Automatically Configuring Algorithms
—
Some recent work

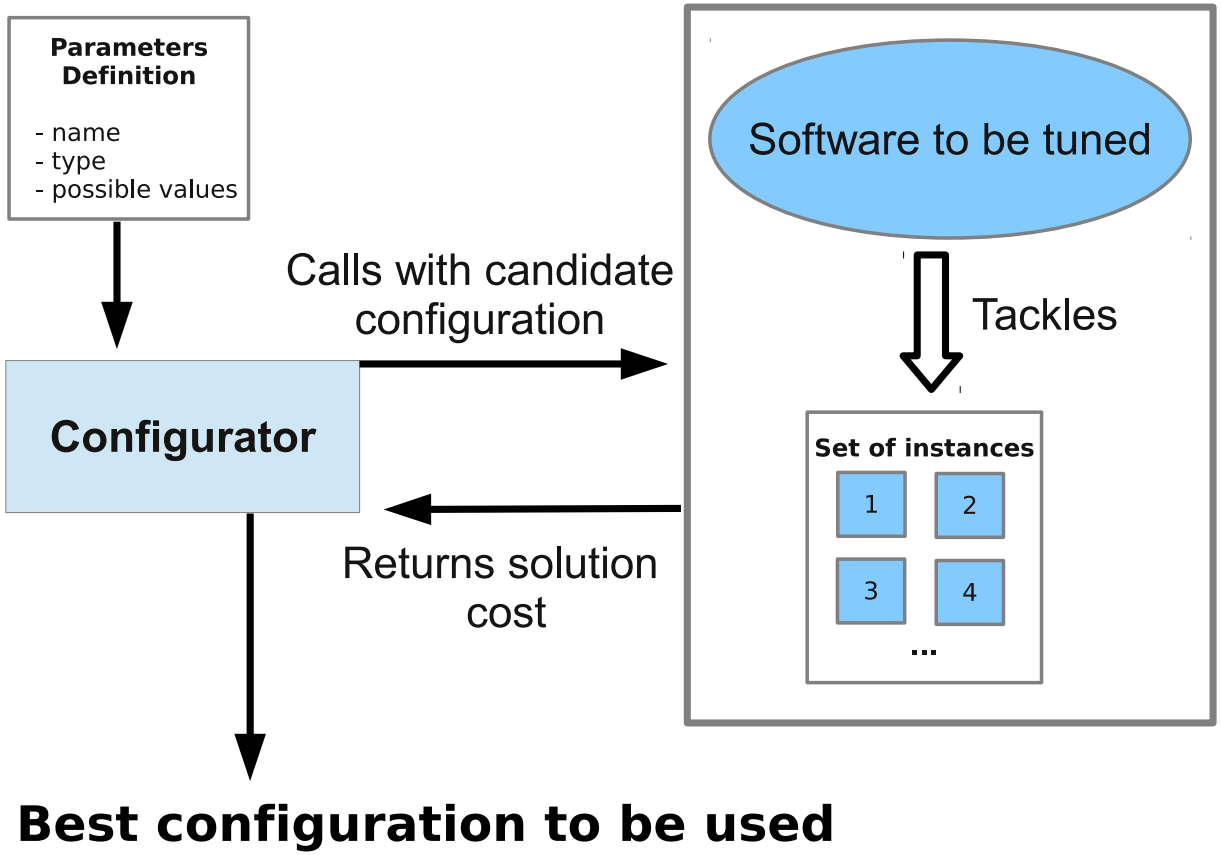
Thomas Stützle, Leonardo Bezerra, Jérémie Dubois-Lacoste, Tianjun Liao, Manuel López-Ibáñez, Franco Mascia and Leslie Perez Caceres

IRIDIA, CoDE, Université Libre de Bruxelles
Brussels, Belgium

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

Some recent researches where we made (heavy) use of automatic algorithm configuration

Automated offline configuration



Typical performance measures

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

Approaches to configuration

- 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 & students, 2007, 2009, 2010] ...
- experimental design techniques
  - e.g. CALIBRA [Adenso–Díaz, Laguna, 2006], [Ridge&Kudenko, 2007], [Coy et al., 2001], [Ruiz, Stützle, 2005]
- model-based optimization approaches
  - e.g. SPO [Bartz-Beielstein et al., 2005, 2006, ..], *SMAC* [Hutter et al., 2011, ..]
- 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

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 racing

sample configurations from initial distribution

While not terminate()

apply race
modify sampling distribution
sample configurations






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.

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

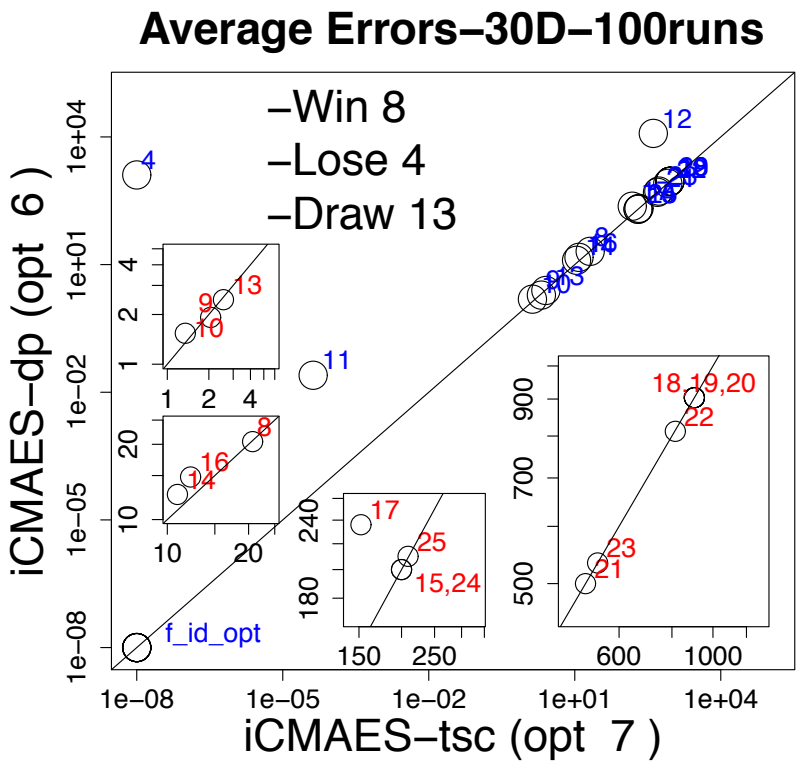
- implementation of Iterated Racing in R (but no knowledge of R necessary)
  - Goal 1: flexible
  - Goal 2: easy to use
- Parallel evaluation (MPI, multi-cores, grid engine ..)
- initial candidates

