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
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

\[ g, \tau_{ij}, \eta_{ij} \]
Applying Ant Colony Optimization

Modeling side

- Problem
- Solution components
- Pheromone model

Algorithm side

- Probabilistic solution construction
- Local search
- Pheromone update
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*
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

Parameters Definition
- name
- type
- possible values

Configurator

Calls with candidate configuration

Returns solution cost

Best configuration to be used

Software to be tuned

Tackles

Set of instances
1
2
3
4
...

WCCI 2016, Vancouver, Canada
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-tableing 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 commercialized 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]
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*

\[ N(\mu_{d}^{z}, \sigma_{d}^{i}) \in [\underline{x}_d, \overline{x}_d] \]

\( \mu_{d}^{z} = \) value of parameter \( d \) in elite configuration \( z \)
\( \sigma_{d}^{i} = \) decreases with the number of iterations

Categorical parameter  \( X_d \in \{ x_1, x_2, \ldots, x_{n_d} \} \)
\( \Rightarrow \) *Discrete probability distribution*

\[ \text{Pr}^z \{ X_d = x_j \} = \begin{array}{c|c|c|c}
  x_1 & x_2 & \ldots & x_{n_d} \\
  0.1 & 0.3 & \ldots & 0.4 \\
\end{array} \]

- Updated by increasing probability of parameter value in elite configuration
- Other probabilities are reduced
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*
The irace package: usage

Training instances → Parameter space → Configuration scenario

calls with $\theta,i$ returns $c(\theta,i)$

irace

targetRunner
Example application of irace: ACOTSP


- 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

<table>
<thead>
<tr>
<th>#</th>
<th>name</th>
<th>switch</th>
<th>type</th>
<th>values</th>
<th>conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>algorithm</td>
<td>&quot;--&quot;</td>
<td>c</td>
<td>(as, mmas, eas, ras, acs)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>localsearch</td>
<td>&quot;--localsearch&quot;</td>
<td>c</td>
<td>(0, 1, 2, 3)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>alpha</td>
<td>&quot;--alpha&quot;</td>
<td>r</td>
<td>(0.00, 5.00)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>beta</td>
<td>&quot;--beta&quot;</td>
<td>r</td>
<td>(0.00, 10.00)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>rho</td>
<td>&quot;--rho&quot;</td>
<td>r</td>
<td>(0.01, 1.00)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ants</td>
<td>&quot;--ants&quot;</td>
<td>i</td>
<td>(5, 100)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>q0</td>
<td>&quot;--q0&quot;</td>
<td>r</td>
<td>(0.0, 1.0)</td>
<td>algorithm == &quot;acs&quot;</td>
</tr>
<tr>
<td>8</td>
<td>rasrank</td>
<td>&quot;--rasranks&quot;</td>
<td>i</td>
<td>(1, 100)</td>
<td>algorithm == &quot;ras&quot;</td>
</tr>
<tr>
<td>9</td>
<td>elitistants</td>
<td>&quot;--elitistants&quot;</td>
<td>i</td>
<td>(1, 750)</td>
<td>algorithm == &quot;eas&quot;</td>
</tr>
<tr>
<td>10</td>
<td>nnls</td>
<td>&quot;--nnls&quot;</td>
<td>i</td>
<td>(5, 50)</td>
<td>localsearch %in% c(1,2,3)</td>
</tr>
<tr>
<td>11</td>
<td>dlb</td>
<td>&quot;--dlb&quot;</td>
<td>c</td>
<td>(0, 1)</td>
<td>localsearch %in% c(1,2,3)</td>
</tr>
</tbody>
</table>
Example application of irace: ACOTSP

```bash
#!/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
  
  $ cat forbidden.txt
  
  (alpha == 0.0) & (beta == 0.0)
  ```
Other configurators: ParamILS, SMAC
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 $\Theta_0$
- 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 $\Theta_p$
  - evaluate configurations in $\Theta_p$
  - $\Theta_0 := \Theta_0 \cup \Theta_p$
  - update $\theta^* \in \Theta_0$
- **end**

**output:** $\theta^*$
SMAC extends surrogate model-based configuration to complex algorithm configuration tasks and across multiple instances.

