Automated Algorithm Configuration: Recent Advances and Prospects

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

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

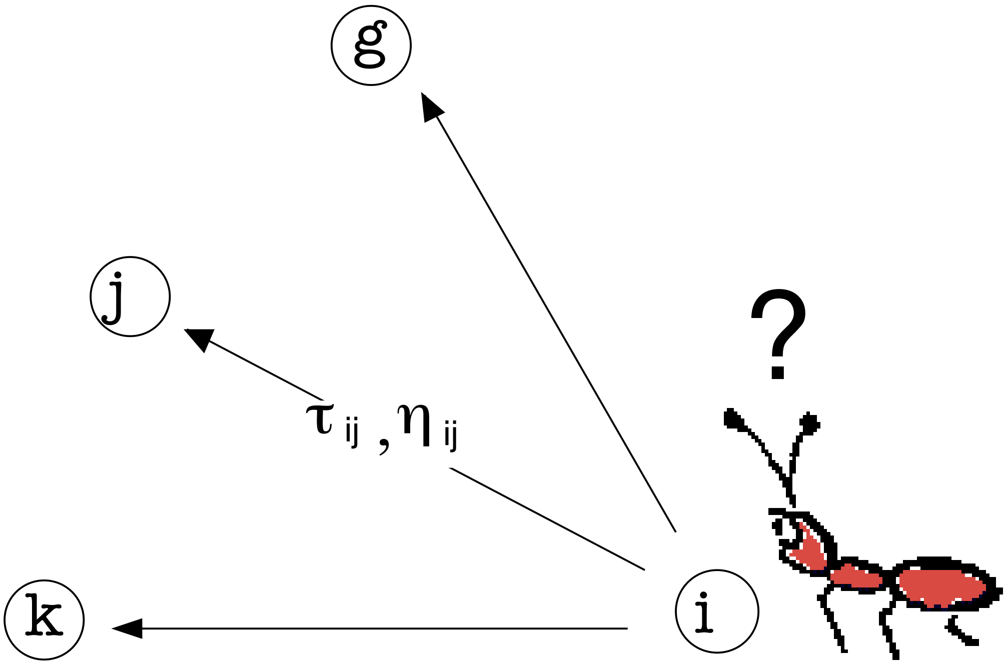
Much active research on hybrids between exact and approximate algorithms!

Design choices and parameters everywhere

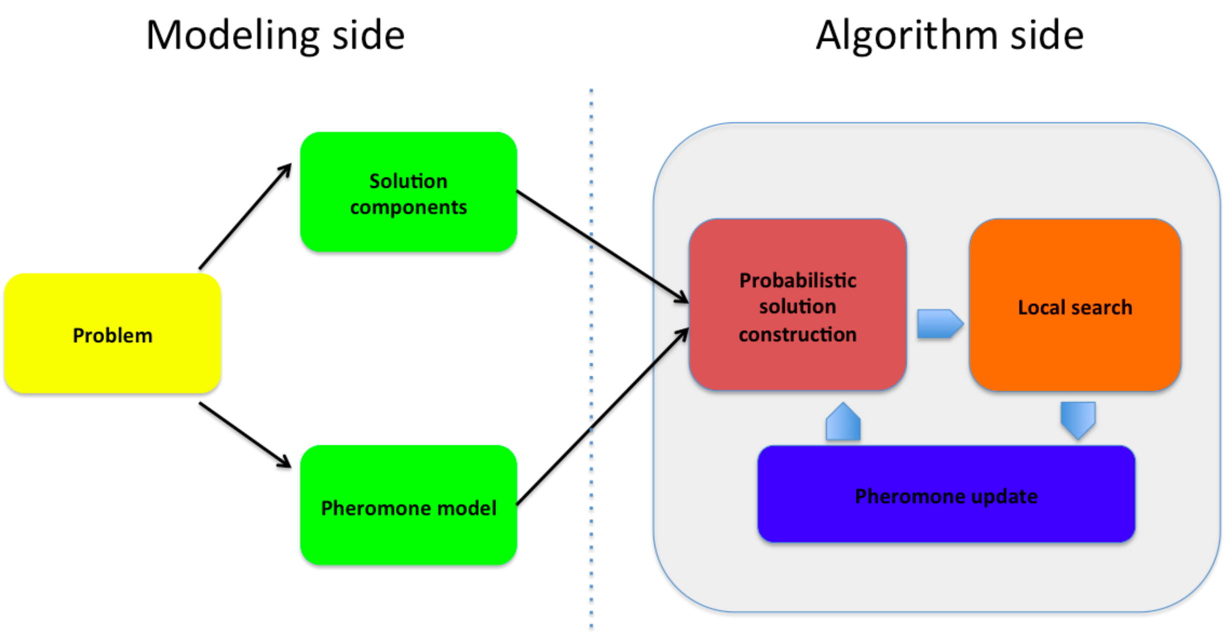
Todays high-performance optimizers involve a large number of design choices and parameter settings