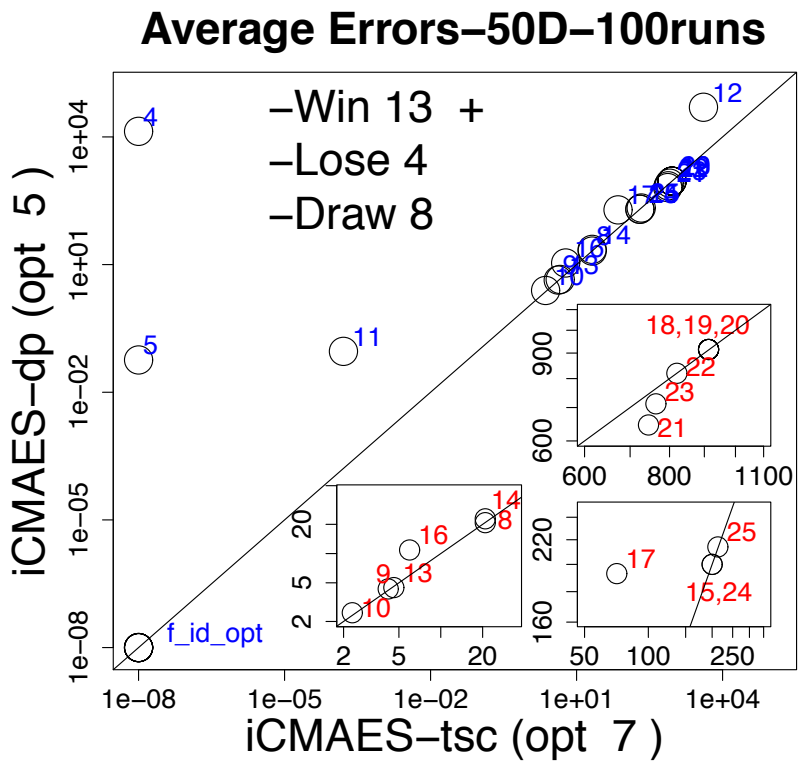
Applications in black-box continuous optimization




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





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

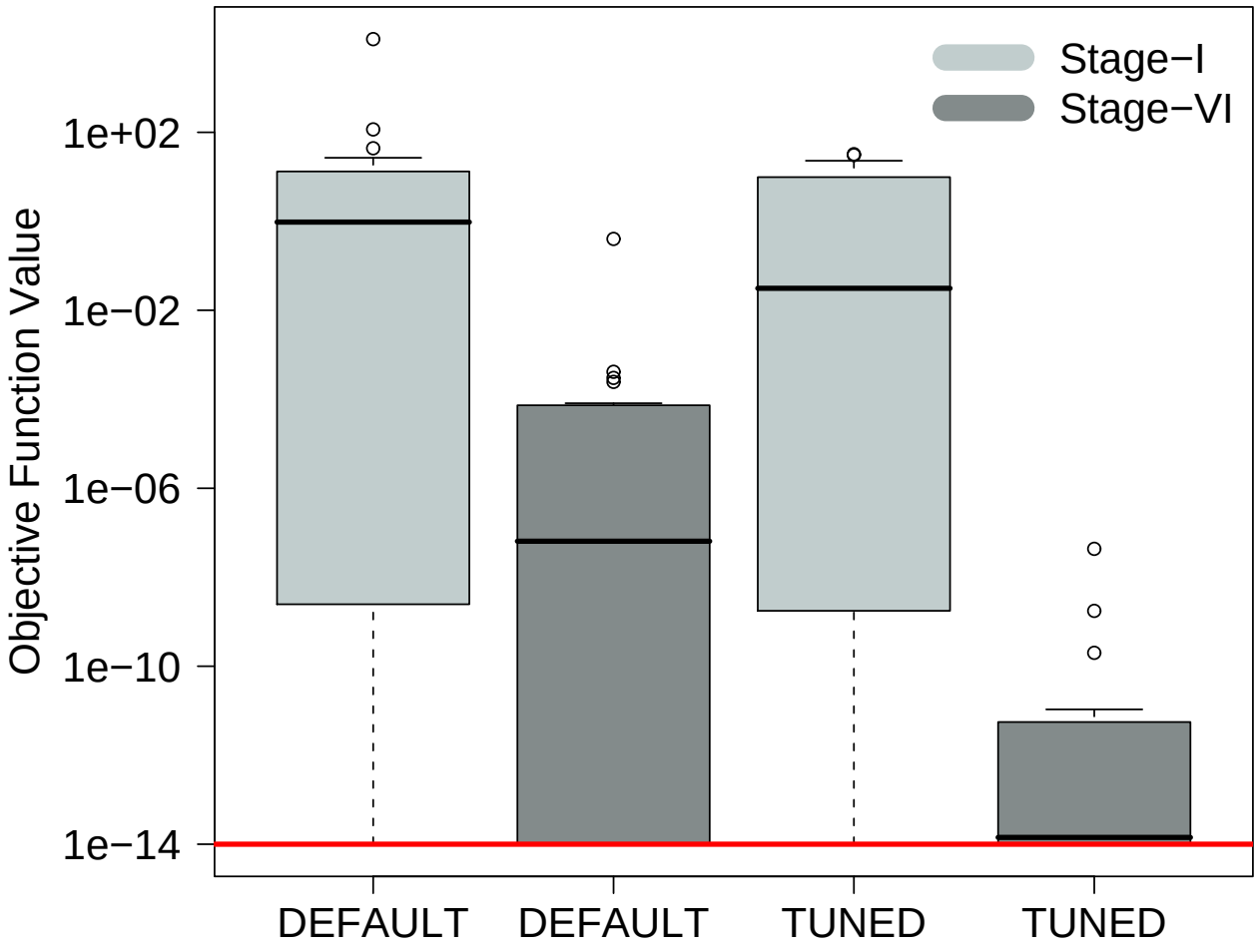
[Montes de Oca, Aydın, Stützle, 2011]

- 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]



Comparison of continuous optimizers

[Liao, Molina, Stützle, 2012]

