Automatic (Offline) Configuration of Algorithms

Thomas Stützle
stuetzle@ulb.ac.be
http://iridia.ulb.ac.be/~stuetzle

Manuel López-Ibáñez
manuel.lopez-ibanez@ulb.ac.be
http://iridia.ulb.ac.be/~manuel

IRIDIA, CoDE, Université Libre de Bruxelles (ULB),
Brussels, Belgium

GECCO'15 Companion, July 11-15, 2015, Madrid, Spain
ACM 978-1-4503-3488-4/15/07.
http://dx.doi.org/10.1145/2739482.2756581

Part I

Automatic Algorithm Configuration (Overview)

Solving complex optimization problems

The algorithmic solution of hard optimization problems is one of the CS/OR success stories!

- Exact (systematic search) algorithms
  - branch&bound, branch&cut, constraint programming, ...
  - guarantees of optimality but often time/memory consuming
  - powerful general-purpose software available
- Approximation algorithms
  - heuristics, local search, metaheuristics, hyperheuristics ...
  - rarely provable guarantees but often fast and accurate
  - typically special-purpose software

Design choices and parameters everywhere

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

- Exact solvers
  - Design choices: alternative models, pre-processing, variable selection, value selection, branching rules ...
    + numerical parameters
  - SCIP solver: more than 200 parameters that influence search
- (Meta)-heuristic solvers
  - Design choices: solution representation, operators, neighborhoods, pre-processing, strategies, ... + numerical parameters
  - Multi-objective ACO algorithms with 22 parameters (see part 2)

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, ...
- Parameters may be *conditional* to specific values of other parameters

Configuring algorithms involves setting categorical, ordinal and numerical parameters

Mario's Pizza Delivery Problem (Birattari, 2004; Birattari et al., 2002)

Mario collects phone orders for 30 minutes. He wants to schedule deliveries to get back to the pizzeria as fast as possible.

- Scheduling deliveries is an *optimization problem*
- A different *problem instance* arises every 30 minutes
- Limited amount of time for scheduling, say one minute
- Limited amount of time to implement an optimization algorithm, say one week

Manual design and tuning

Human expert + trial-and-error/statistics

Towards more systematic approaches

Traditional approaches

- Trial–and–error design guided by expertise/intuition
  - ✘ prone to over-generalizations, limited exploration of design alternatives, human biases
- Guided by theoretical studies
  - ✘ often based on over-simplifications, specific assumptions, few parameters

Can we make this approach more principled and automatic?

Towards automatic (offline) algorithm configuration

Automatic algorithm configuration

- apply powerful search techniques to design algorithms
- use computation power to explore algorithm design spaces
- free human creativity for higher level tasks

Offline configuration and online parameter control

Offline tuning / Algorithm configuration

- Learn best parameters before *solving* an instance
- Configuration done on training instances
- Performance measured over test ($\neq$ training) instances

Online tuning / Parameter control / Reactive search

- Learn parameters *while* solving an instance
- No training phase
- Limited to very few crucial parameters
- Often static parameter tuning done online (Pellegrini et al., 2014)

The algorithm configuration problem (Birattari, 2009)

A configuration instance (scenario) is defined by a tuple

$$\langle \Theta, \mathcal{I}, \mathcal{C}, c_\theta \rangle$$

Θ : set of potential algorithm configurations (possibly infinite)

$\mathcal{I}$: set of instances (possibly infinite), from which instances are sampled with certain probability

$\mathcal{C} \colon \Theta \times \mathcal{I} \to \mathbb{R}$: cost measure, where:

$\mathcal{C}(\theta, i)$: cost of configuration $\theta \in \Theta$ on instance $i \in \mathcal{I}$

$c(\theta, i)$: cost after running *one time* configuration θ on instance i

c_θ : function of the cost $\mathcal{C}$ of a configuration θ with respect to the distribution of the random variable $\mathcal{I}$

The algorithm configuration problem (Birattari, 2009)

A configuration instance (scenario) is defined by a tuple

$$\langle \Theta, \mathcal{I}, \mathcal{C}, c_\theta \rangle$$

parameter space (Θ), problem instances ($\mathcal{I}$),
optimization objective ($\mathcal{C}$), tuning objective (c_θ)

Goal: find the best configuration θ^* such that:

$$\theta^* = \arg\min_{\theta \in \Theta} c_\theta$$

- c_θ could be $E_{\mathcal{C},\mathcal{I}}[\theta]$ *or* sum of ranks *or* ...
- Analytical solution not possible $\Rightarrow$ estimate c_θ
  - by sampling $I_{\text{train}} \sim \mathcal{I}$ (training instances)
  - by sampling $\mathcal{C}(\theta, i), i \in I_{\text{train}}$ (running experiments)

AC is a stochastic optimization problem

Decision variables

- discrete (categorical, ordinal, integer) and continuous

Stochasticity

- of the target algorithm
- of the problem instances

Typical tuning goals

- maximize solution quality within given time
- minimize run-time to decision / optimal solution

AC requires specialized methods

Methods for Automatic Algorithm Configuration

experimental design, ANOVA
CALIBRA (Adenso-Díaz & Laguna, 2006)
others (Coy et al., 2001; Ridge & Kudenko, 2007; Ruiz & Maroto, 2005)

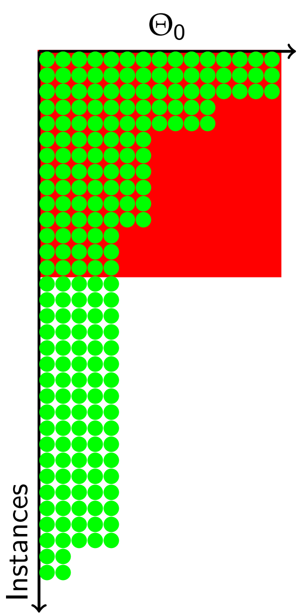
numerical optimization
MADS (Audet & Orban, 2006), CMA-ES, BOBYQA (Yuan et al., 2012)

heuristic optimization
meta-GA (Grefenstette, 1986), ParamILS (Hutter et al., 2007b, 2009), gender-based GA (Ansótegui et al., 2009), linear GP (Oltean, 2005), REVAC(++) (Nannen & Eiben, 2006; Smit & Eiben, 2009, 2010) . . .

model-based
SPO (Bartz-Beielstein et al., 2005, 2010), SMAC (Hutter et al., 2011)

sequential statistical testing
F-race, iterated F-race (Balaprakash et al., 2007; Birattari et al., 2002)
irace (López-Ibáñez et al., 2011)

The racing approach (Birattari et al., 2002)



- 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 racing approach (Birattari et al., 2002)

How to discard?