- 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: multi-objective ACO algorithms with 22 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*

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
  $\leadsto$ 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

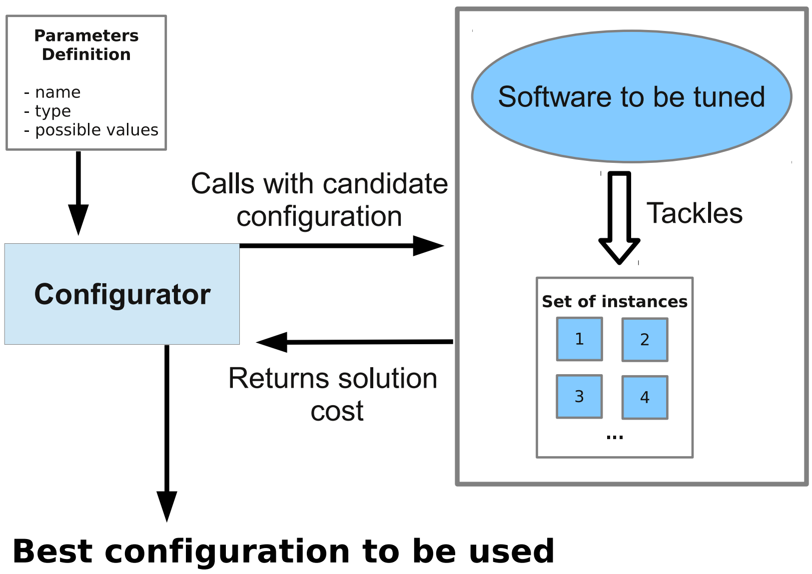
- configure algorithm before deploying it
- configuration on training instances
- related to algorithm design

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



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, . . .]

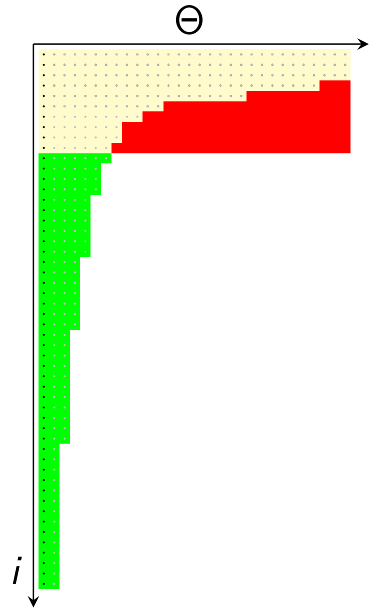
General, domain-independent methods required: (i) applicable to all variable types, (ii) multiple training instances, (iii) high performance, (iv) scalable

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

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

Other tools: ParamILS, SMAC

ParamILS

- iterated local search in configuration space
- requires discretization of numerical parameters
- http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/

SMAC

- surrogate model assisted search process
- encouraging results for large configuration spaces
- http://www.cs.ubc.ca/labs/beta/Projects/SMAC/

capping: effective speed-up technique for configuration target run-time

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]

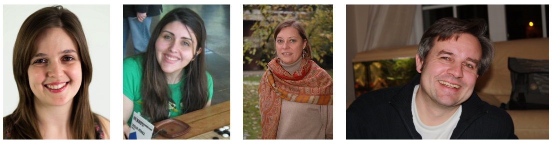
- powerful commercial (e.g. CPLEX) and non-commercial (e.g. SCIP) solvers available
- large number of parameters (tens to hundreds)
- default configurations not necessarily best for specific problems

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

Example, configuration of Branch & Cut & Price algorithm

Branch&Cut&Price algorithm



Branch & Cut & Price for Network Pricing

[Violin Phd thesis, 2014; joint work with Labbé, Perez, Stützle]

- Branch & Cut & Price algorithm for a network pricing problem
- 15 categorical, 4 real and 1 integer parameters are tuned
- tuning with irace

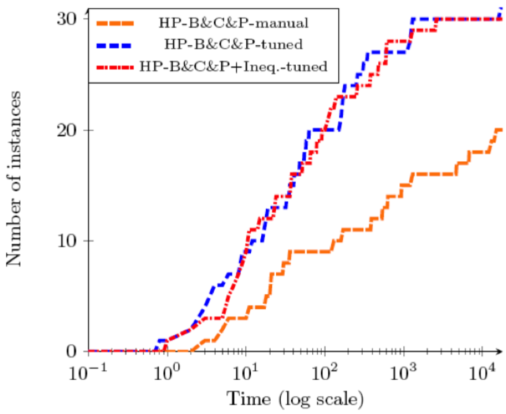


Figure 1: Performance profile graphs on solving (HP) for complete graph instances, comparing different configurations of HP-B&C&P