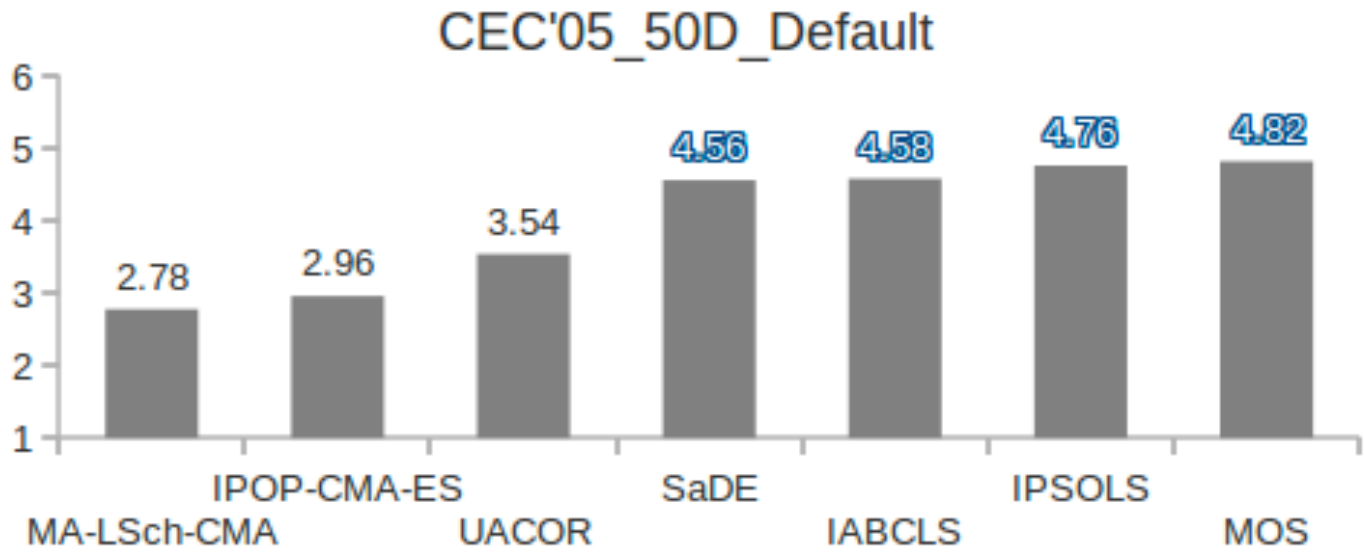
Solvers

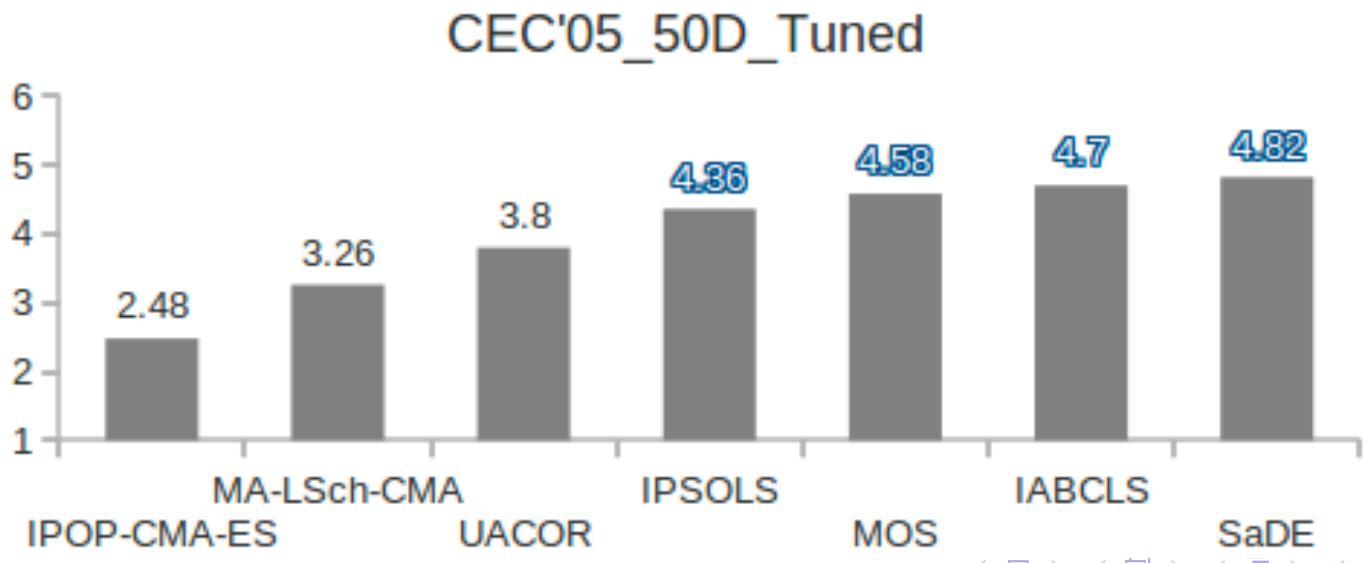
- IPOP-CMAES (evolution strategy)
- MA-LSch-CMA (memetic algorithm)
- UACOR (ant colony optimization)
- IPSOLS (particle swarm optimization)
- IABCLS (artificial bee colony)
- SaDE (self-adaptive differential evolution)
- MOS (differential evolution, local search hybrid)

Benchmark sets

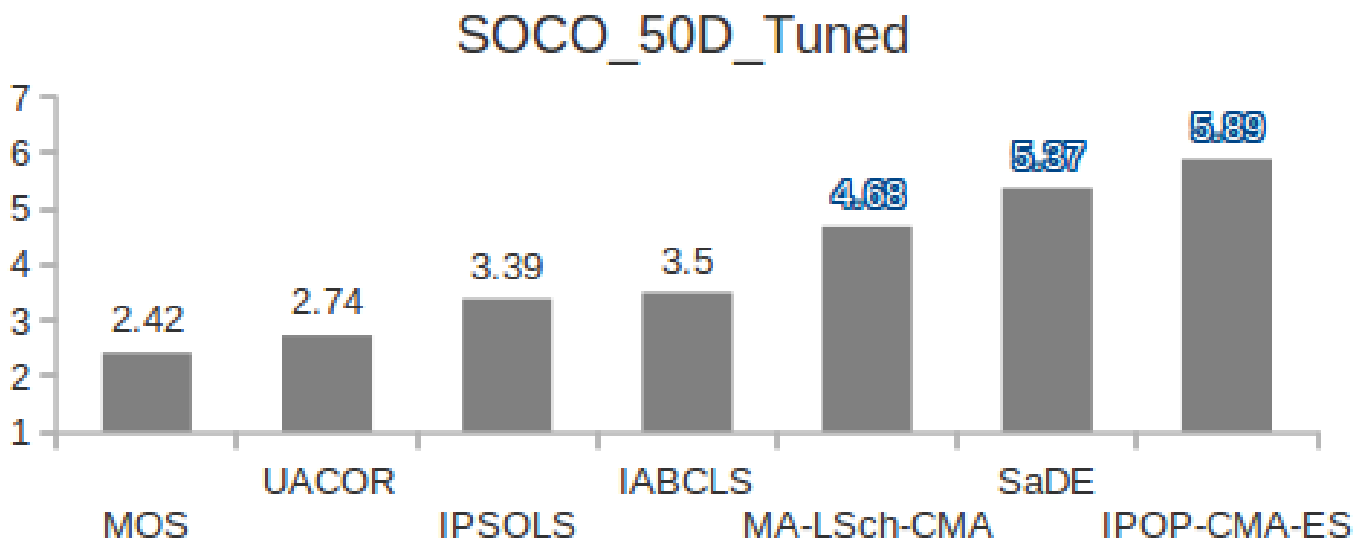
- CEC'05 benchmark set (25 functions, scalable)
- Soft Computing special issue (19 functions, scalable)

