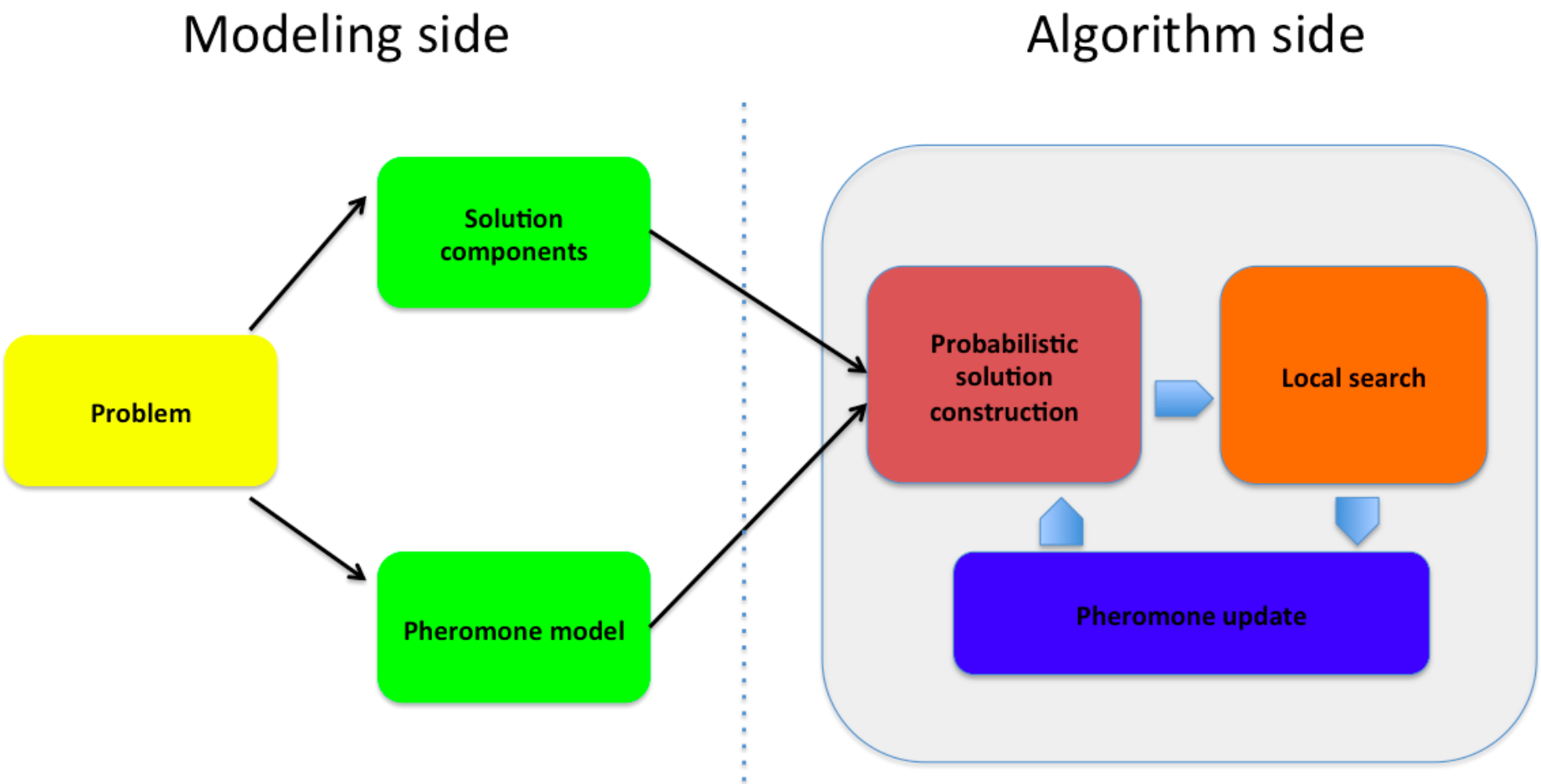
Todays high-performance optimizers involve a large number of design choices and parameter settings

- 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: hybrid SLS methods with 200+ parameters (plus several still hidden ones)

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

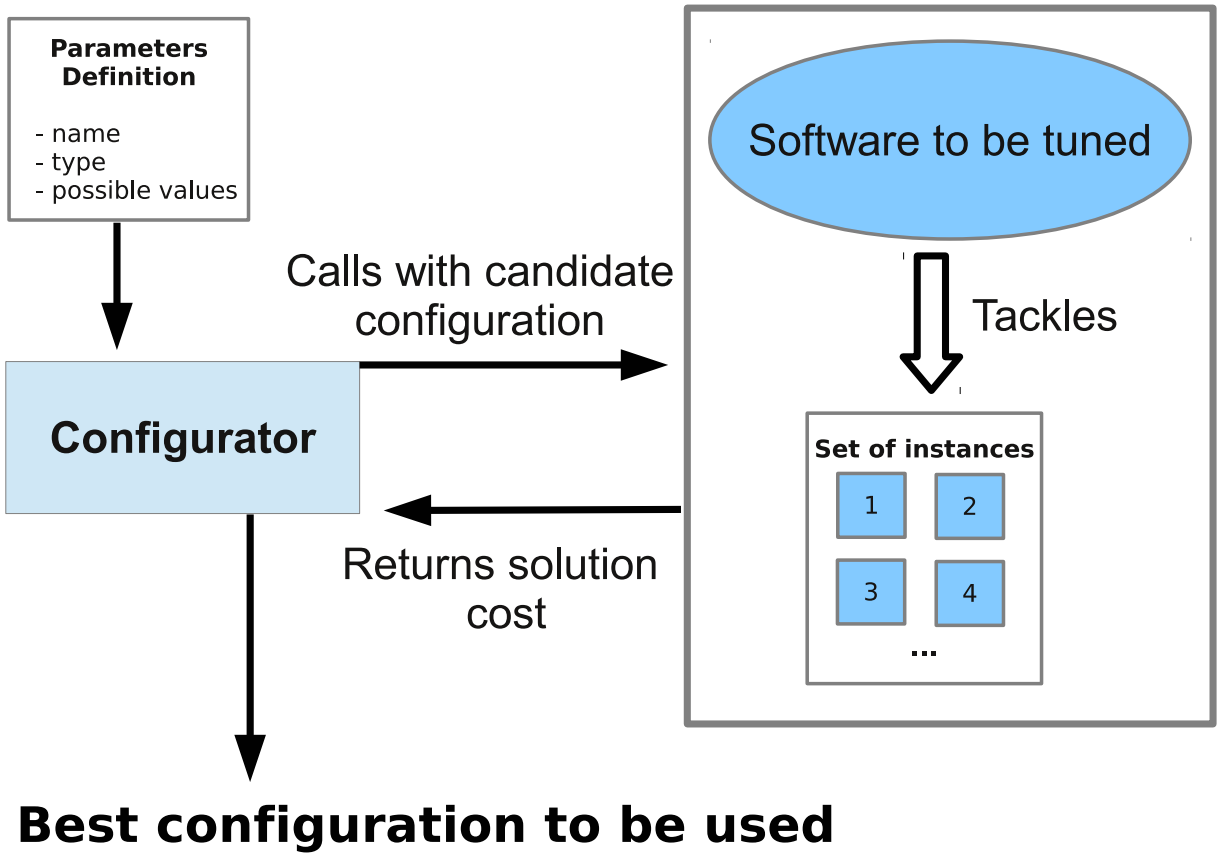
Design and configuration of algorithms involves setting categorical, ordinal, **and** *numerical parameters*

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

Automatic offline configuration



Typical performance measures

- maximize solution quality (within given computation time)
- minimize computation time (to reach optimal solution)

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

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, ...]