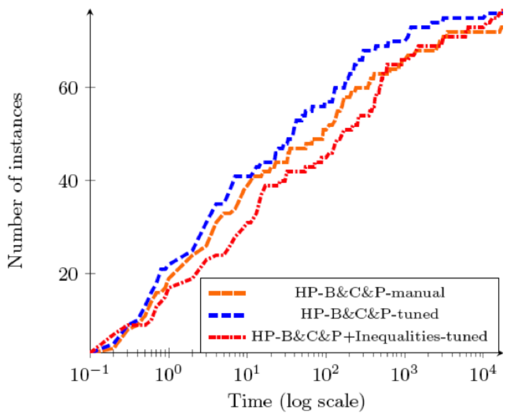


Figure 2: Performance profile graphs on solving (HP) for A1 instances,comparing different configurations of HP-B&C&P

Example, supporting tool in algorithm engineering

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

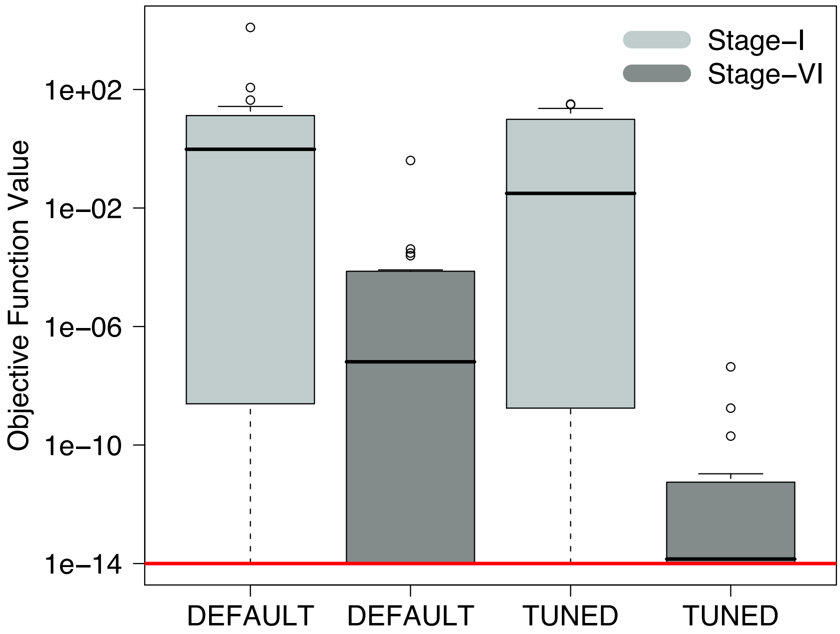
Tuning in-the-loop (re)design of continuous optimizers
[Montes de Oca, Aydın, Stützle, 2011]

- 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

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}
```

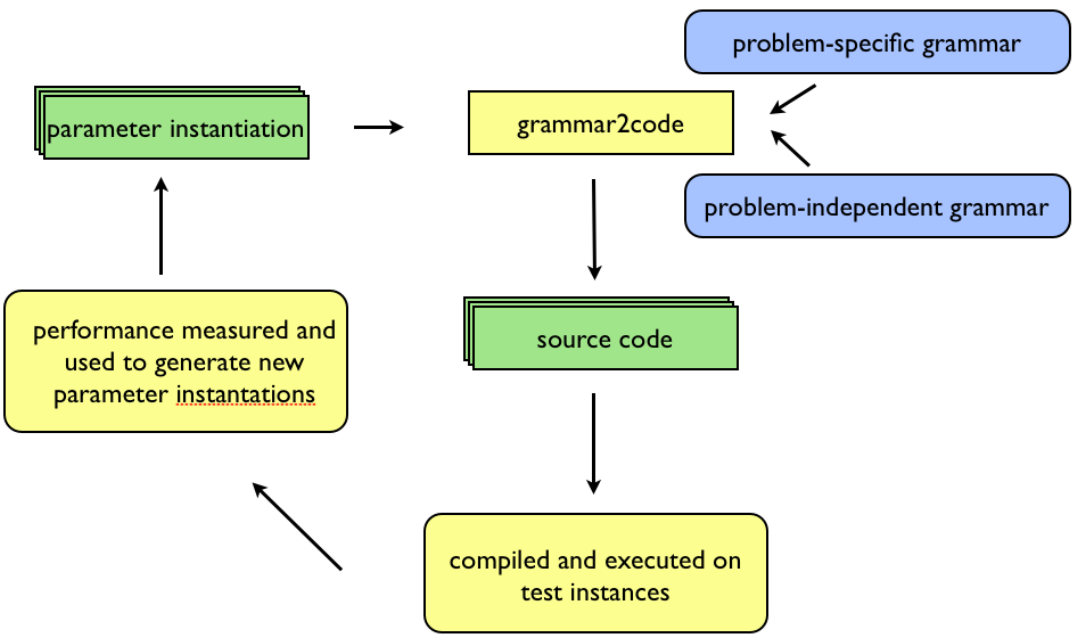

Grammar