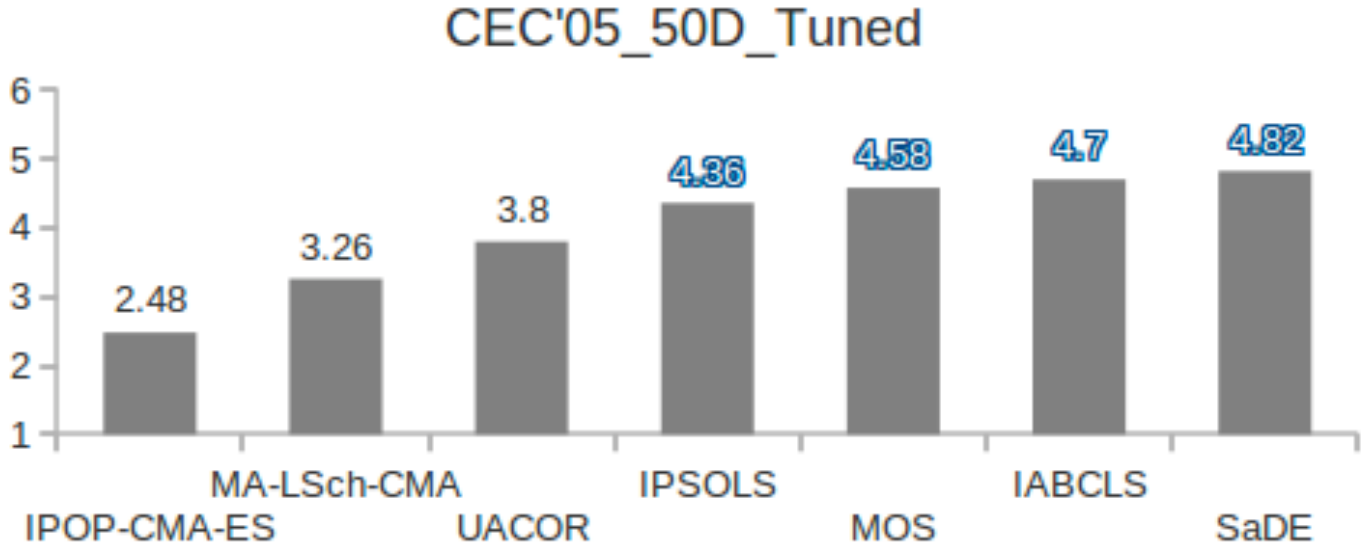
Ranking default vs. tuned, example CEC'05





Ranking on SOCO vs. CEC'05 benchmark sets





Configuring continuous function classes

- family of continuous functions
- example, Rastrigin

$$nA + \sum_{i=1}^{n} (x_i^2 - A\cos(\omega x_i))$$

- generate different values for A, ω, n such that some properties are maintained (e.g. landscape featues
- configure continuous algorithms for it
- some extensions on the way ...

Example, design configurable algorithm framework

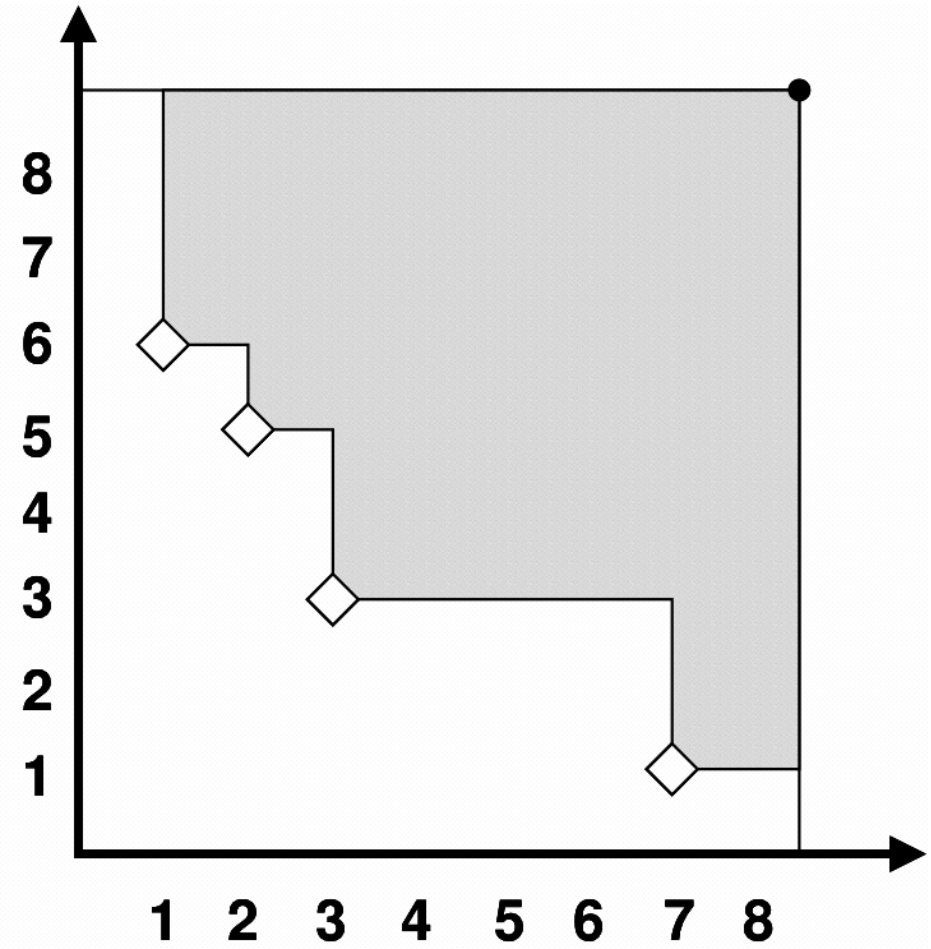
Multi-objective ant colony optimization (MOACO)


MOACO framework

López-Ibáñez, Stützle, 2012

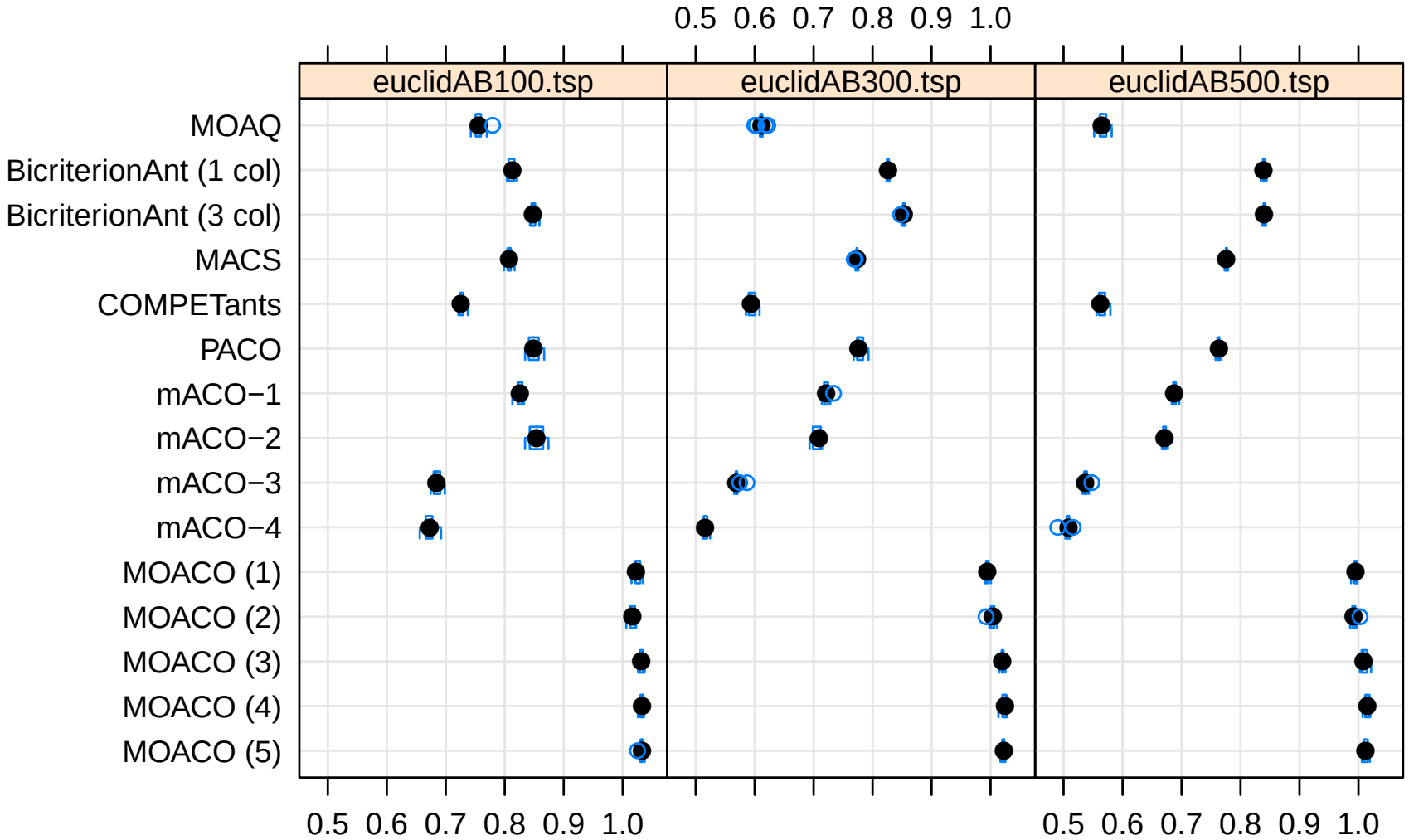
- algorithm framework for multi-objective ACO algorithms
- can instantiate MOACO algorithms from literature
- 10 parameters control the multi-objective part
- 12 parameters control the underlying pure "ACO" part