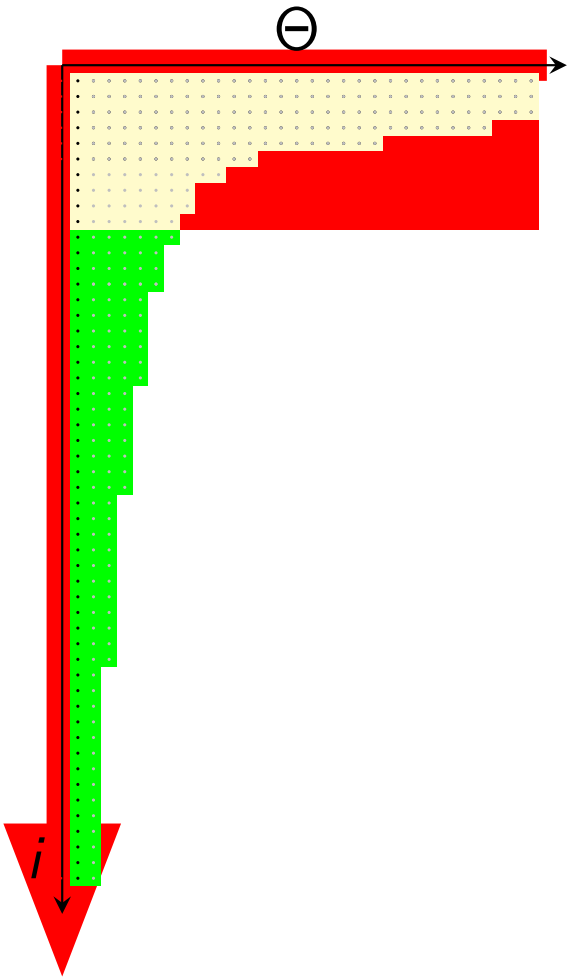
General, domain-independent methods required: (i) applicable to all variable types, (ii) multiple training instances, (iii) high performance, (iv) scalable

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

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()