Main design decisions

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

<table>
<thead>
<tr>
<th>Benchmark set</th>
<th>Default</th>
<th>Configured</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regions200</td>
<td>72</td>
<td>10.5 (11.4 ± 0.9)</td>
<td>6.8</td>
</tr>
<tr>
<td>Conic.SCH</td>
<td>5.37</td>
<td>2.14 (2.4 ± 0.29)</td>
<td>2.51</td>
</tr>
<tr>
<td>CLS</td>
<td>712</td>
<td>23.4 (327 ± 860)</td>
<td>30.43</td>
</tr>
<tr>
<td>MIK</td>
<td>64.8</td>
<td>1.19 (301 ± 948)</td>
<td>54.54</td>
</tr>
<tr>
<td>QP</td>
<td>969</td>
<td>525 (827 ± 306)</td>
<td>1.85</td>
</tr>
</tbody>
</table>

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

**Average Errors–30D–100runs**
- Win 8
- Lose 4
- Draw 13

**Average Errors–50D–100runs**
- Win 13 +
- Lose 4
- Draw 8
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 grammar 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

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

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

<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

\[
<\text{algorithm}> ::= <\text{initialization}> <\text{ils}>
\]
\[
<\text{initialization}> ::= \text{random} \mid <\text{pbs}\_\text{initialization}>
\]
\[
<\text{ils}> ::= \text{ILS}(<\text{perturb}>, <\text{ls}>, <\text{accept}>, <\text{stop}>)
\]
\[
<\text{perturb}> ::= \text{none} \mid <\text{initialization}> \mid <\text{pbs}\_\text{perturb}>
\]
\[
<\text{ls}> ::= <\text{ils} > \mid <\text{descent} > \mid <\text{sa} > \mid <\text{rii} > \mid <\text{pii} > \mid <\text{vns} > \mid <\text{ig} > \mid <\text{pbs}\_\text{ls}>
\]
\[
<\text{accept}> ::= \text{alwaysAccept} \mid \text{improvingAccept} <\text{comparator}>
\]
\[
\mid \text{prob}(<\text{value}\_\text{prob}\_\text{accept}>) \mid \text{probRandom} \mid <\text{metropolis}>
\]
\[
\mid \text{threshold}(<\text{value}\_\text{threshold}\_\text{accept}>) \mid <\text{pbs}\_\text{accept}>
\]
\[
<\text{descent}> ::= \text{bestDescent}(<\text{comparator}>, <\text{stop}>)
\]
\[
\mid \text{firstImprDescent}(<\text{comparator}>, <\text{stop}>)
\]
\[
<\text{sa}> ::= \text{ILS}(<\text{pbs}\_\text{move}>, \text{no}\_\text{ls}, <\text{metropolis}>, <\text{stop}>)
\]
\[
<\text{rii}> ::= \text{ILS}(<\text{pbs}\_\text{move}>, \text{no}\_\text{ls}, \text{probRandom}, <\text{stop}>)
\]
\[
<\text{pii}> ::= \text{ILS}(<\text{pbs}\_\text{move}>, \text{no}\_\text{ls}, \text{prob}(<\text{value}\_\text{prob}\_\text{accept}>) , <\text{stop}>)
\]
\[
<\text{vns}> ::= \text{ILS}(<\text{pbs}\_\text{variable}\_\text{move}>, \text{firstImprDescent}(\text{improvingStrictly}), \text{improvingAccept}(\text{improvingStrictly}), <\text{stop}>)
\]
\[
<\text{ig}> ::= \text{ILS}(<\text{deconst}\_\text{construct}\_\text{perturb}>, <\text{ls}>, <\text{accept}>, <\text{stop}>)
\]
\[
<\text{comparator}> ::= \text{improvingStrictly}\mid \text{improving}
\]
\[
<\text{value}\_\text{prob}\_\text{accept}>::= [0, 1]
\]
\[
<\text{value}\_\text{threshold}\_\text{accept}>::= [0, 1]
\]
\[
<\text{metropolis}> ::= \text{metropolisAccept}(<\text{init}\_\text{temperature}>, <\text{final}\_\text{temperature}>,
\]
\[
\text{decreasing}\_\text{temperature}\_\text{ratio}>, <\text{span}>)
\]
\[
<\text{init}\_\text{temperature}>::= \{1, 2, \ldots, 10000\}
\]
\[
<\text{final}\_\text{temperature}>::= \{1, 2, \ldots, 100\}
\]
\[
<\text{decreasing}\_\text{temperature}\_\text{ratio}>::= [0, 1]
\]
\[
<\text{span}>::= \{1, 2, \ldots, 10000\}
\]
<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}
<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
- Compiled and executed on test instances
- Source code
- Grammar2Code
- Problem-specific grammar
- Problem-independent grammar
Flow-shop problem with weighted tardiness