MOACO framework

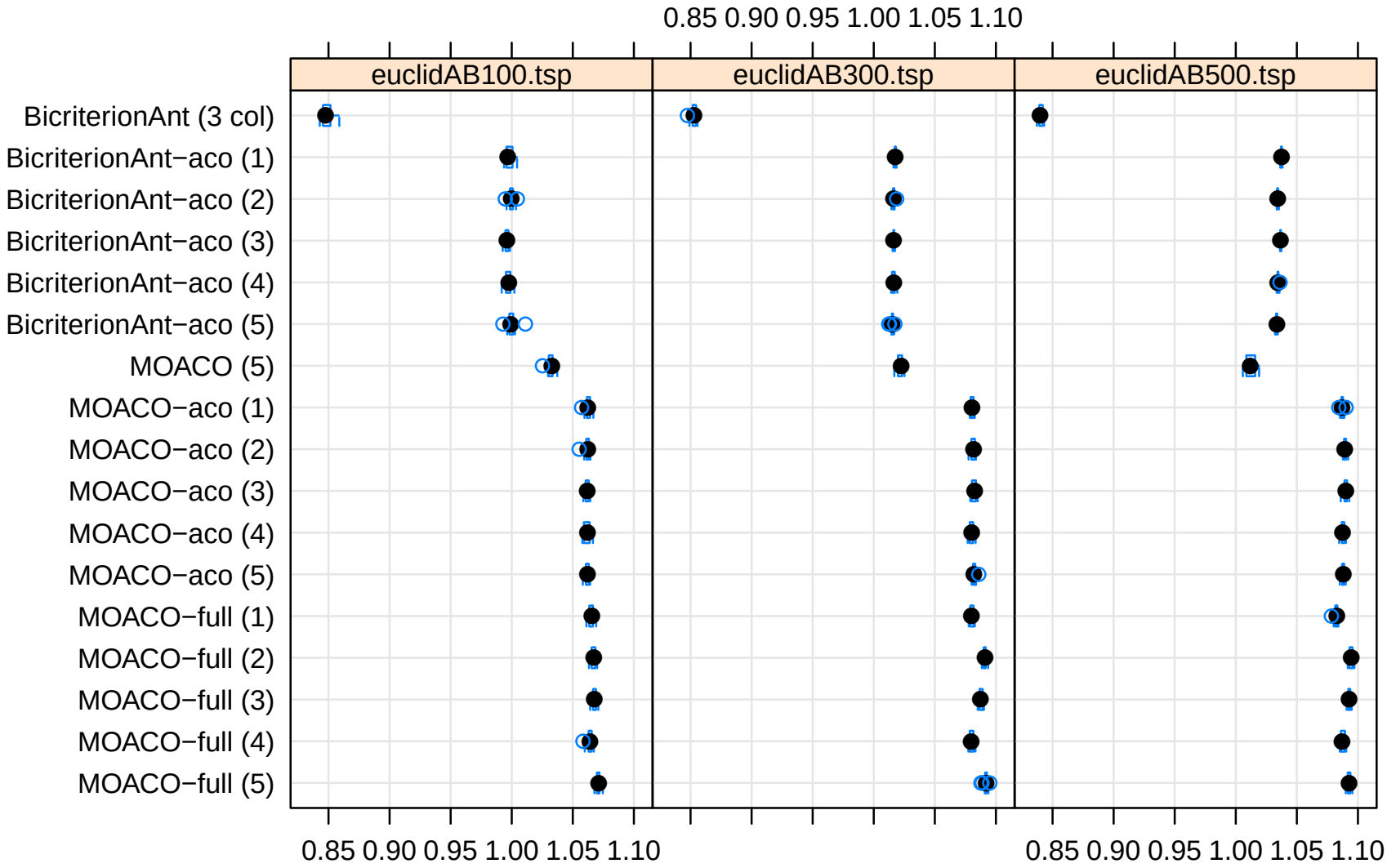


irace + *hypervolume* = *automatic configuration of multi-objective solvers!*

Automatic configuration multi-objective ACO



Automatic configuration multi-objective ACO



Summary

- 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
  - Single-objective frameworks for cont. opt. UACOR, ABC
  - Multi-objective algorithm frameworks (TP+PLS, MOEA)