Statistical testing!

- *F-Race:* Friedman two-way analysis of variance by ranks + Friedman post-hoc test (Conover, 1999)
- Alternative: paired t-test with/without p-value correction (against the best)

Some (early) applications of F-race

Vehicle routing and scheduling problem (Becker et al., 2005)

- first industrial application
- improved commercialized algorithm

International time-tabling competition (Chiarandini et al., 2006)

- winning algorithm configured by F-race
- interactive injection of new configurations

F-race in stochastic optimization (Birattari et al., 2006)

- evaluate "neighbors" using F-race
  (solution cost is a random variable)
- very good performance if variance of solution cost is high

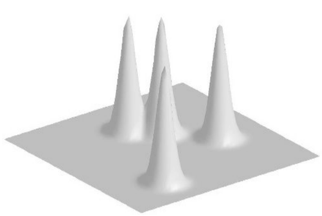
Sampling configuration

F-race is a method for the *selection of the best* among a given set of algorithm configurations $\Theta_0 \subset \Theta$

How to sample algorithm configurations?

- Full factorial
- Random sampling
- Iterative refinement of a sampling model
  $\Rightarrow$ *Iterated F-Race (I/F-Race)* (Balaprakash et al., 2007)

Iterative race: an illustration

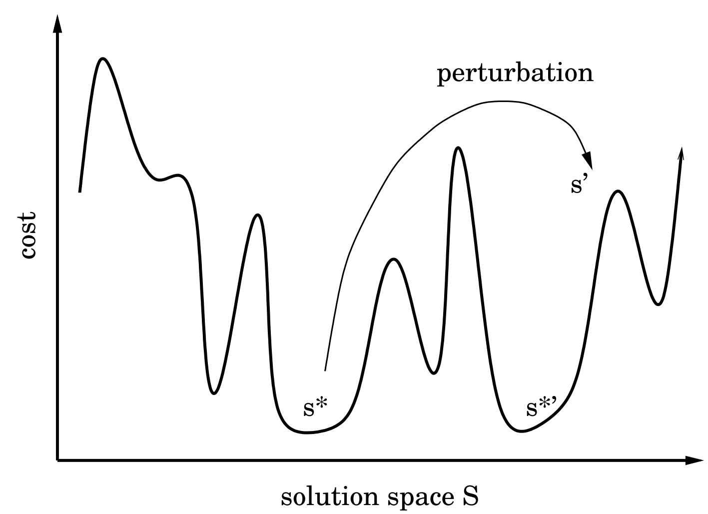


1: sample configurations from initial distribution
2: **while** not terminate() **do**
3: apply race
4: modify the distribution
5: sample configurations with selection probability

☞ more details, see part 2 of the tutorial

ParamILS Framework (Hutter et al., 2007b, 2009)

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, where exactly one parameter changes a value at a time
- neighborhood is searched 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

Applications of ParamILS

- SAT-based verification (Hutter et al., 2007a)
  - SPEAR solver with 26 parameters
    $\Rightarrow$ speed-ups of up to 500 over default configuration
- Configuration of commercial MIP solvers (Hutter et al., 2010)
  - CPLEX (63 parameters), Gurobi (25 parameters) and lpsolve (47 parameters) for various instance distributions of MIP encoded optimization problems
  - speed-ups ranged between a factor of 1 (none) to 153

Gender-based genetic algorithm (Ansótegui et al., 2009)

Parameter encoding

- variable structure that is inspired by *And/Or* trees
- *And* nodes separate variables that can be optimized independently
- instrumental for defining the crossover operator

Main details

- crossover between configurations from different sub-populations
- parallel evaluation of candidates supports early termination of poor performing candidates (inspired by racing / capping)
- designed for minimization of computation time

Promising initial results

Relevance Estimation and Value Calibration (REVAC)

(Nannen & Eiben, 2006; Smit & Eiben, 2009, 2010)

REVAC is an EDA for tuning *numerical* parameters

Main details

- variables are treated as independent
- multi-parent crossover of best parents to produce one child per iteration
- relevance of parameters is estimated by Shannon entropy

Extensions

- REVAC++ uses racing and sharpening (Smit & Eiben, 2009)
- training on more than one instance (Smit & Eiben, 2010)

Numerical optimization techniques

MADS / OPAL

- Mesh-adaptive direct search applied to parameter tuning of other direct-search methods (Audet & Orban, 2006)
- later extension to OPAL (*OP*timization of *AL*gorithms) framework (Audet et al., 2010)
- Limited experiments

Other continuous optimizers (Yuan et al., 2012, 2013)

- study of CMAES, BOBYQA, MADS, and irace for tuning continuous and quasi-continuous parameters
- BOBYQA best for few parameters; CMAES best for many
- post-selection mechanism appears promising

Model-based Approaches (SPOT, SMAC)

Idea: Use surrogate models to predict performance

Algorithmic scheme

1: generate and evaluate initial set of configurations Θ_0
2: choose best-so-far configuration $\theta^* \in \Theta_0$
3: **while** tuning budget available **do**
4: learn surrogate model $\mathcal{M}\colon \Theta \mapsto R$
5: use model $\mathcal{M}$ to generate promising configurations Θ_p
6: evaluate configurations in Θ_p
7: $\Theta_0 := \Theta_0 \cup \Theta_p$
8: update $\theta^* \in \Theta_0$
9: **output:** θ^*

Sequential parameter optimization (SPO) toolbox

(Bartz-Beielstein et al., 2005, 2010)

Main design decisions

- Gaussian stochastic processes for $\mathcal{M}$ (in most variants)
- Expected improv. criterion (EIC) $\Rightarrow$ promising configurations
- Intensification mechanism $\Rightarrow$ increase num. of evals. of θ^*

Practicalities

- SPO is implemented in the comprehensive SPOT R package
- Most applications to numerical parameters on one instance
- SPOT includes analysis and visualization tools

Sequential model-based algorithm configuration (SMAC)

(Hutter et al., 2011)

SMAC extends surrogate model-based configuration to complex algorithm configuration tasks and across multiple instances

Main design decisions

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

Which method would you use?

