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

**Today's 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
    ▶ $\alpha, \beta$ 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
    ▶ $\rho$: evaporation rate
    ▶ $\tau_0$: initial pheromone level

▶ local search
  ▶ ... many more ...
Parameter types

- **categorical** parameters
  - choice of constructive procedure, choice of recombination operator, choice of branching strategy, . . .

- **ordinal** parameters
  - neighborhoods, lower bounds, . . .

- **numerical** parameters
  - 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*
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 et 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

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

- Design space definition
  - Algorithmic components
  - Rules for combining components

- Parameter space

- Automatic Parameter Configuration

- Instances

- Effective Algorithm
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 $\leadsto$ 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
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

\[\text{irace} + \text{hypervolume} = \text{automatic configuration of multi-objective solvers!}\]
Automatic configuration multi-objective ACO

![Graph showing performance of different algorithms on three datasets: euclidAB100.tsp, euclidAB300.tsp, euclidAB500.tsp. The x-axis represents the threshold value, and the y-axis shows the performance. The algorithms include MOAQ, BicriterionAnt (1 col), BicriterionAnt (3 col), MACS, COMPETants, PACO, mACO−1, mACO−2, mACO−3, mACO−4, MOACO (1), MOACO (2), MOACO (3), MOACO (4), MOACO (5).]
### Automatic configuration multi-objective ACO

<table>
<thead>
<tr>
<th>MOACO−full (5)</th>
<th>MOACO−full (4)</th>
<th>MOACO−full (3)</th>
<th>MOACO−full (2)</th>
<th>MOACO−full (1)</th>
<th>MOACO−aco (5)</th>
<th>MOACO−aco (4)</th>
<th>MOACO−aco (3)</th>
<th>MOACO−aco (2)</th>
<th>MOACO−aco (1)</th>
<th>BicriterionAnt−aco (5)</th>
<th>BicriterionAnt−aco (4)</th>
<th>BicriterionAnt−aco (3)</th>
<th>BicriterionAnt−aco (2)</th>
<th>BicriterionAnt−aco (1)</th>
<th>BicriterionAnt (3 col)</th>
<th>euclidAB100.tsp</th>
<th>euclidAB300.tsp</th>
<th>euclidAB500.tsp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.85 0.90 0.95 1.00 1.05 1.10</td>
<td>0.85 0.90 0.95 1.00 1.05 1.10</td>
<td>0.85 0.90 0.95 1.00 1.05 1.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Graphical representation of performance metrics for different algorithms and problem instances.
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

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

<table>
<thead>
<tr>
<th>Component</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preference</td>
<td>〈 Set-partitioning, Quality, Diversity 〉</td>
</tr>
<tr>
<td>BuildMatingPool</td>
<td>〈 Preference_Mat, Selection 〉</td>
</tr>
<tr>
<td>Replacement</td>
<td>〈 Preference_Rep, Removal 〉</td>
</tr>
<tr>
<td>Replacement_Ext</td>
<td>〈 Preference_Ext, Removal_Ext 〉</td>
</tr>
</tbody>
</table>
## Representing known MOEAs

<table>
<thead>
<tr>
<th>Alg.</th>
<th>BuildMatingPool</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SetPart</td>
<td>Quality</td>
</tr>
<tr>
<td>MOGA</td>
<td>rank</td>
<td>—</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>depth</td>
<td>—</td>
</tr>
<tr>
<td>SPEA2</td>
<td>strength</td>
<td>—</td>
</tr>
<tr>
<td>IBEA</td>
<td>—</td>
<td>binary</td>
</tr>
<tr>
<td>HypE</td>
<td>—</td>
<td>$I_H^h$</td>
</tr>
<tr>
<td>SMS</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

(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**
  - \( D_{\text{training}} = \{20, 21, \ldots, 60\} \setminus D_{\text{testing}} = \{30, 40, 50\} \)

- **Configuration setup**
  - all compared algorithms fine-tuned
  - tuning budget 25,000 algorithm runs
## Experimental results