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

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}
```

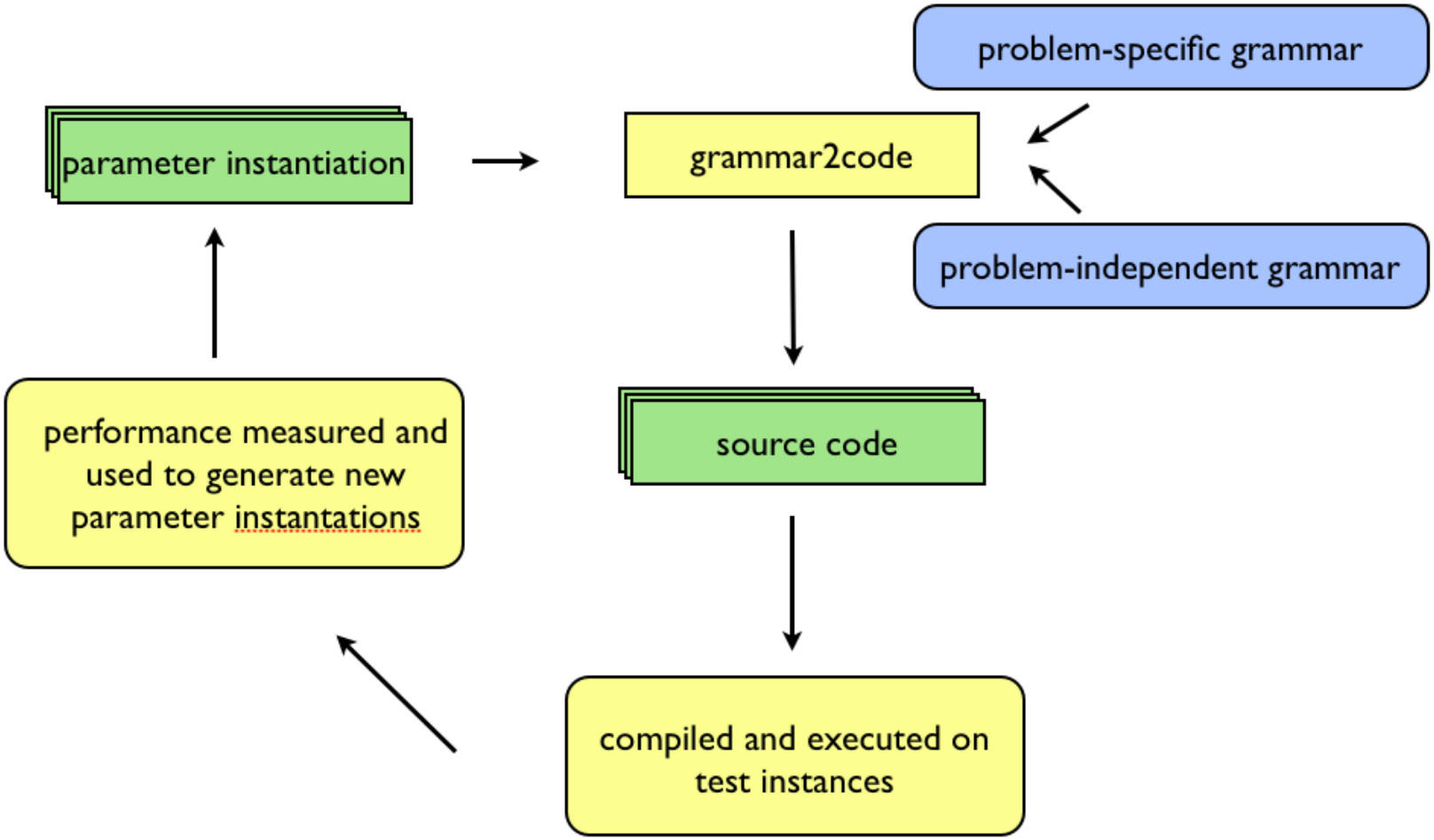
Grammar