- Automatic configuration:
  - 1, 2 or 3 levels of recursion ($r$)
  - 80, 127, and 174 parameters, respectively
  - budget: $r \times 10000$ 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 $\leadsto$ 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
$\text{irace + hypervolume = automatic configuration of multi-objective solvers!}$
Automatic configuration multi-objective ACO

![Graph showing performance of different ACO algorithms across three datasets (euclidAB100.tsp, euclidAB300.tsp, euclidAB500.tsp) with varying parameters (0.5 to 1.0). 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>Euclidean AB 100.tsp</th>
<th>Euclidean AB 300.tsp</th>
<th>Euclidean AB 500.tsp</th>
</tr>
</thead>
<tbody>
<tr>
<td>BicriterionAnt (3 col)</td>
<td>BicriterionAnt–aco (1)</td>
<td>BicriterionAnt–aco (5)</td>
</tr>
<tr>
<td>MOACO (5)</td>
<td>MOACO–aco (1)</td>
<td>MOACO–full (5)</td>
</tr>
<tr>
<td>MOACO–aco (2)</td>
<td>MOACO–aco (2)</td>
<td>MOACO–full (4)</td>
</tr>
<tr>
<td>MOACO–aco (3)</td>
<td>MOACO–aco (3)</td>
<td>MOACO–full (3)</td>
</tr>
<tr>
<td>MOACO–aco (4)</td>
<td>MOACO–aco (4)</td>
<td>MOACO–full (2)</td>
</tr>
<tr>
<td>MOACO–aco (5)</td>
<td>MOACO–aco (5)</td>
<td>MOACO–full (1)</td>
</tr>
</tbody>
</table>

0.85 0.90 0.95 1.00 1.05 1.10

● ● ● ● ●

0.85 0.90 0.95 1.00 1.05 1.10

● ● ● ● ●

0.85 0.90 0.95 1.00 1.05 1.10

● ● ● ● ●

EuclidAB100.tsp

EuclidAB300.tsp

EuclidAB500.tsp

WCCI 2016, Vancouver, Canada
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