- Only numerical parameters:
  - Homogeneous instances $\Rightarrow$ numerical optimizers, e.g., **BOBYQA** (few parameters), **CMA-ES** (many)
  - Expensive homogeneous instances $\Rightarrow$ **SPO**
  - Heterogeneous instances $\Rightarrow$ **numerical optimizers + (racing or post-selection)**
- Categorical and numerical parameters
  - Tuning goal is time $\Rightarrow$ **SMAC**
  - Tuning goal is quality $\Rightarrow$ **IRACE**

Disclaimer: *This is a personal opinion based on our own experience*

AClib: A Benchmark Library for Algorithm Configuration

F. Hutter, M. López-Ibáñez, C. Fawcett, M. Lindauer, H. H. Hoos, K. Leyton-Brown and T. Stützle. **AClib: a Benchmark Library for Algorithm Configuration**, Learning and Intelligent Optimization Conference (LION 8), 2014.

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 !

Why automatic algorithm configuration?

- improvement over manual, ad-hoc methods for tuning
- reduction of development time and human intervention
- increased number of potential designs
- empirical studies, comparisons of algorithms
- support for end-users of algorithms

. . . and it has become feasible due to increase in computational power!

Scaling to expensive instances

What if my problem instances are too difficult/large?

- Cloud computing / Large computing clusters
- J. Styles and H. H. Hoos. **Ordered racing protocols for automatically configuring algorithms for scaling performance.** GECCO, 2013

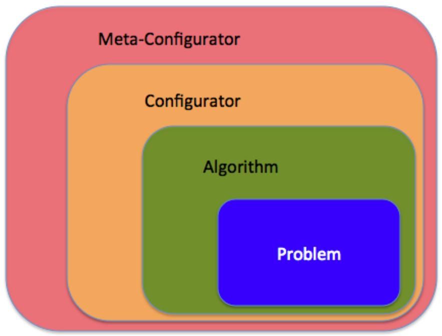
  Tune on easy instances,
  then ordered F-race on increasingly difficult ones

- F. Mascia, M. Birattari, and T. Stützle. **Tuning algorithms for tackling large instances: An experimental protocol.** Learning and Intelligent Optimization, LION 7, 2013.

  Tune on easy instances,
  then scale parameter values to difficult ones

Configuring configurators

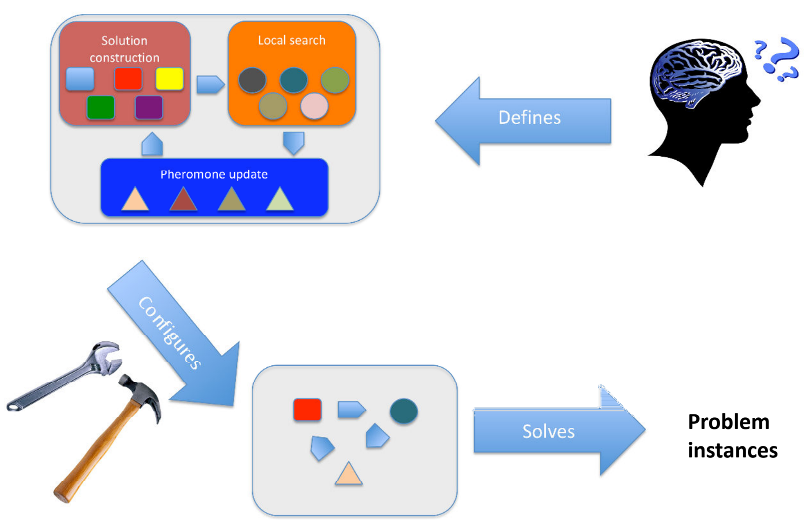
What about configuring automatically the configurator?
. . . and configuring the configurator of the configurator?



✔ it can be done (Hutter et al., 2009) but . . .

✘ it is costly and iterating further leads to diminishing returns

Towards a paradigm shift in algorithm design



Part II

Iterated Racing (irace)

What is Iterated Racing and irace?

Iterated Racing (irace)

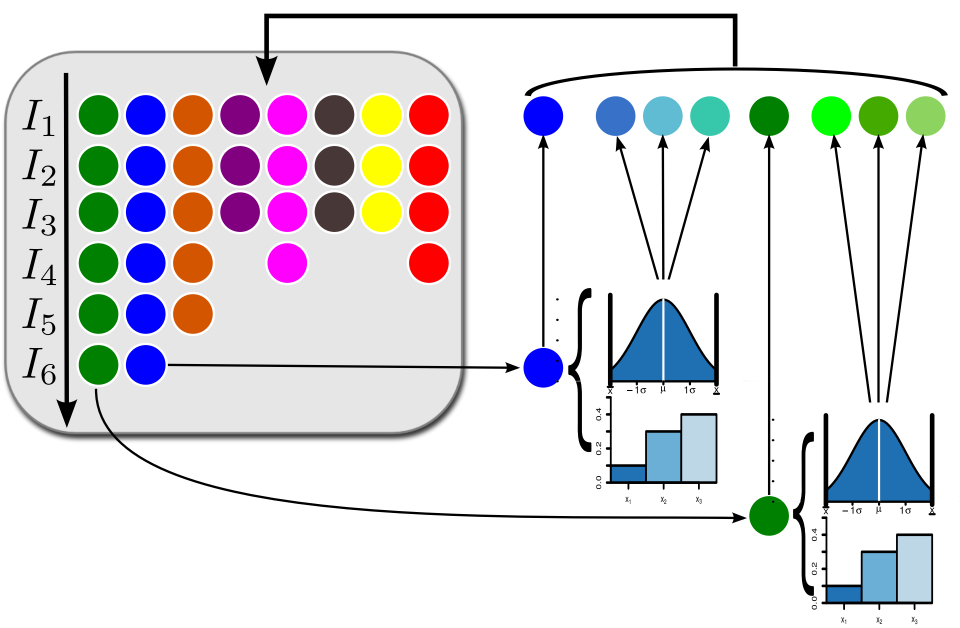
1. A variant of I/F-Race with several extensions
   - I/F-Race proposed by Balaprakash, Birattari, and Stützle (2007)
   - Refined by Birattari, Yuan, Balaprakash, and Stützle (2010)
   - Further refined and extended by López-Ibáñez, Dubois-Lacoste, Stützle, and Birattari (2011)
2. A software package implementing the variant proposed by López-Ibáñez, Dubois-Lacoste, Stützle, and Birattari (2011)