```
     <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

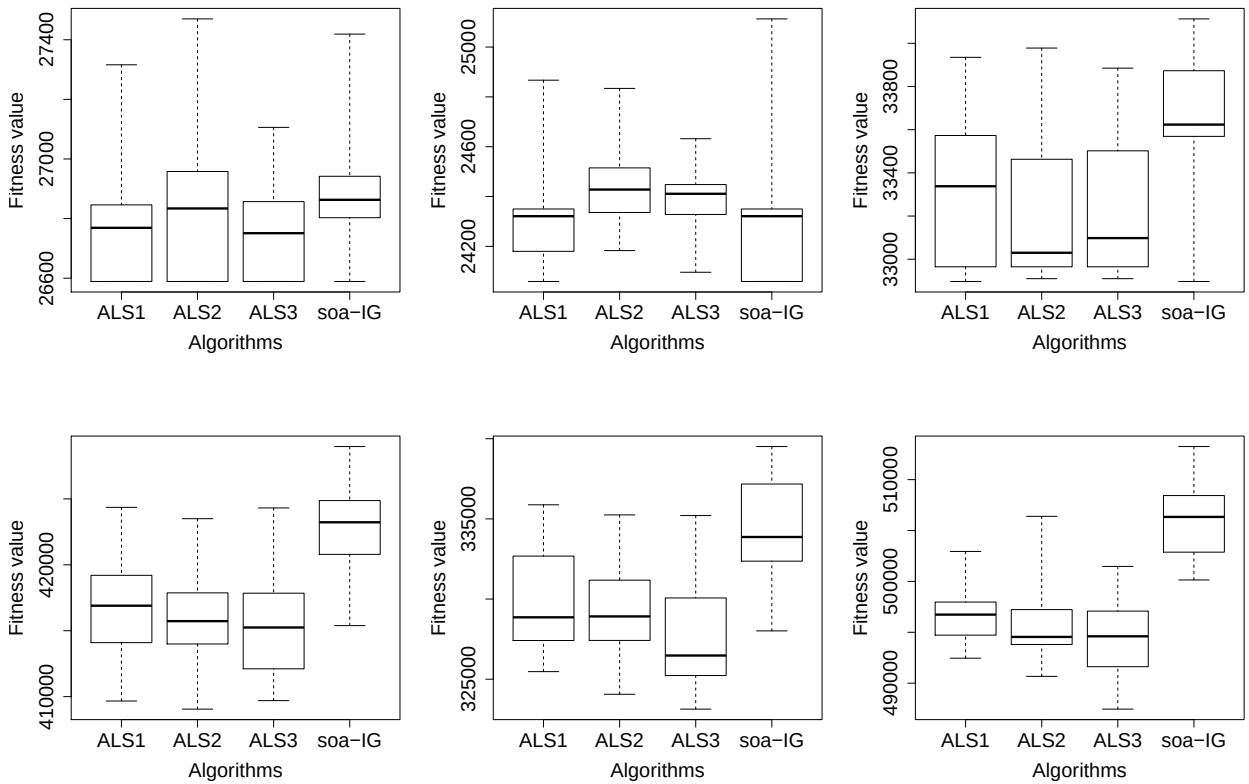
```
     <algorithm> ::= <initialization> <ils>
<initialization> ::= random | <pbs_initialization>
          <ils> ::= ILS(<perturb>, <ls>, <accept>, <stop>)
<perturb> ::= none | <initialization> | <pbs_perturb>
     <ls> ::= <ils> | <descent> | <sa> | <rii> | <pii> | <vns> | <ig> | <pbs_ls
 <accept>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



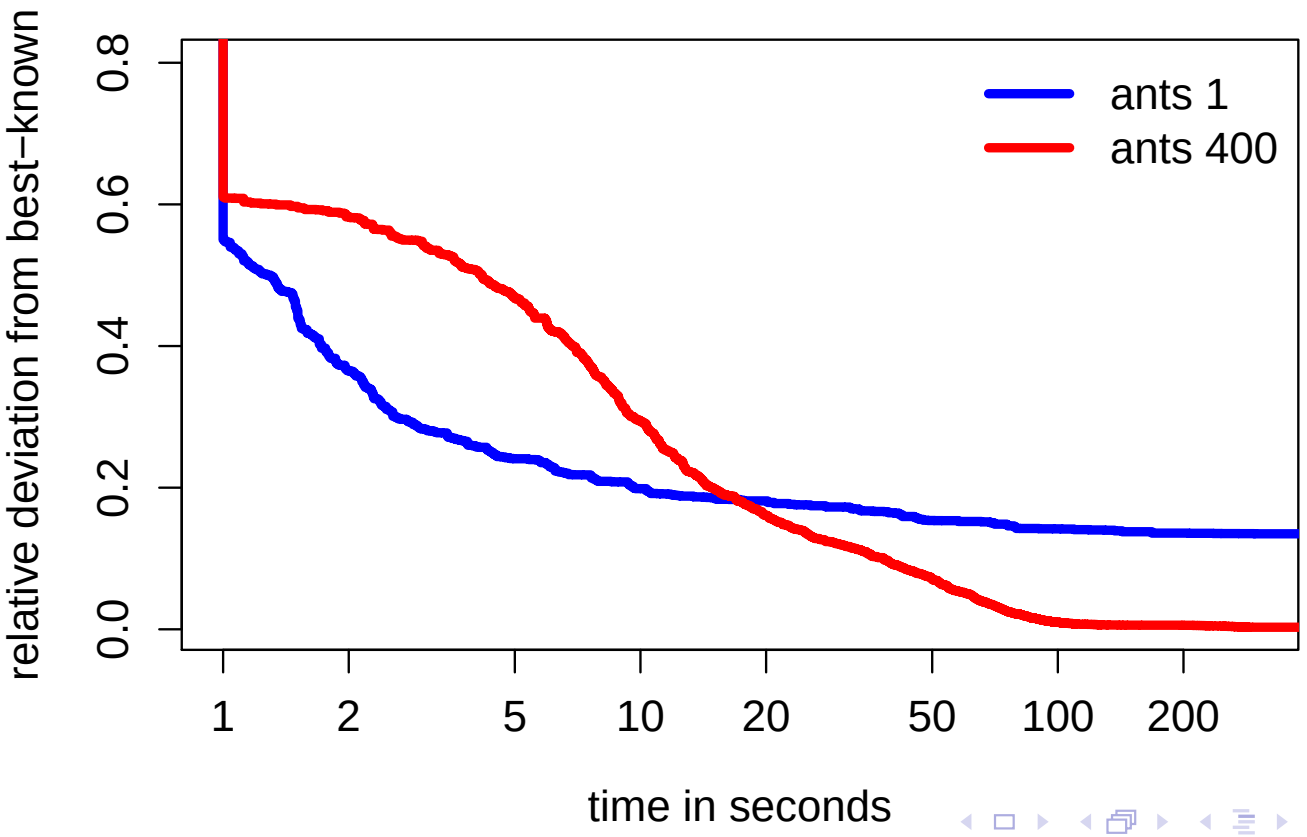
results are competitive or superior to state-of-the-art algorithm

Example, new applications

Improving automatically the anytime behavior of algorithms

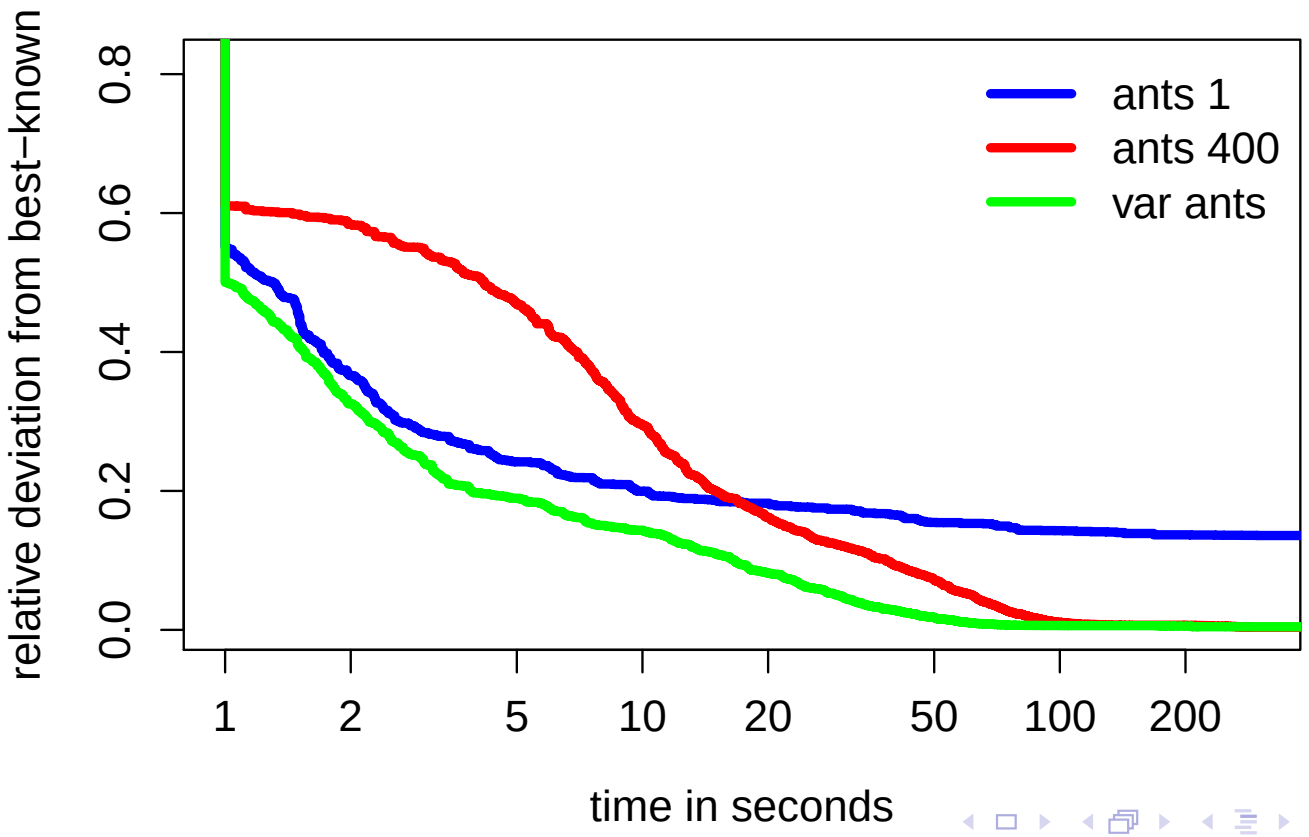

"Anytime" Algorithms [Zilberstein, 1996]

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



“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

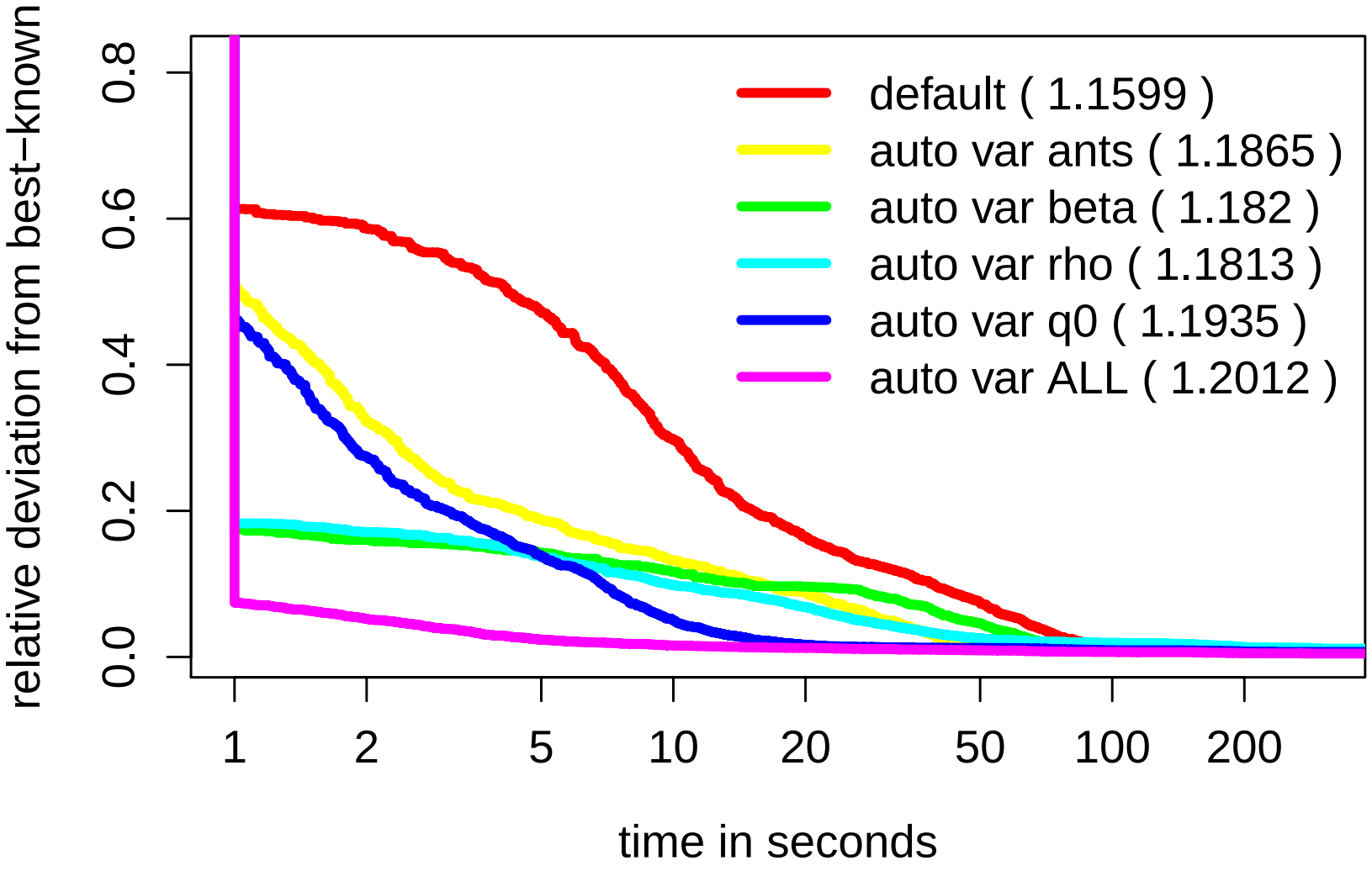