```
     <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

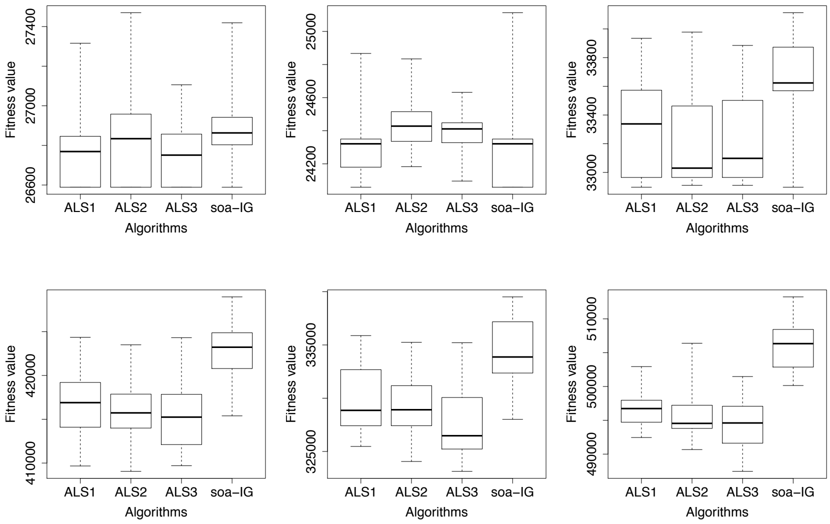
```
     <algorithm> ::= <initialization> <ils>
<initialization> ::= random | <pbs_initialization>
           <ils> ::= ILS(<perturb>, <ls>, <accept>, <stop>)
<perturb> ::= none | <initialization> | <pbs_perturb>
     <ls> ::= <ils> | <descent> | <sa> | <rii> | <pii> | <vns> | <ig> | <pbs_ls
 <accept> ::= 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 × 10 000 trials each of 30 seconds



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

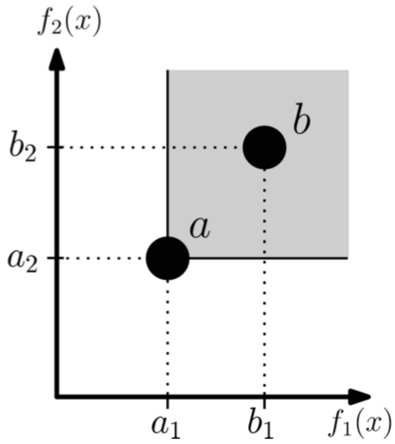