Iterated Racing

Iterated Racing $\supseteq$ I/F-Race

1. **Sampling** new configurations according to a probability distribution
2. **Selecting** the best configurations from the newly sampled ones by means of racing
3. **Updating** the probability distribution in order to bias the sampling towards the best configurations

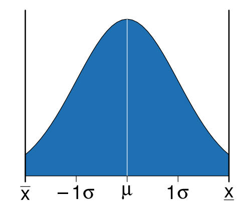
Iterated Racing



Iterated Racing: Sampling distributions

Numerical parameter $X_d \in [\underline{x}_d, \overline{x}_d]$

$\Rightarrow$ *Truncated normal distribution*



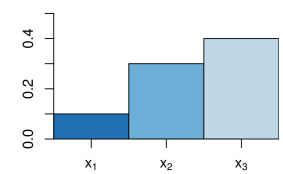
$$\mathcal{N}(\mu_d^z, \sigma_d^i) \in [\underline{x}_d, \overline{x}_d]$$

μ_d^z = value of parameter d in elite configuration z

σ_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*



$\Pr^z\{X_d = x_j\} =$

x_1	x_2	...	x_{n_d}
0.1	0.3	...	0.4

- Updated by increasing probability of parameter value in elite configuration
- Other probabilities are reduced

Iterated Racing: Soft-restart

✘ irace may converge too fast
$\Rightarrow$ the same configurations are sampled again and again
✔ Soft-restart !

1. *Compute distance* between sampled candidate configurations
2. If distance is zero, *soft-restart* the sampling distribution of the parents
   Numerical parameters : σ_d^i is "brought back" to its value at two iterations earlier, approx. σ_d^{i-2}

   Categorical parameters : "smoothing" of probabilities, increase low values, decrease high values.
3. *Resample*

Iterated Racing: Other features

1. Initial configurations
   - Seed irace with the default configuration or configurations known to be good for other problems
2. Parallel evaluation
   - Configurations within a race can be evaluated in parallel using MPI, multiple cores, Grid Engine / qsub clusters
3. Forbidden configurations *(new in 1.05)*
   - `popsize < 5 & LS == "SA"`
4. Recovery file *(new in 1.05)*
   - allows resuming a previous irace run

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

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

- R package available at CRAN:

  http://cran.r-project.org/package=irace

```
R> install.packages("irace")
```

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

- Use it from inside R . . .

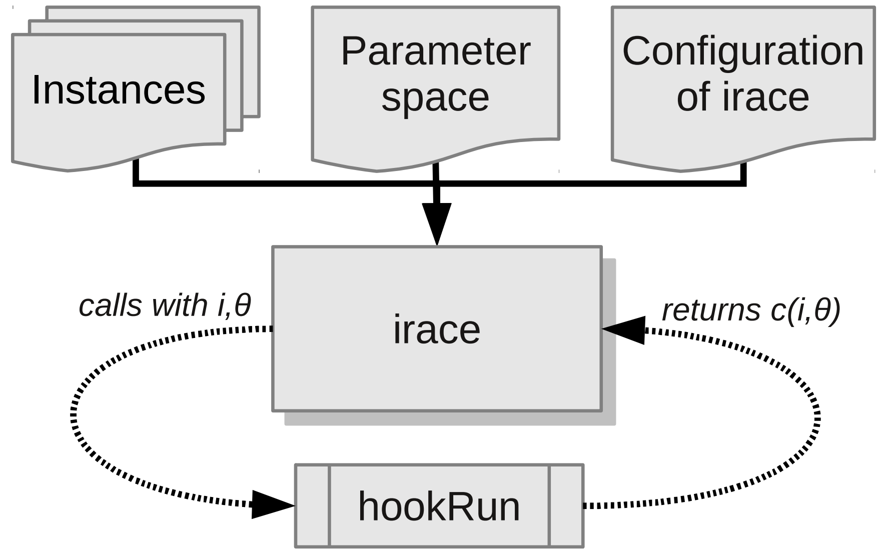
```
R> result <- irace(tunerConfig = list(maxExperiments = 1000),
                   parameters = parameters)
```

- . . . or through command-line: (See `irace --help`)

```
irace --max-experiments 1000 --param-file parameters.txt
```

✔ No knowledge of R needed

The irace Package



The irace Package: Instances

- TSP instances

```
$ dir Instances/
3000-01.tsp 3000-02.tsp 3000-03.tsp ...
```

- Continuous functions

```
$ cat instances.txt
function=1 dimension=100
function=2 dimension=100
...
```

- Parameters for an instance generator

```
$ cat instances.txt
I1 --size 100 --num-clusters 10 --sym yes --seed 1
I2 --size 100 --num-clusters 5 --sym no --seed 1
...
```

- Script / R function that generates instances
  ☞ if you need this, tell us!

The irace Package: Parameter space

- Categorical (`c`), ordinal (`o`), integer (`i`) and real (`r`)
- Subordinate parameters (`| condition`)

$ `cat parameters.txt`

```
# Name        Label/switch        Type   Domain            Condition
LS            "--localsearch "     c     {SA, TS, II}
rate          "--rate="            o     {low, med, high}
population    "--pop "             i     (1, 100)
temp          "--temp "            r     (0.5, 1)          | LS == "SA"
```

- For real parameters, number of decimal places is controlled by option *digits* (`--digits`)

The irace Package: Options

- *digits*: number of decimal places to be considered for the real parameters (default: 4)
- *maxExperiments*: maximum number of runs of the algorithm being tuned (tuning budget)
- *testType*: either F-test or t-test
- *firstTest*: specifies how many instances are seen before the first test is performed (default: 5)
- *eachTest*: specifies how many instances are seen between tests (default: 1)

The irace Package: hook-run

- A script/program that calls the software to be tuned:

```
./hook-run instance candidate-number candidate-parameters ...
```

- An R function:

```
hook.run <- function(instance, candidate, extra.params = NULL,
                     config = list())
{
  ...
}
```

Flexibility: If there is something you cannot tune, let us know!