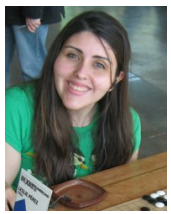
apply race

modify sampling distribution

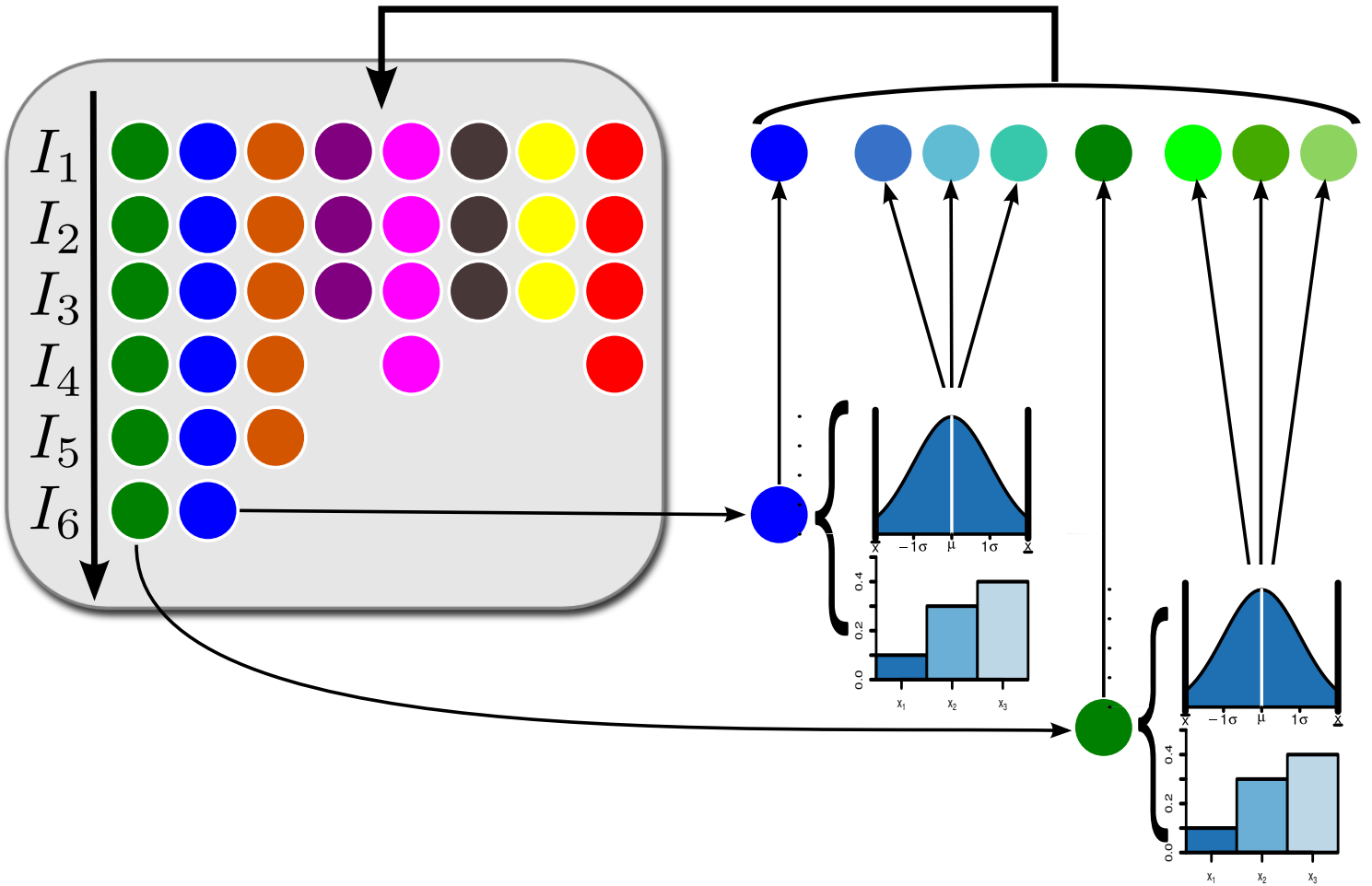
sample configurations






Iterated race: sampling



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*

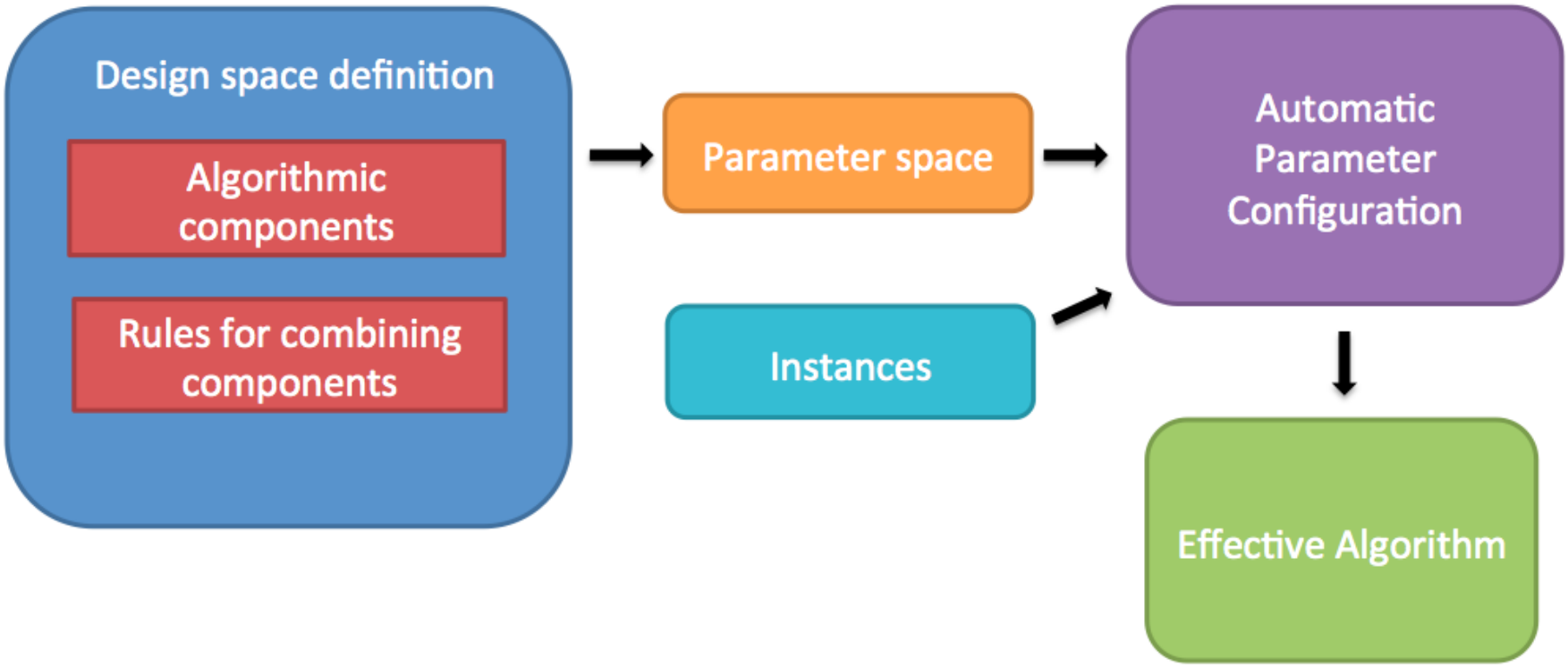
http://iridia.ulb.ac.be/irace

- 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

Automatic design of algorithms from algorithm frameworks

General approach



Main approaches

Top-down approaches

- develop flexible framework following a fixed algorithm template with alternatives
- apply high-performing configurators
- Examples: Satenstein, MOACO, MOEA, MIP Solvers?(!)

Bottom-up approaches

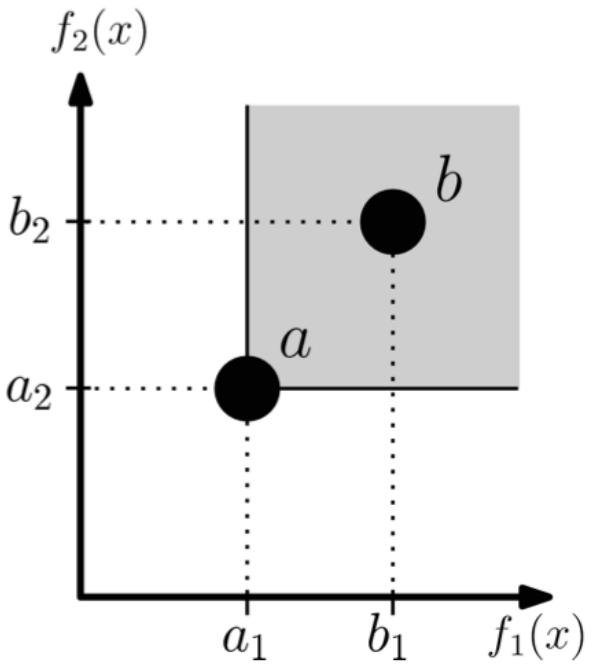
- flexible framework implementing algorithm components
- define rules for composing algorithms from components e.g. through grammars
- frequently usage of genetic programming, grammatical evolution etc.

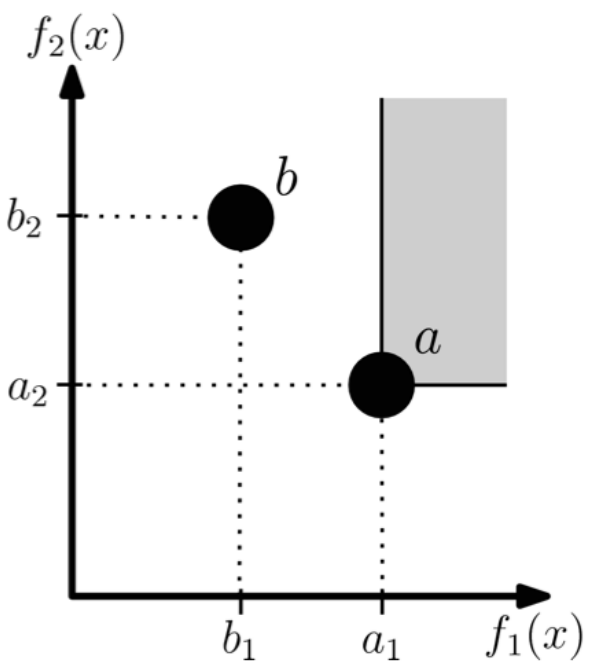
Example, design configurable algorithm framework