<table>
<thead>
<tr>
<th>2-obj</th>
<th>3-obj</th>
<th>5-obj</th>
<th>2-obj</th>
<th>3-obj</th>
<th>5-obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>ΔR = 126</td>
<td>ΔR = 127</td>
<td>ΔR = 107</td>
<td>ΔR = 169</td>
<td>ΔR = 130</td>
<td>ΔR = 97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Auto(_D^2) (1339)</th>
<th>Auto(_D^3) (1500)</th>
<th>Auto(_D^5) (1002)</th>
<th>Auto(_W^2) (1692)</th>
<th>Auto(_W^3) (1375)</th>
<th>Auto(_W^5) (1170)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEA2(_D^2) (1562)</td>
<td>IBEA(_D^3) (1719)</td>
<td>SMS(_D^5) (1550)</td>
<td>SPEA2(_W^2) (2097)</td>
<td>SMS(_W^3) (1796)</td>
<td>SMS(_W^5) (1567)</td>
</tr>
<tr>
<td>IBEA(_D^2) (1940)</td>
<td>SMS(_D^3) (1918)</td>
<td>IBEA(_D^5) (1867)</td>
<td>NSGA-II(_W^2) (2542)</td>
<td>IBEA(_W^3) (1843)</td>
<td>IBEA(_W^5) (1746)</td>
</tr>
<tr>
<td>NSGA-II(_D^2) (2143)</td>
<td>HypE(_D^3) (2019)</td>
<td>SPEA2(_D^5) (2345)</td>
<td>SMS(_W^2) (2621)</td>
<td>SPEA2(_W^3) (2600)</td>
<td>SPEA2(_W^5) (2747)</td>
</tr>
<tr>
<td>HypE(_D^2) (2338)</td>
<td>SPEA2(_D^3) (2164)</td>
<td>NSGA-II(_D^5) (2346)</td>
<td>IBEA(_W^2) (2777)</td>
<td>NSGA-II(_W^3) (3315)</td>
<td>NSGA-II(_W^5) (3029)</td>
</tr>
<tr>
<td>SMS(_D^2) (2406)</td>
<td>NSGA-II(_D^3) (2528)</td>
<td>HypE(_D^5) (2674)</td>
<td>HypE(_W^2) (2851)</td>
<td>HypE(_W^3) (3431)</td>
<td>MOGA(_W^5) (4268)</td>
</tr>
<tr>
<td>MOGA(_D^2) (2970)</td>
<td>MOGA(_D^3) (2851)</td>
<td>MOGA(_D^5) (2915)</td>
<td>MOGA(_W^2) (4320)</td>
<td>MOGA(_W^3) (4540)</td>
<td>HypE(_W^5) (4373)</td>
</tr>
</tbody>
</table>
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

\[ s_0 := \text{initSolution} \]
\[ s^* := \text{ls}(s_0) \]

**repeat**
\[ s' := \text{perturb}(s^*, \text{history}) \]
\[ s^{*'} := \text{ls}(s') \]
\[ s^* := \text{accept}(s^*, s^{*'}, \text{history}) \]

**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

<table>
<thead>
<tr>
<th></th>
<th>perturb</th>
<th>ls</th>
<th>accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA</td>
<td>random move</td>
<td>∅</td>
<td>Metropolis</td>
</tr>
<tr>
<td>PII</td>
<td>random move</td>
<td>∅</td>
<td>Metropolis, fixed T</td>
</tr>
<tr>
<td>TS</td>
<td>∅</td>
<td>TS</td>
<td>∅</td>
</tr>
<tr>
<td>ILS</td>
<td>any</td>
<td>any</td>
<td>any</td>
</tr>
<tr>
<td>IG</td>
<td>destruct/construct</td>
<td>any</td>
<td>any</td>
</tr>
<tr>
<td>GRASP</td>
<td>rand. greedy sol.</td>
<td>any</td>
<td>∅</td>
</tr>
</tbody>
</table>
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}
<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

parameter instantiation →

performance measured and used to generate new parameter instantiations

→

problem-specific grammar

grammar2code

problem-independent grammar

→

source code

compiled and executed on test instances
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

**Results**

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>ARPD</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGrS</td>
<td>0.22</td>
</tr>
<tr>
<td>IGtb</td>
<td>0.24</td>
</tr>
<tr>
<td>irace1</td>
<td>0.26</td>
</tr>
<tr>
<td>irace2</td>
<td>0.28</td>
</tr>
<tr>
<td>irace3</td>
<td>0.30</td>
</tr>
<tr>
<td>irace4</td>
<td>0.30</td>
</tr>
</tbody>
</table>

**Chart**

95% confidence limits
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)