Example #1

ACOTSP

Example: ACOTSP

- ACOTSP: ant colony optimization algorithms for the TSP
- Command-line program:

```
./acotsp -i instance -t 300 --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: ACOTSP

$ cat parameters.txt

```
# name       switch            type  values      conditions
algorithm    "--"              c (as,mmas,eas,ras,acs)
localsearch "--localsearch " c (0, 1, 2, 3)
alpha        "--alpha "        r (0.00, 5.00)
beta         "--beta "         r (0.00, 10.00)
rho          "--rho  "         r (0.01, 1.00)
ants         "--ants "         i (5, 100)
q0           "--q0 "           r (0.0, 1.0) | algorithm == "acs"
rasrank      "--rasranks "     i (1, 100)   | algorithm == "ras"
elitistants "--elitistants " i (1, 750)   | algorithm == "eas"
nnls         "--nnls "         i (5, 50)    | localsearch %in% c(1,2,3)
dlb          "--dlb "          c (0, 1)     | localsearch %in% c(1,2,3)
```


Example: ACOTSP

$ cat hook-run

```
#!/bin/bash
INSTANCE=$1
CANDIDATENUM=$2
CAND_PARAMS=$*
STDOUT="c${CANDIDATENUM}.stdout"
FIXED_PARAMS=" --time 20 --tries 1 --quiet "
acotsp $FIXED_PARAMS -i $INSTANCE $CAND_PARAMS 1> $STDOUT
COST=$(grep -oE 'Best [-+0-9.e]+' $STDOUT |cut -d' ' -f2)
if ! [[ "${COST}" =~ ^[-+0-9.e]+$ ]] ; then
   error "${STDOUT}: Output is not a number"
fi
echo "${COST}"
exit 0
```


Example: ACOTSP

$ cat tune-conf

```
execDir <- "./acotsp-arena"
instanceDir <- "./Instances"
maxExperiments <- 1000
digits <- 2
```

✔ Good to go:

```
$ mkdir acotsp-arena
$ irace
```


Example #2

A more complex example:
MOACO framework

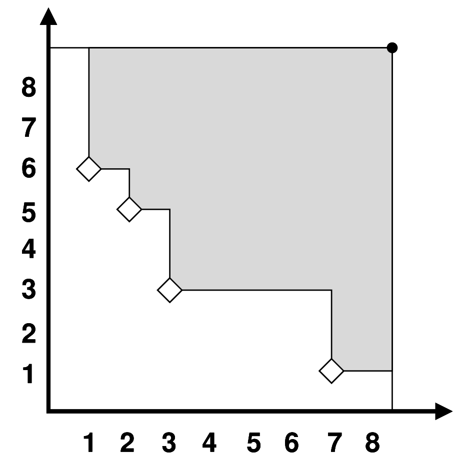
A more complex example: MOACO framework

Manuel López-Ibáñez and Thomas Stützle. **The automatic design of multi-objective ant colony optimization algorithms**. *IEEE Transactions on Evolutionary Computation*, 2012.

- A flexible framework of multi-objective ant colony optimization algorithms
- Parameters controlling multi-objective algorithmic design
- Parameters controlling underlying ACO settings
- Instantiates 9 MOACO algorithms from the literature
- Hundreds of potential ~~papers~~ algorithm designs

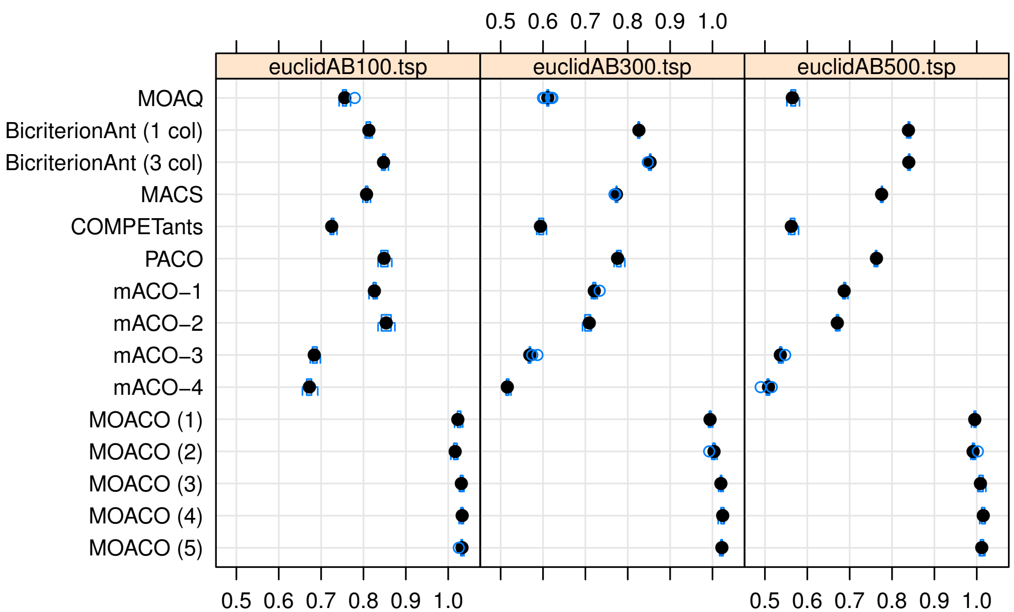
A more complex example: MOACO framework

✘ Multi-objective! Output is an approximation to the Pareto front!



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

Results: Multi-objective components



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 top-down frameworks for MIP: CPLEX, SCIP
  - Single-objective top-down framework for SAT, SATzilla *(Xu, Hutter, Hoos, and Leyton-Brown, 2008)*
  - Multi-objective automatic configuration with SPO *(Wessing, Beume, Rudolph, and Naujoks, 2010)*
  - Multi-objective framework for PFSP, TP+PLS *(Dubois-Lacoste, López-Ibáñez, and Stützle, 2011)*

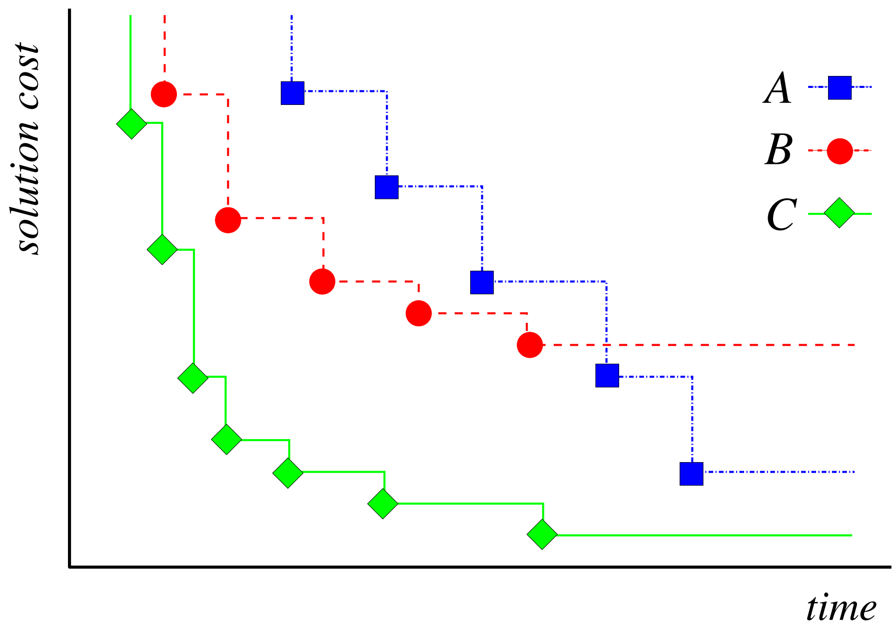
Automatically Improving the Anytime Behavior of Optimization Algorithms with irace

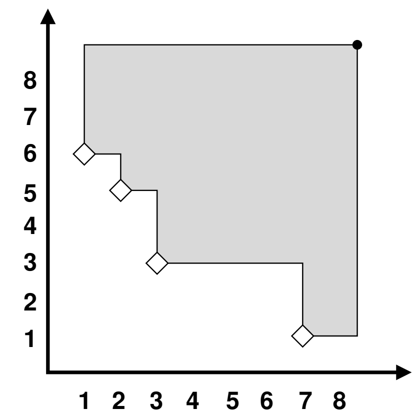
Anytime Algorithm (Dean & Boddy, 1988)

- May be interrupted at any moment and returns a solution
- Keeps improving its solution until interrupted
- Eventually finds the optimal solution

Good Anytime Behavior (Zilberstein, 1996)

Algorithms with good *"anytime" behavior* produce as high quality result as possible at any moment of their execution.





Hypervolume measure ≈ Anytime behaviour

Manuel López-Ibáñez and Thomas Stützle. **Automatically improving the anytime behaviour of optimisation algorithms**. *European Journal of Operational Research*, 2014.

Automatically Improving the Anytime Behavior

Scenario #1

Online parameter adaptation to make an algorithm more robust to different termination criteria

- ✘ Which parameters to adapt? How? ⇒ More parameters!
- ✔ Use irace (offline) to select the best parameter adaptation strategies

Scenario #2

General purpose black-box solvers (CPLEX, SCIP, ...)

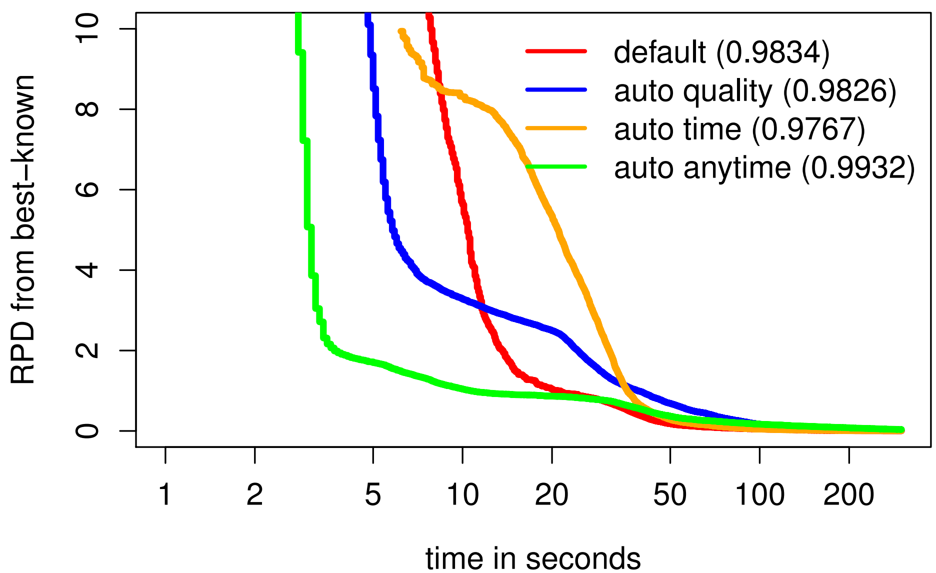
- Hundred of parameters
- Tuned by default for solving fast to *optimality*

Automatically Improving the Anytime Behavior

SCIP: an open-source mixed integer programming (MIP) solver (Achterberg, 2009)

- 200 parameters controlling search, heuristics, thresholds, ...
- Benchmark set: Winner determination problem for combinatorial auctions (Leyton-Brown et al., 2000)
  1 000 training + 1 000 testing instances
- Single run timeout: 300 seconds
- irace budget (*maxExperiments*): 5 000 runs

Automatically Improving the Anytime Behavior



Example #4

From Grammars to Parameters:
How to use irace to design algorithms from a grammar description?

Automatic Design of Algorithms: Top-down vs. Bottom-up

Top-down approaches

- Flexible frameworks:

  SATenstein (KhudaBukhsh et al., 2009)
  MOACO framework (López-Ibáñez and Stützle, 2012)
  MIP solvers: CPLEX, SCIP

- Automatic configuration tools:

  ParamILS (Hutter et al., 2009)
  irace (Birattari et al., 2010; López-Ibáñez et al., 2011)

Bottom-up approaches

- Based on GP and trees (Vázquez-Rodríguez & Ochoa, 2010)
- Based on GP and Lisp-like S-expressions (Fukunaga, 2008)
- Based on GE and a grammar description (Burke et al., 2012)

Bottom-up approach using grammars + irace
(Mascia, López-Ibáñez, Dubois-Lacoste, and Stützle, 2014)

Iterated Greedy (IG) for One-Dimensional Bin Packing

Burke, E.K., Hyde, M.R., Kendall, G.: **Grammatical evolution of local search heuristics**. *IEEE Transactions on Evolutionary Computation* 16(7), 406–417 (2012)