Example, design configurable algorithm framework

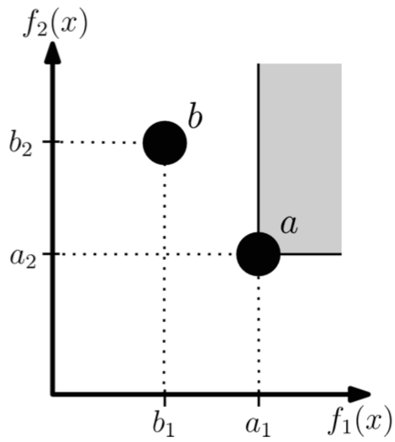
Multi-objective ant colony optimization (MOACO)

Multi-objective Optimization

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





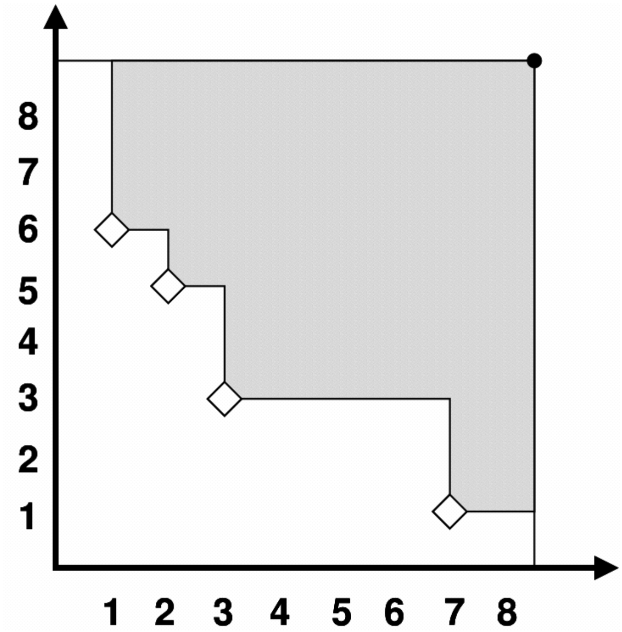
MOACO framework

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

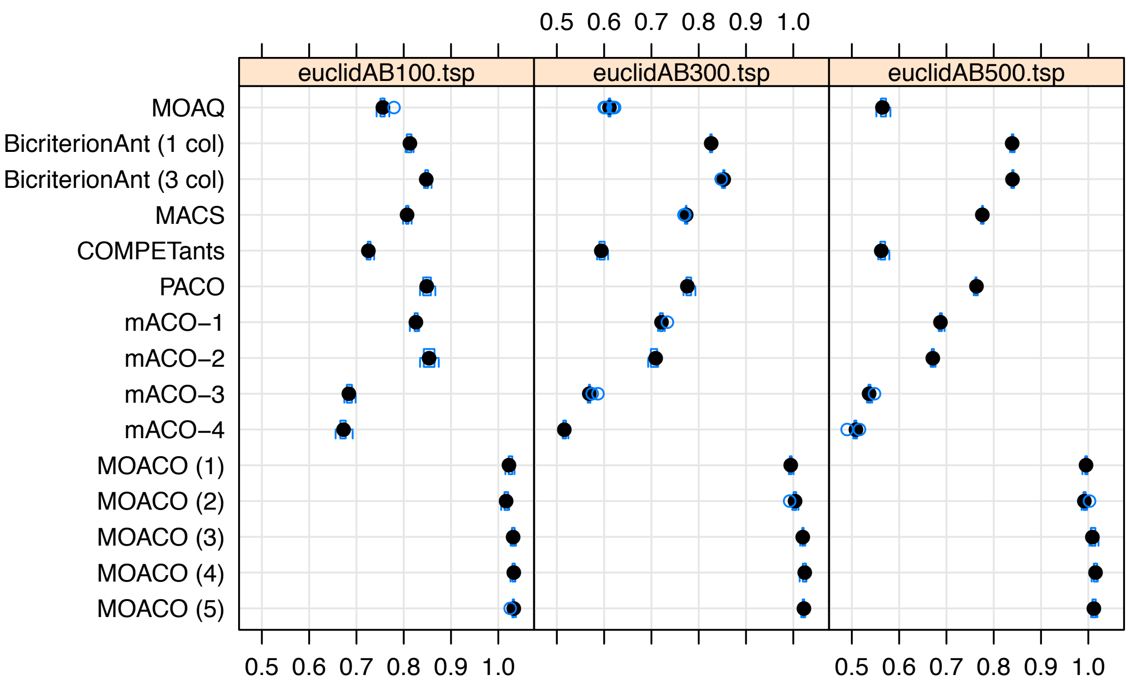
Example of a top-down approach to algorithm configuration

MOACO framework

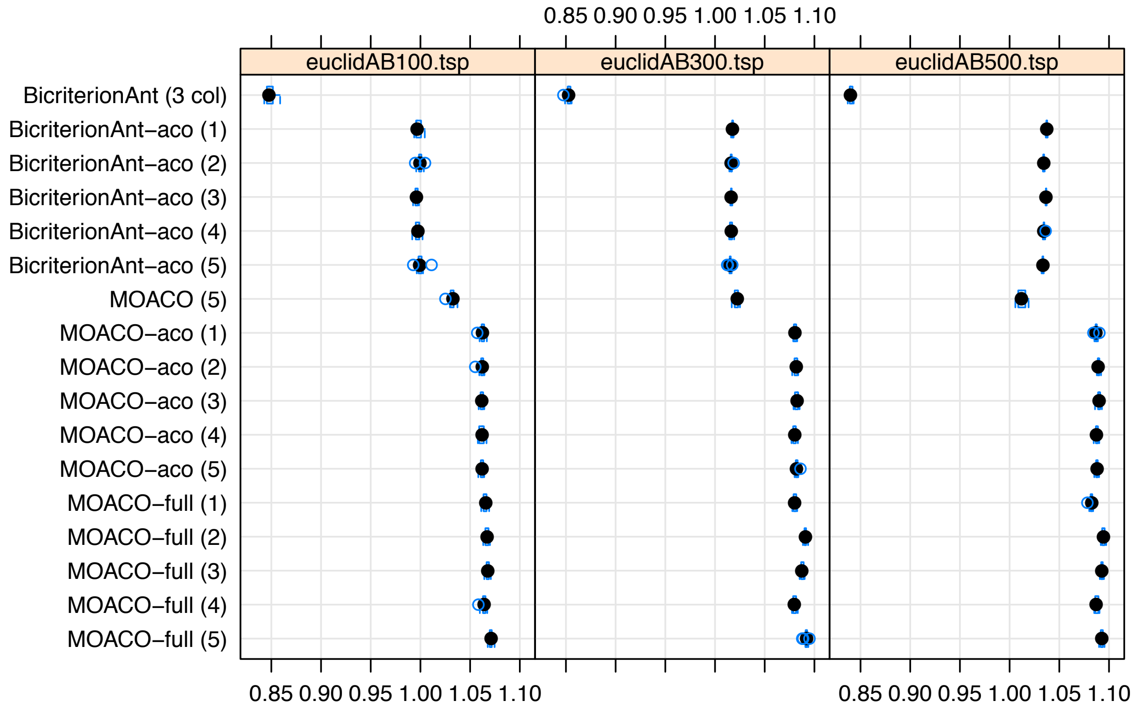


irace + hypervolume = automatic configuration of multi-objective solvers!