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

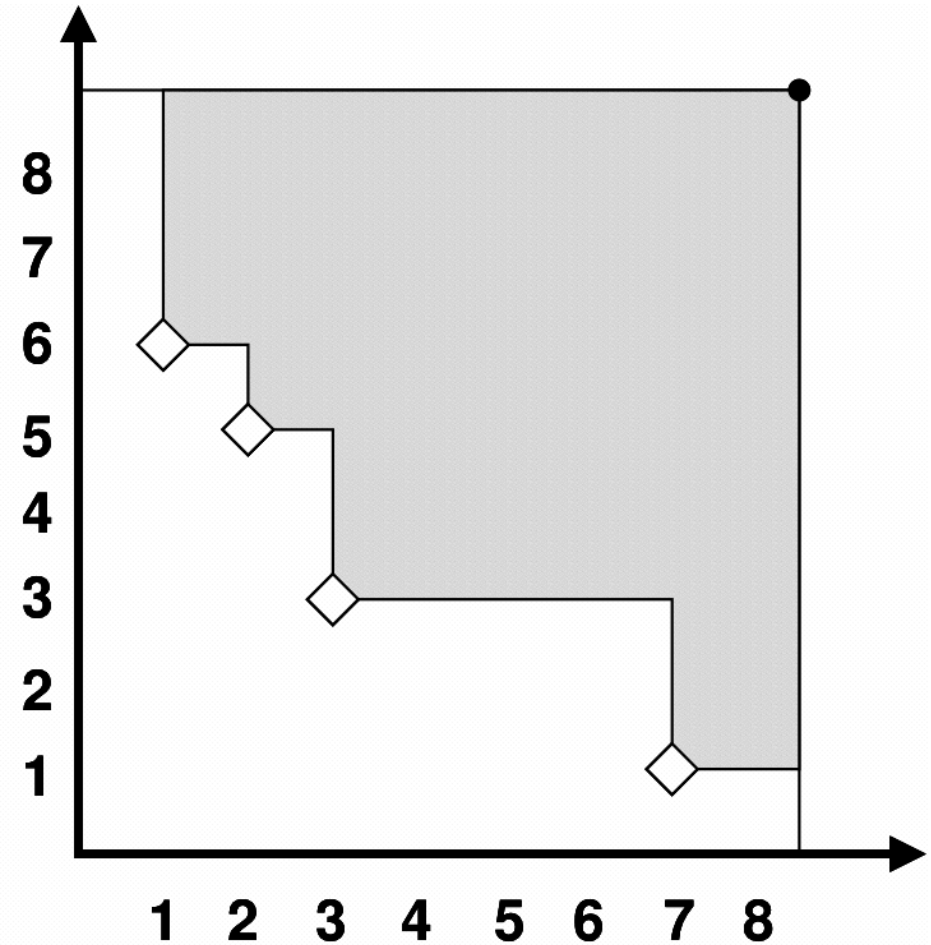
Example of a top-down approach to algorithm configuration

MOACO framework: main components

López-Ibáñez, Stützle, 2012

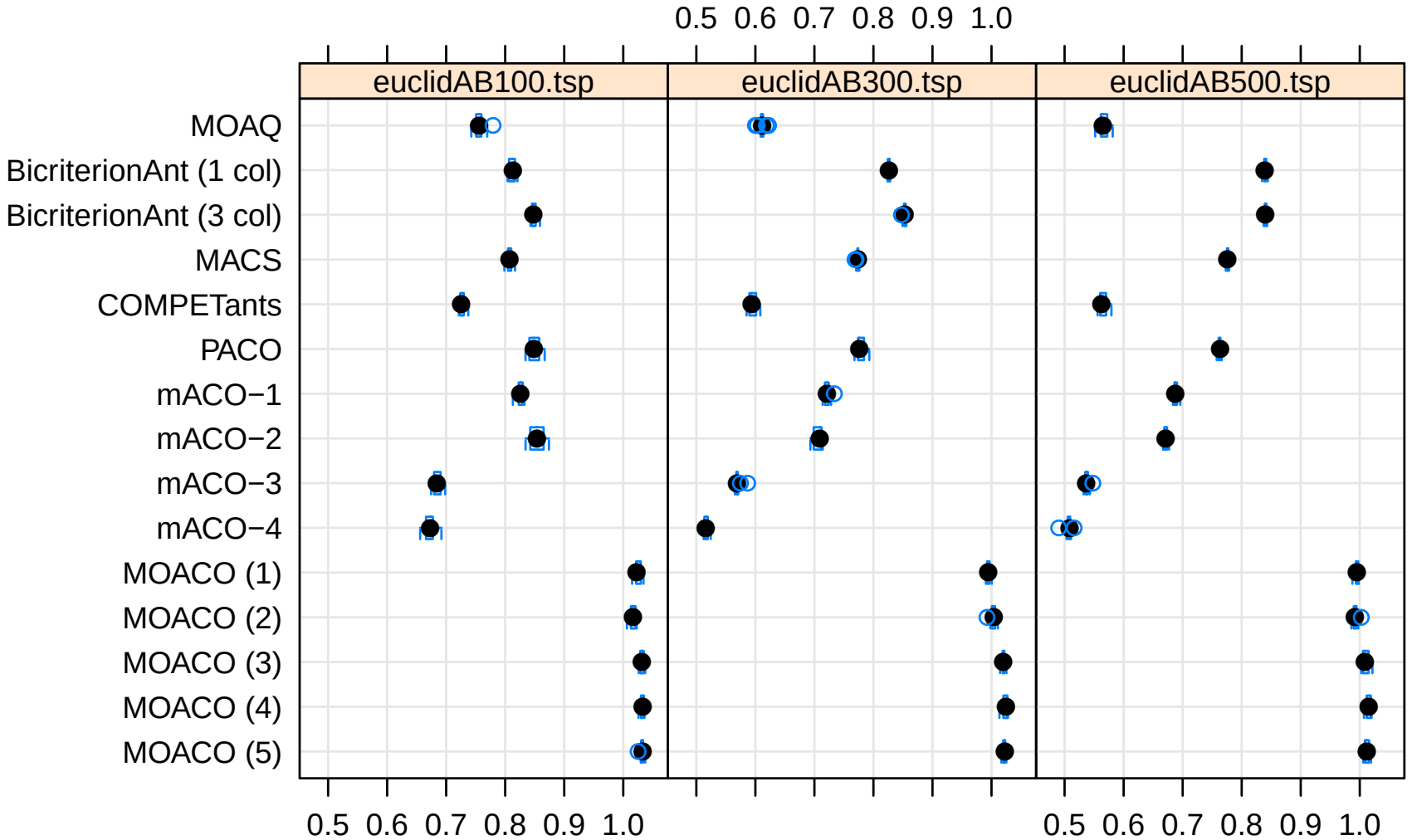
- Multi-objective part
  - number of pheromone / heuristic matrices
  - aggregation of pheromone / heuristic matrices
  - number of weights
  - weight strategy (which weight use next?)
  - pheromone update (what is rewarded?)
  - single vs. multiple colonies
- Single-objective part
  - Underlying ACO algorithms