1: \( \text{pop} \leftarrow \text{Initialization}() \)
2: \( \text{if } \text{type}(\text{pop}_{\text{ext}}) \neq \text{none} \)
3: \( \text{pop}_{\text{ext}} \leftarrow \text{pop} \)
4: \( \text{repeat} \)
5: \( \text{pool} \leftarrow \text{BuildMatingPool}(\text{pop}) \)
6: \( \text{pop}_{\text{new}} \leftarrow \text{Variation}(\text{pool}) \)
7: \( \text{pop}_{\text{new}} \leftarrow \text{Evaluation}(\text{pop}_{\text{new}}) \)
8: \( \text{pop} \leftarrow \text{Replacement}(\text{pop}, \text{pop}_{\text{new}}) \)
9: \( \text{if } \text{type}(\text{pop}_{\text{ext}}) = \text{bounded} \text{ then} \)
10: \( \text{pop}_{\text{ext}} \leftarrow \text{Replacement}_{\text{Ext}}(\text{pop}_{\text{ext}}, \text{pop}_{\text{new}}) \)
11: \( \text{else if } \text{type}(\text{pop}_{\text{ext}}) = \text{unbounded} \text{ then} \)
12: \( \text{pop}_{\text{ext}} \leftarrow \text{pop}_{\text{ext}} \cup \text{pop} \)
13: \( \text{until } \text{termination criteria met} \)
14: \( \text{if } \text{type}(\text{pop}_{\text{ext}}) = \text{none} \)
15: \( \text{return } \text{pop} \)
16: \( \text{else} \)
17: \( \text{return } \text{pop}_{\text{ext}} \)
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(<em>{Ext}), Removal(</em>{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; 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
  - $D_{training} = \{20, 21, \ldots, 60\} \backslash D_{testing} = \{30, 40, 50\}$

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

<table>
<thead>
<tr>
<th></th>
<th>DTLZ</th>
<th></th>
<th>WFG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2-obj</td>
<td>3-obj</td>
<td>5-obj</td>
</tr>
<tr>
<td>( \Delta R )</td>
<td>126</td>
<td>127</td>
<td>107</td>
</tr>
<tr>
<td>Auto(_D2)</td>
<td>(1339)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auto(_D3)</td>
<td>(1500)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auto(_D5)</td>
<td>(1002)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPEA2(_D2)</td>
<td>(1562)</td>
<td>IBEA(_D3)</td>
<td>SMS(_D5)</td>
</tr>
<tr>
<td></td>
<td>(1719)</td>
<td>(1550)</td>
<td></td>
</tr>
<tr>
<td>IBEA(_D2)</td>
<td>(1940)</td>
<td>SMS(_D3)</td>
<td>IBEA(_D5)</td>
</tr>
<tr>
<td></td>
<td>(1918)</td>
<td>(1867)</td>
<td></td>
</tr>
<tr>
<td>NSGA-II(_D2)</td>
<td>(2143)</td>
<td>HypE(_D3)</td>
<td>SPEA2(_D5)</td>
</tr>
<tr>
<td></td>
<td>(2019)</td>
<td>(2345)</td>
<td></td>
</tr>
<tr>
<td>HypE(_D2)</td>
<td>(2338)</td>
<td>SPEA2(_D3)</td>
<td>NSGA-II(_D5)</td>
</tr>
<tr>
<td></td>
<td>(2164)</td>
<td>(2346)</td>
<td></td>
</tr>
<tr>
<td>SMS(_D2)</td>
<td>(2406)</td>
<td>NSGA-II(_D3)</td>
<td>HypE(_D5)</td>
</tr>
<tr>
<td></td>
<td>(2528)</td>
<td>(2674)</td>
<td></td>
</tr>
<tr>
<td>MOGA(_D2)</td>
<td>(2970)</td>
<td>MOGA(_D3)</td>
<td>MOGA(_D5)</td>
</tr>
<tr>
<td></td>
<td>(2851)</td>
<td>(2915)</td>
<td></td>
</tr>
</tbody>
</table>
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. ablation)
  - 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 aim to produce as high quality results as possible, independent of the computation time allowed.

Anytime Algorithms [Zilberstein, 1996]
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 *q₀* 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

![Graph showing relative deviation from best-known values for different settings.

- Default: 1.1599
- Auto var ants: 1.1865
- Auto var beta: 1.182
- Auto var rho: 1.1813
- Auto var q0: 1.1935
- Auto var ALL: 1.2012

The graph compares the relative deviation from the best-known values across different settings of parameters for a given time in seconds.]
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 \textit{much easier}

\textit{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

\textbf{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

WCCI 2016, Vancouver, Canada
Few other topics
What if my problem instances are too difficult/large?

- Cloud computing / Large computing clusters

  Tune on small instances,
  then extend to increasingly larger ones


  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

![http://www.aclib.net/](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)