```
<program> ::= <choosebins>
              remove_pieces_from_bins()
              <repack>

<choosebins> ::= <type> | <type> <choosebins>

<type> ::= highest_filled(<num>, <ignore>, <remove>)
         | lowest_filled(<num>, <ignore>, <remove>)
         | random_bins(<num>, <ignore>, <remove>)
         | gap_lessthan(<num>, <threshold>, <ignore>, <remove>)
         | num_of_pieces(<num>, <numpieces>, <ignore>, <remove>)

      <num> ::= 2 | 5 | 10 | 20 | 50
<threshold> ::= average | minimum | maximum
<numpieces> ::= 1 | 2 | 3 | 4 | 5 | 6
   <ignore> ::= 0.995 | 0.997 | 0.999 | 1.0 | 1.1
   <remove> ::= ALL | ONE
   <repack> ::= best_fit | worst_fit | first_fit
```


GE representation

codons =

3	5	1	2	7	4

1. Start at <program>
2. Expand until rule with alternatives
3. Compute $(3 \bmod 2) + 1 = 2 \Rightarrow$ <type> <choosebins>
4. Compute $(5 \bmod 5) + 1 = 1 \Rightarrow$ highest_filled(<num>,)
5. ... until complete expansion or maximum number of wrappings

```
<program> ::= highest_filled(<num>, <ignore>)
              <choosebins>
              remove_pieces_from_bins()
              <repack>

     <num> ::= 2 | 5 | 10 | 20 | 50
```


Parametric representation

Parametric representation ⇒ Grammar expansion ?

```
--type highest-filled --num 5 --remove ALL ...
```

Grammar description → Parameter space → irace

Grammar ⇒ Parameter space ?

Parametric representation

Grammar ⇒ Parameter space ?

- Rules without alternatives ⇒ no parameter

```
<highest_filled> ::= highest_filled(<num>, <ignore>)
```


Parametric representation

Grammar ⇒ Parameter space ?

- Rules with alternative choices ⇒ categorical parameters

```
<type> ::= highest_filled(<num>, <ignore>, <remove>)
         | lowest_filled(<num>, <ignore>, <remove>)
         | random_bins(<num>, <ignore>, <remove>)
```

becomes

```
--type (highest_filled, lowest_filled, random_bins)
```


Parametric representation

Grammar ⇒ Parameter space ?

- Rules with numeric terminals ⇒ numerical parameters

```
<num> ::= [0, 100]
```

becomes

```
--num   [0, 100]
```


Parametric representation

Grammar ⇒ Parameter space ?

- Rules that can be applied more than once
  ⇒ one extra parameter per application