MOACO framework

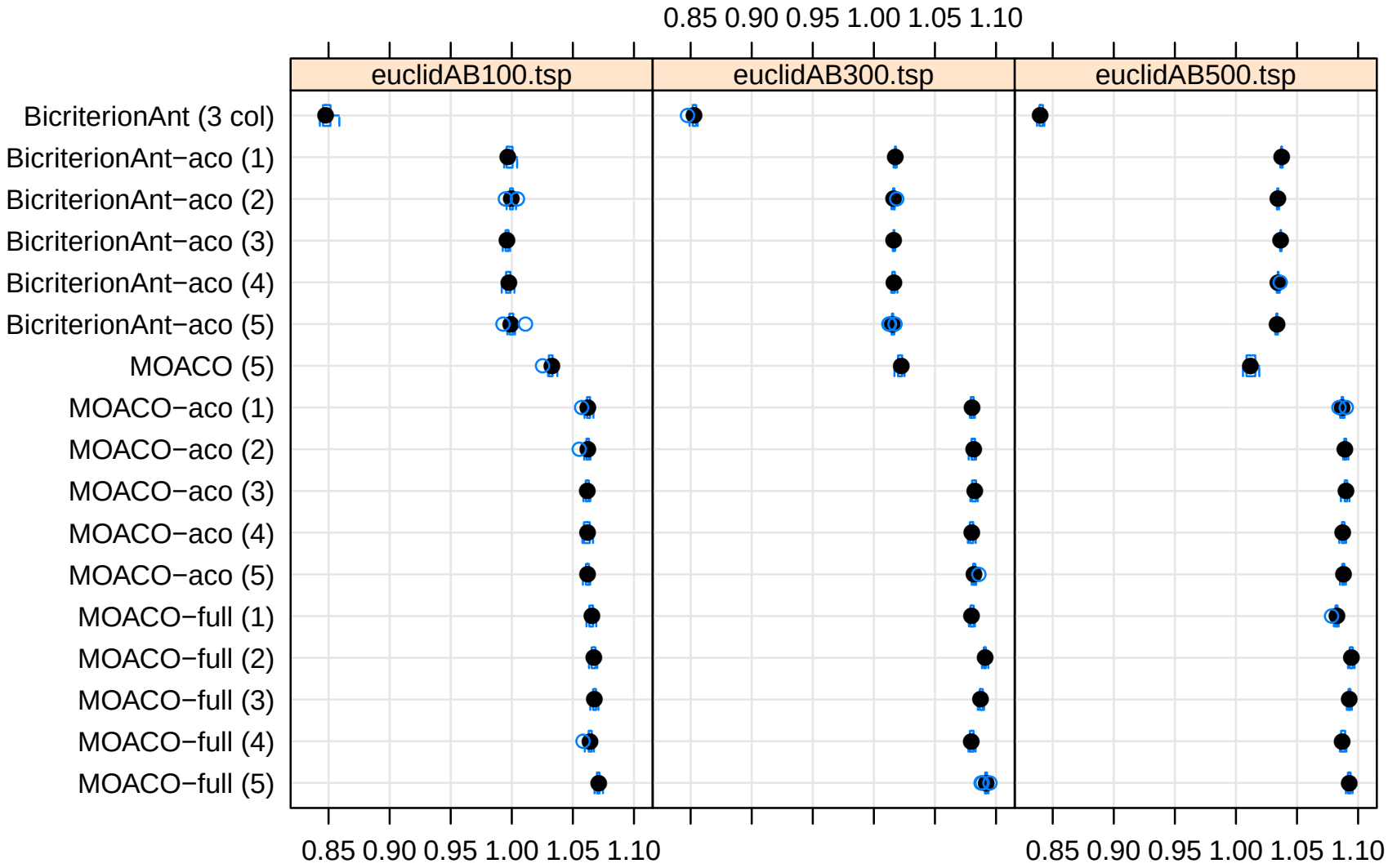


irace + hypervolume = automatic configuration of multi-objective solvers!

Automatic configuration multi-objective ACO



Automatic configuration multi-objective ACO



Summary

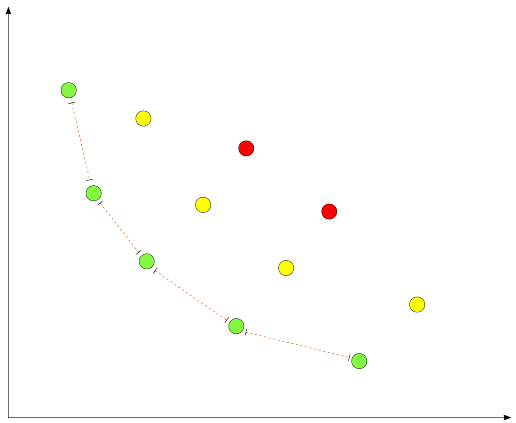
- ~~We propose a new MOACO algorithm that...~~
- We propose an approach to automatically design MOACO algorithms:
  1. Synthesize state-of-the-art knowledge into a flexible MOACO framework
  2. Explore the space of potential designs automatically using irace
- Other examples:
  - Single-objective frameworks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

Example, new applications

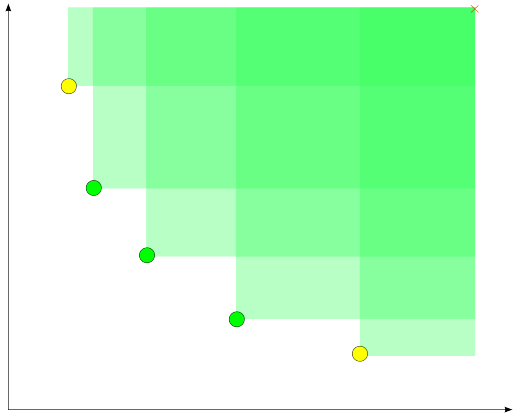
Multi-objective evolutionary algorithms (MOEA)



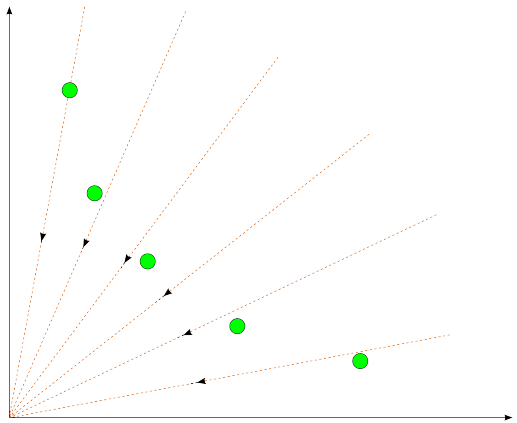
Multi-objective evolutionary algorithms



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

[Bezerra, Lópes-Ibáñez, Stützle, 2016]


```
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)

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
- extended version of AutoMOEA
  - extensions of template (weights, local search, etc.)
  - more comprehensive benchmarks sets
  - in-depth comparison of MOEAs
  - design space analysis (e.g. ablation)

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

Example, bottom-up generation of hybrid SLS 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 through 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}
```