New approach

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

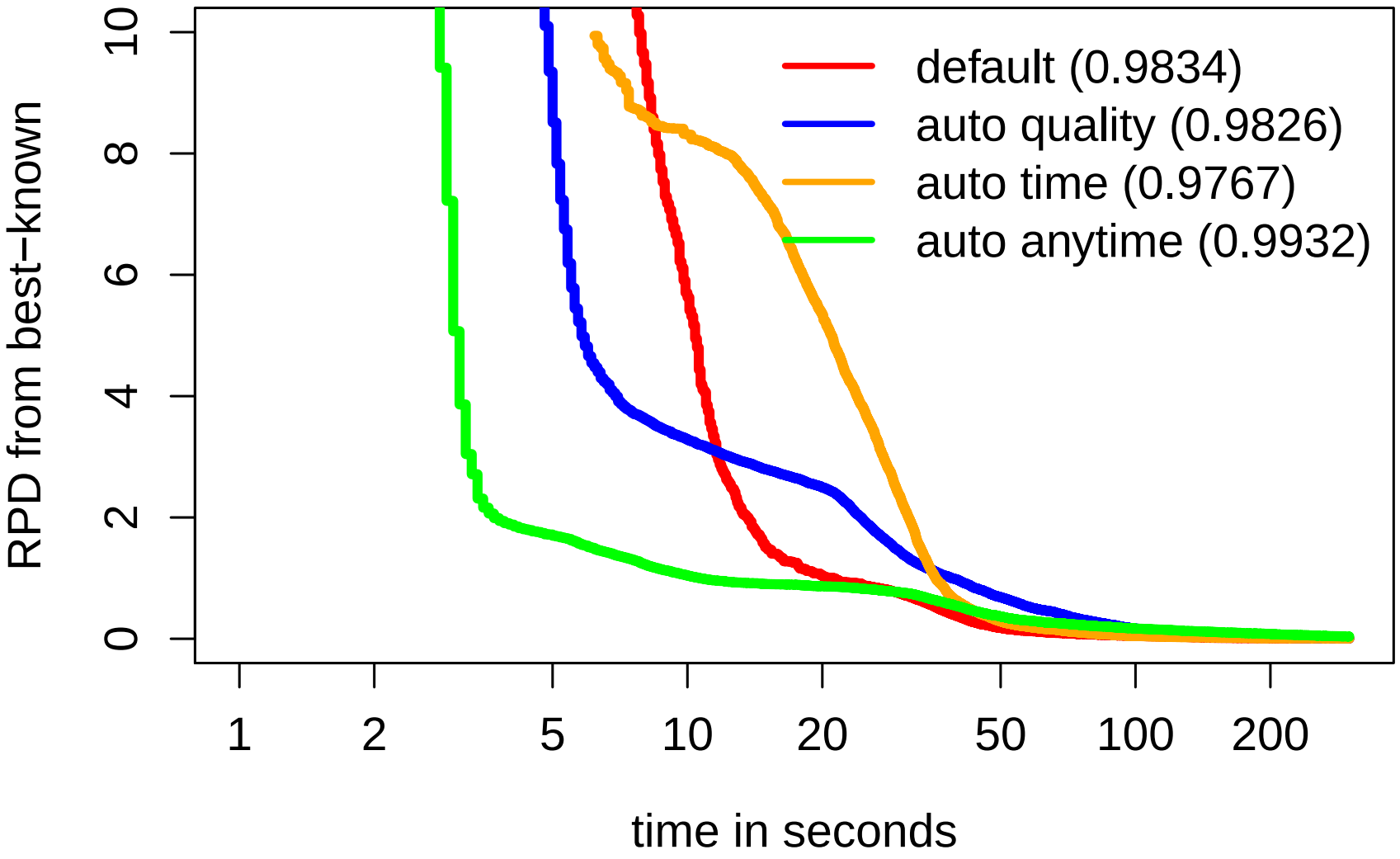
- Less effort: 1 week instead of a year!
- Same or even better results
- Improving the anytime behaviour of metaheuristics becomes *much easier*

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

Improving anytime behavior of SCIP



Concluding remarks

- using automatic configuration is rewarding in terms of development time and algorithm performance
- can (easily) be integrated into algorithm engineering tasks
- bottom-up approach probably scales better with algorithm complexity
- automatic configuration will have profound impact on how optimization software is developed

Future work

- more powerful configurators
- more and more complex applications
- promote usage of configurators

Questions

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