Automatic configuration multi-objective ACO



Automatic configuration multi-objective ACO



Summary

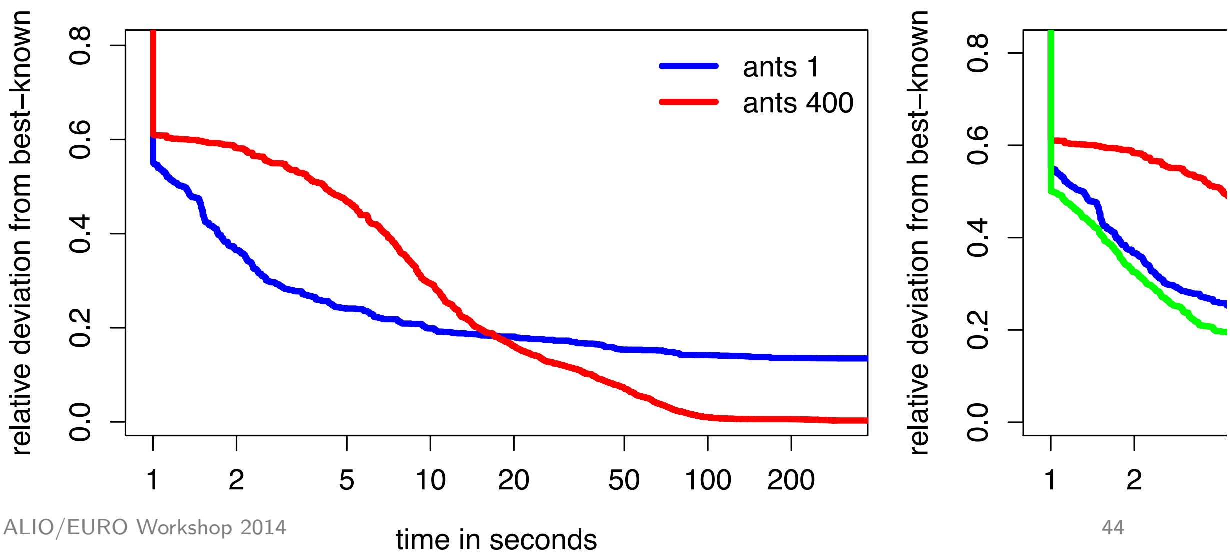
- ~~We propose a new MOACO algorithm that. . .~~
- We propose an approach to automatically design MOACO algorithms:
  1. Synthesize state-of-the-art knowledge into a flexible MOACO framework
  2. Explore the space of potential designs automatically using irace
- Other examples:
  - Single-objective frameworks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

Example, new applications

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