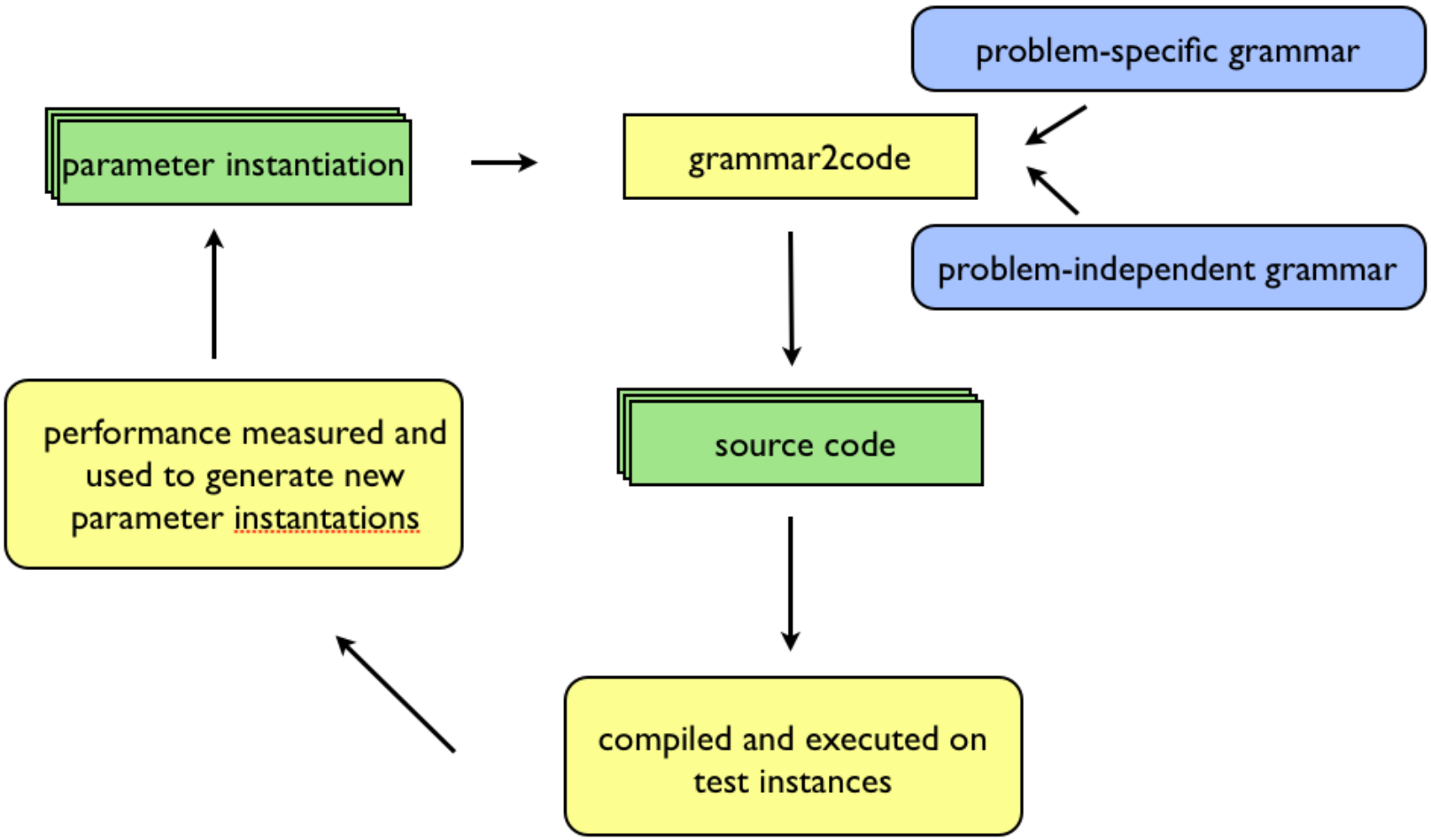
Grammar

```
     <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

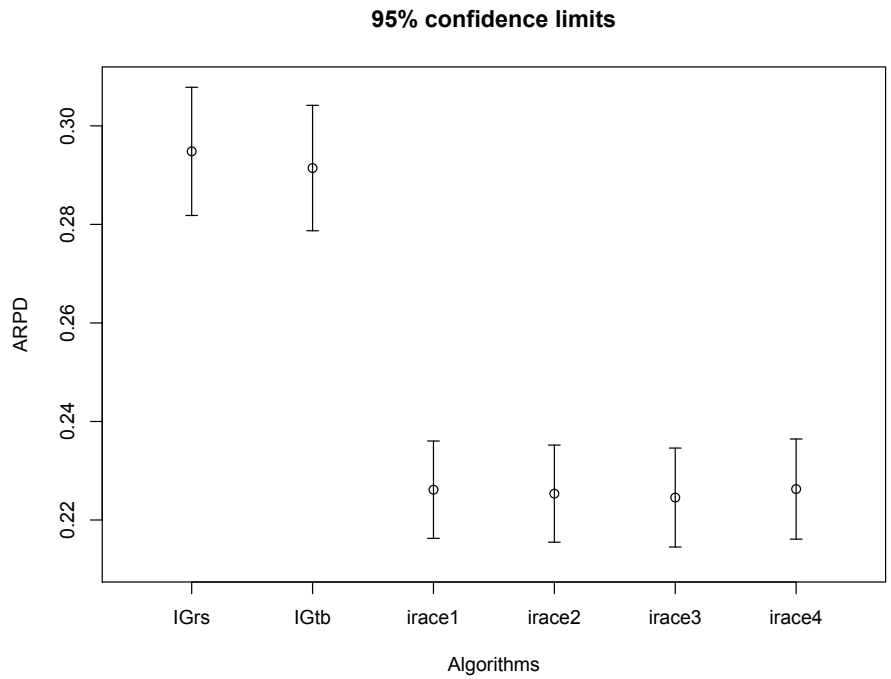
```
     <algorithm> ::= <initialization> <ils>
<initialization> ::= random | <pbs_initialization>
          <ils> ::= ILS(<perturb>, <ls>, <accept>, <stop>)
<perturb> ::= none | <initialization> | <pbs_perturb>
     <ls> ::= <ils> | <descent> | <sa> | <rii> | <pii> | <vns> | <ig> | <pbs_ls
 <accept> ::= 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 makespan objective

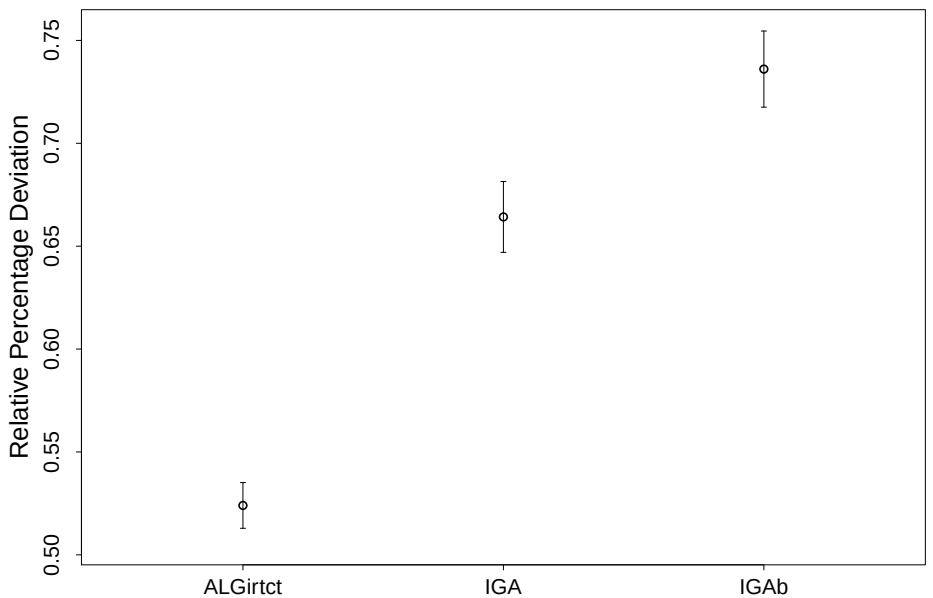
- Automatic configuration:
  - max. three levels of recursion
  - biased / unbiased grammar resulting in 262 and 502 parameters, respectively
  - budget: 200 000 trials of $n \cdot m \cdot 0.03$ seconds



Results are clearly superior to state-of-the-art

Flow-shop problem with total completion time objective

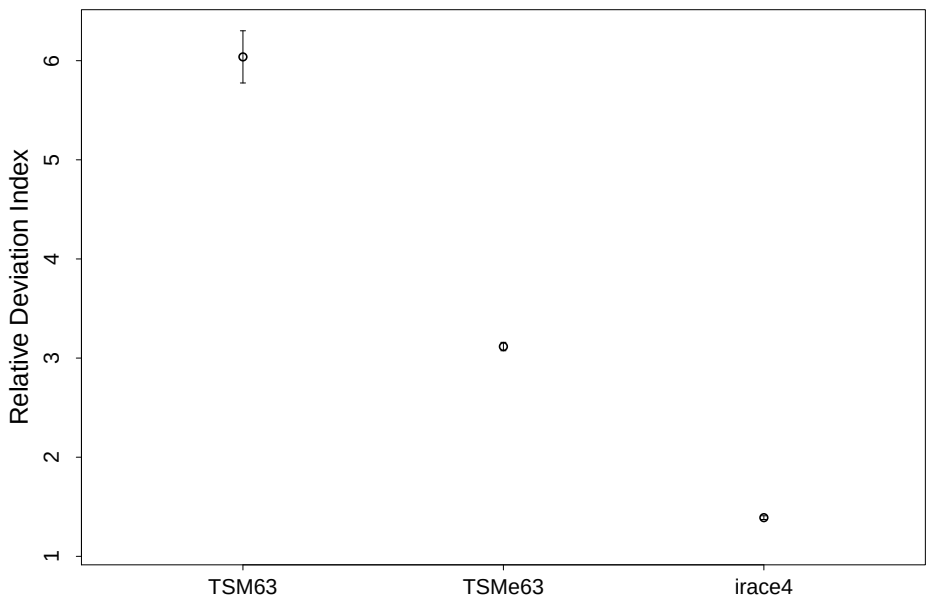
- Automatic configuration:
  - max. three levels of recursion
  - biased / unbiased grammar resulting in 262 and 502 parameters, respectively
  - budget: 100 000 trials of $n \cdot m \cdot 0.03$ seconds



Results are clearly superior to state-of-the-art

Flow-shop problem with total tardiness objective

- Automatic configuration:
  - max. three levels of recursion
  - biased / unbiased grammar resulting in 262 and 502 parameters, respectively
  - budget: 100 000 trials of $n \cdot m \cdot 0.03$ seconds



Results are clearly superior to state-of-the-art

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

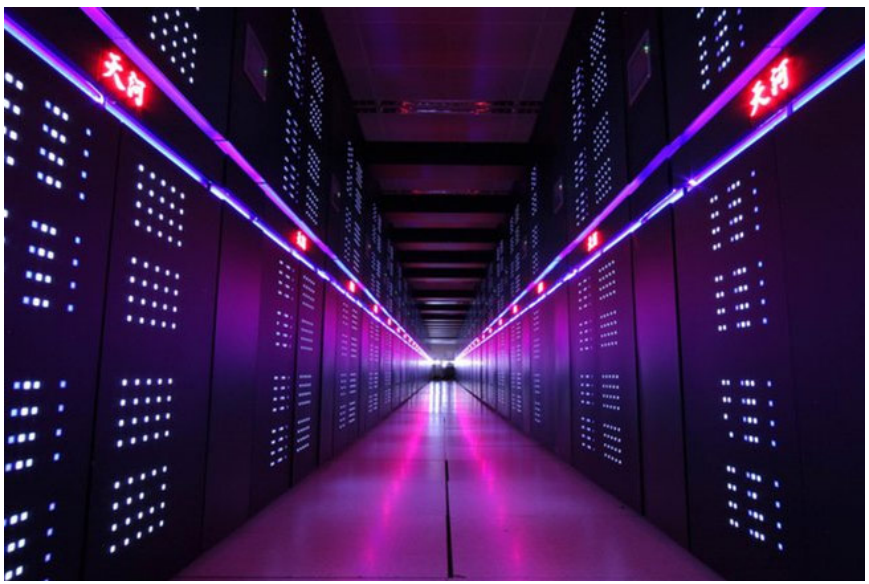
Current/future work

- extensions to other methods and templates
- dealing with complexity of hybrid algorithms
- increase generality, tackling wide problem classes

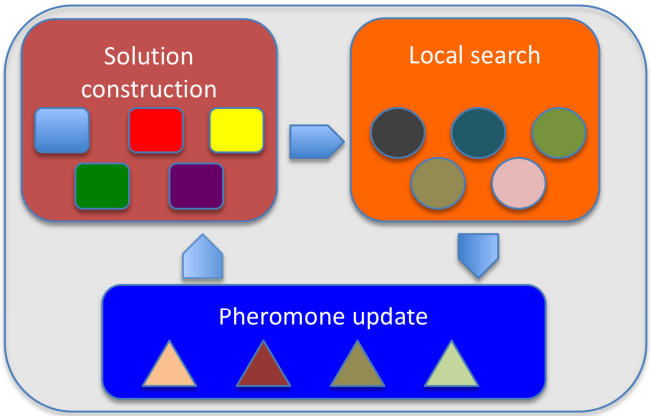
Summary and Perspectives

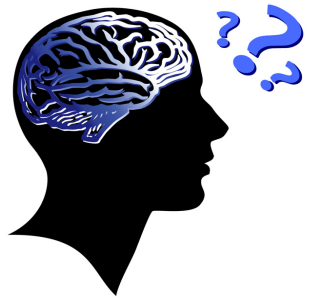
Why automatic algorithm configuration?