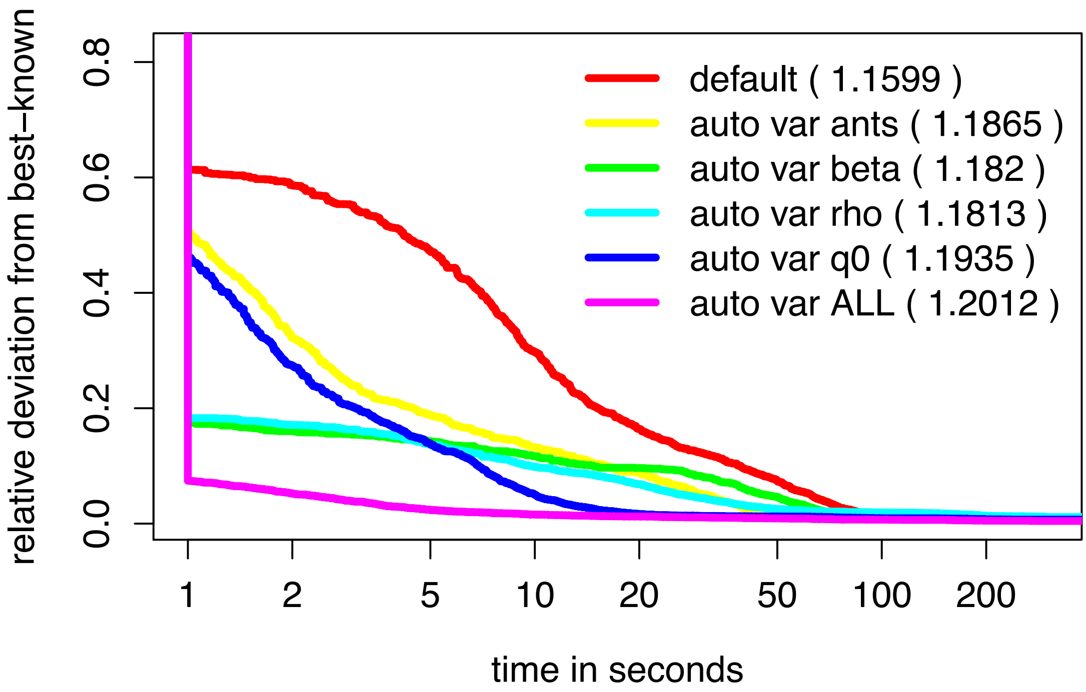
New approach

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*

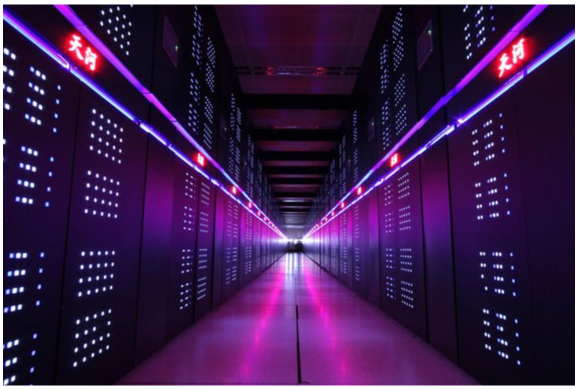
We can use offline configuration of online strategies for improving anytime behaviour

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

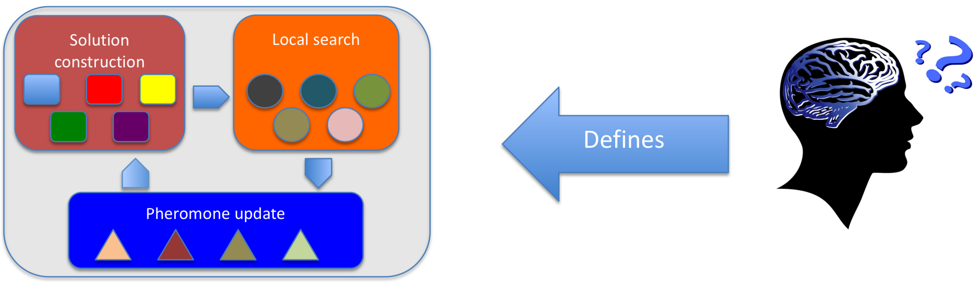
Remark: We improved anytime behavior also for SCIP solver v.2.1.0 configuring more than 200 parameters as proof of concept.

Why automatic algorithm configuration?

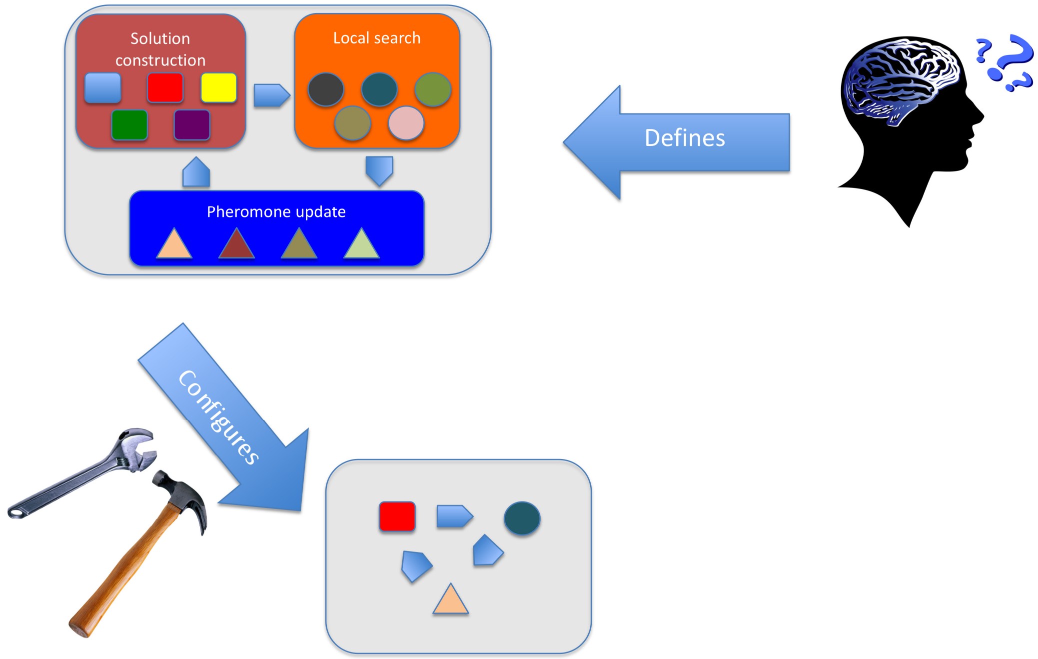
- improvement over manual, ad-hoc methods for tuning