- 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

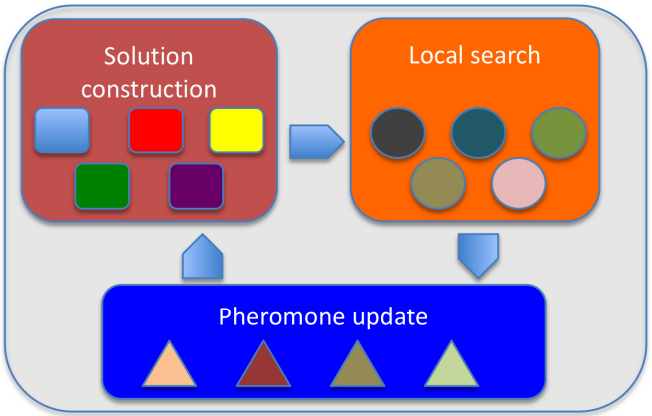


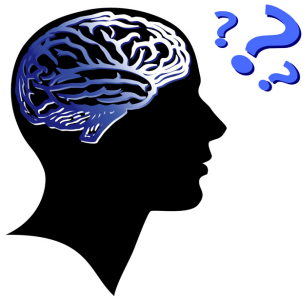
Towards a shift of paradigm in algorithm design

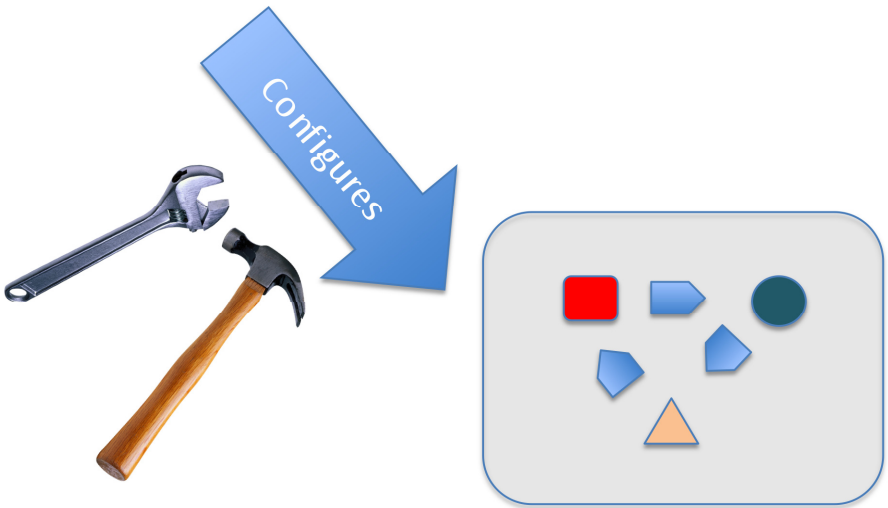




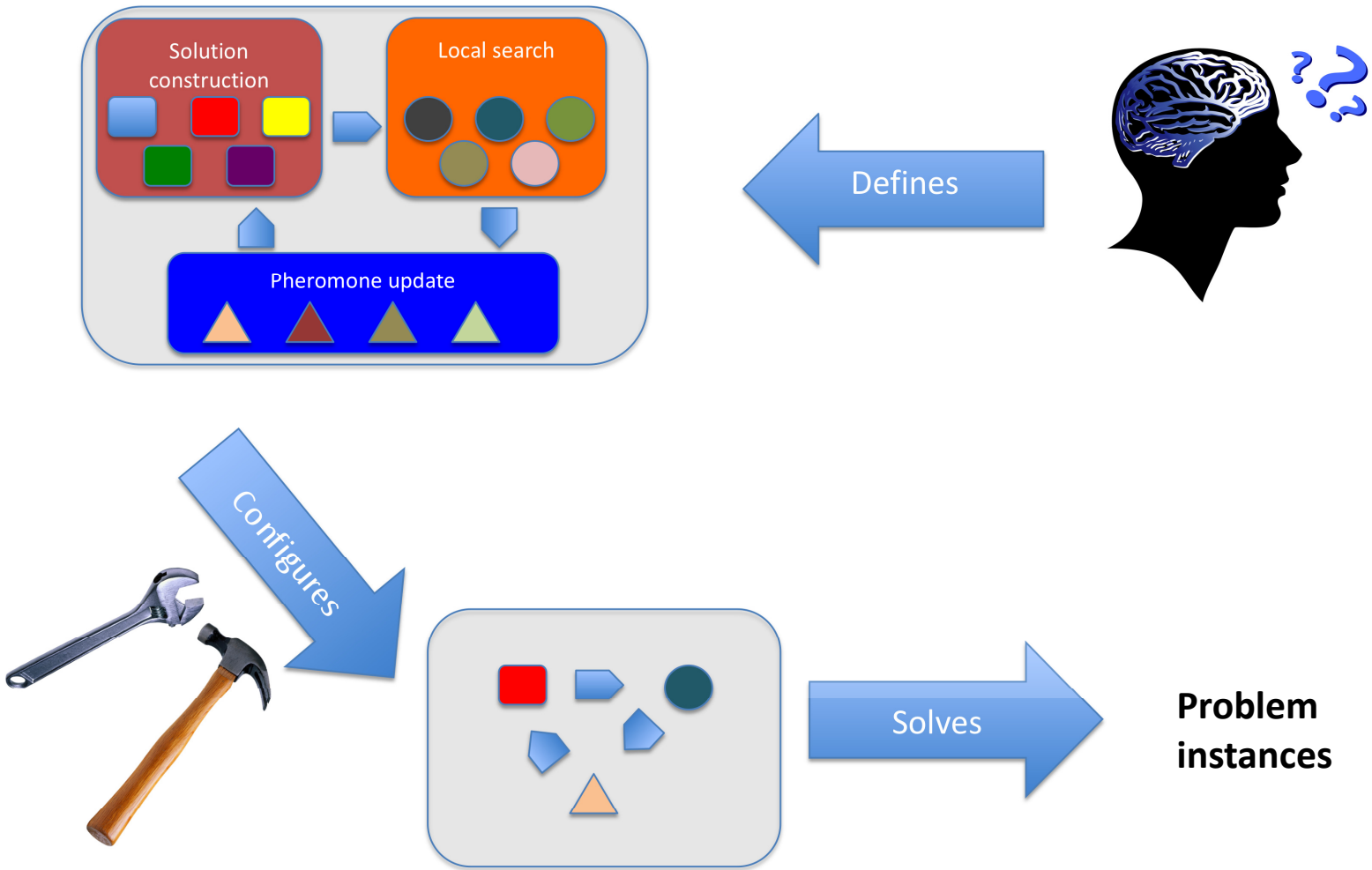
Towards a shift of paradigm in algorithm design







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
- is a promising tool with potential high impact

Future work

- more powerful configurators
- configurable frameworks XXL
- 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)