- reduction of development time and human intervention
- increase number of considerable degrees of freedom
- empirical studies, comparisons of algorithms
- support for end users of algorithms



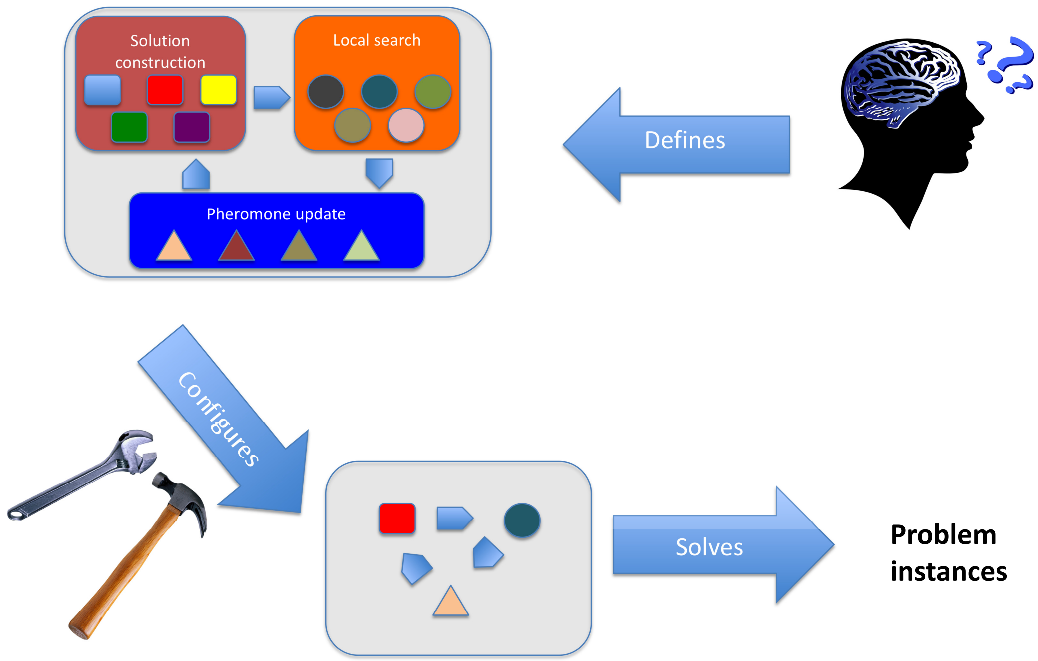
Towards a shift of paradigm in algorithm design



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

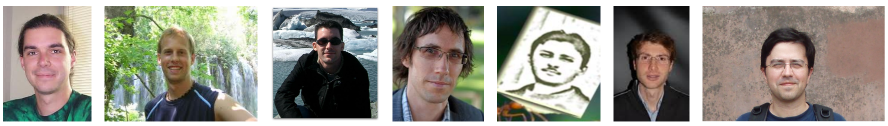
Future work

- more powerful configurators
- more and more complex applications
- exploitation of data gained
- best practice

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)