```
<choosebins> ::= <type> | <type> <choosebins>
      <type> ::= highest(<num>) | lowest(<num>)
```

can be represented by

--type1	{highest, lowest}	
--num1	(1, 5)	
--type2	{highest, lowest, ""}	
--num2	(1, 5)	if type2 $\neq$ ""
--type3	{highest, lowest, ""}	if type2 $\neq$ ""
...	...	

Summary

✔ irace works better than GE for designing IG algorithms for bin-packing and PFSP-WT

Franco Mascia, Manuel López-Ibáñez, Jérémie Dubois-Lacoste, and Thomas Stützle. **Grammar-based Generation of Stochastic Local Search Heuristics Through Automatic Algorithm Configuration Tools.** *Computers & Operations Research*, 2014.

✔ Not limited to IG!

Marie-Eléonore Marmion, Franco Mascia, Manuel López-Ibáñez, and Thomas Stützle. **Automatic Design of Hybrid Stochastic Local Search Algorithms**. In *Hybrid Metaheuristics*, vol. 7919 of LNCS, pages 144–158. Springer, 2013.

An overview of applications of irace

Done already

- Parameter tuning
  - single-objective optimization metaheuristics
  - MIP solvers (SCIP) with $>$ 200 parameters.
  - multi-objective optimization metaheuristics
  - anytime algorithms (improve time-quality trade-offs)
- Automatic algorithm design
  - From a flexible framework of algorithm components
  - From a grammar description
- Machine learning
  - Automatic model selection for high-dimensional survival analysis (Lang et al., 2014)
  - Hyperparameter tuning (mlr R package, Bischl et al.)

An overview of applications of irace

irace (and others) works great for

- Complex parameter spaces:
  numerical, categorical, ordinal, subordinate (conditional)
- Large parameter spaces (few hundred parameters)
- Heterogeneous instances
- Medium to large tuning budgets (thousands of runs)
- Individual runs require from seconds to hours
- Multi-core CPUs, MPI, Grid-Engine clusters

An overview of applications of irace

What we haven't deal with yet

- Extremely large parameter spaces (thousands of parameters)
- Extremely heterogeneous instances
- Small tuning budgets (500 or less runs)
- Very large tuning budgets (millions of runs)
- Individual runs require days
- Parameter tuning of decision algorithms / minimize time

We are looking for interesting benchmarks / applications!
Talk to us!

Acknowledgments

The tutorial has benefited from collaborations and discussions with our colleagues:

Prasanna Balaprakash, Mauro Birattari, Jérémie Dubois-Lacoste, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown, Tianjun Liao, Marie-Eléonore Marmion, Franco Mascia, Marco Montes de Oca, Leslie Pérez, Zhi Yuan.

The research leading to the results presented here has received funding from diverse projects:

- European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement nº 246939
- PAI project COMEX funded by the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office
- and the EU FP7 ICT Project COLOMBO, Cooperative Self-Organizing System for Low Carbon Mobility at Low Penetration Rates (agreement no. 318622)

fnrs
LA LIBERTÉ DE CHERCHER

Manuel López-Ibáñez and Thomas Stützle acknowledge support of the F.R.S.-FNRS of which they are a post-doctoral researcher and a research associate, respectively.

Questions

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

References I

T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009.

B. Adenso-Díaz and M. Laguna. Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research*, 54(1):99–114, 2006.

C. Ansótegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In I. P. Gent, editor, *Principles and Practice of Constraint Programming, CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 142–157. Springer, Heidelberg, Germany, 2009. doi: 10.1007/978-3-642-04244-7_14.

C. Audet and D. Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM Journal on Optimization*, 17(3):642–664, 2006.

C. Audet, C.-K. Dang, and D. Orban. Algorithmic parameter optimization of the DFO method with the OPAL framework. In K. Naono, K. Teranishi, J. Cavazos, and R. Suda, editors, *Software Automatic Tuning: From Concepts to State-of-the-Art Results*, pages 255–274. Springer, 2010.

P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In T. Bartz-Beielstein, M. J. Blesa, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 4771 of *Lecture Notes in Computer Science*, pages 108–122. Springer, Heidelberg, Germany, 2007.

T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. Sequential parameter optimization. In *Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005)*, pages 773–780, Piscataway, NJ, Sept. 2005. IEEE Press.

T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. The sequential parameter optimization toolbox. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 337–360. Springer, Berlin, Germany, 2010.

S. Becker, J. Gottlieb, and T. Stützle. Applications of racing algorithms: An industrial perspective. In E.-G. Talbi, P. Liardet, P. Collet, E. Lutton, and M. Schoenauer, editors, *Artificial Evolution*, volume 3871 of *Lecture Notes in Computer Science*, pages 271–283. Springer, Heidelberg, Germany, 2005.

M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*, volume 197 of *Studies in Computational Intelligence*. Springer, Berlin/Heidelberg, Germany, 2009. doi: 10.1007/978-3-642-00483-4.

M. Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2004.

References II

M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In W. B. Langdon et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002*, pages 11–18. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

M. Birattari, P. Balaprakash, and M. Dorigo. The ACO/F-RACE algorithm for combinatorial optimization under uncertainty. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. J. Gutjahr, R. F. Hartl, and M. Reimann, editors, *Metaheuristics – Progress in Complex Systems Optimization*, volume 39 of *Operations Research/Computer Science Interfaces Series*, pages 189–203. Springer, New York, NY, 2006.

M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-race and iterated F-race: An overview. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer, Berlin, Germany, 2010.

E. K. Burke, M. R. Hyde, and G. Kendall. Grammatical evolution of local search heuristics. *IEEE Transactions on Evolutionary Computation*, 16(7):406–417, 2012. doi: 10.1109/TEVC.2011.2160401.

M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9(5):403–432, Oct. 2006. doi: 10.1007/s10951-006-8495-8.

W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, NY, third edition, 1999.

S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7(1):77–97, 2001.

T. Dean and M. S. Boddy. An analysis of time-dependent planning. In *Proceedings of the 7th National Conference on Artificial Intelligence, AAAI-88*, pages 49–54. AAAI Press, 1988.

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. Automatic configuration of state-of-the-art multi-objective optimizers using the TP+PLS framework. In N. Krasnogor and P. L. Lanzi, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011*, pages 2019–2026. ACM Press, New York, NY, 2011. ISBN 978-1-4503-0557-0. doi: 10.1145/2001576.2001847.

A. S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):31–61, Mar. 2008. doi: 10.1162/evco.2008.16.1.31.

J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

References III

F. Hutter, D. Babić, H. H. Hoos, and A. J. Hu. Boosting verification by automatic tuning of decision procedures. In *FMCAD'07: Proceedings of the 7th International Conference Formal Methods in Computer Aided Design*, pages 27–34, Austin, Texas, USA, 2007a. IEEE Computer Society, Washington, DC, USA.

F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proc. of the Twenty-Second Conference on Artifical Intelligence (AAAI '07)*, pages 1152–1157, 2007b.

F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, Oct. 2009.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Automated configuration of mixed integer programming solvers. In A. Lodi, M. Milano, and P. Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010*, volume 6140 of *Lecture Notes in Computer Science*, pages 186–202. Springer, Heidelberg, Germany, 2010.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. A. Coello Coello, editor, *Learning and Intelligent Optimization, 5th International Conference, LION 5*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, Heidelberg, Germany, 2011.

A. R. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown. SATenstein: Automatically building local search SAT solvers from components. In C. Boutilier, editor, *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 517–524. AAAI Press, Menlo Park, CA, 2009.

M. Lang, H. Kotthaus, Marwedel, C. Weihs, J. Rahnenführer, and B. Bischl. Automatic model selection for high-dimensional survival analysis. *Journal of Statistical Computation and Simulation*, 2014. doi: 10.1080/00949655.2014.929131.

K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In A. Jhingran et al., editors, *ACM Conference on Electronic Commerce (EC-00)*, pages 66–76. ACM Press, New York, NY, 2000. doi: 10.1145/352871.352879.

M. López-Ibáñez and T. Stützle. The automatic design of multi-objective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875, 2012. doi: 10.1109/TEVC.2011.2182651.

M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011. URL http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf.

References IV

F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Computers & Operations Research*, 51:190–199, 2014. doi: 10.1016/j.cor.2014.05.020.

V. Nannen and A. E. Eiben. A method for parameter calibration and relevance estimation in evolutionary algorithms. In M. Cattolico et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2006*, pages 183–190. ACM Press, New York, NY, 2006. doi: 10.1145/1143997.1144029.

M. Oltean. Evolving evoluionary algorithms using linear genetic programming. *Evolutionary Computation*, 13(3):387–410, 2005.

P. Pellegrini, F. Mascia, T. Stützle, and M. Birattari. On the sensitivity of reactive tabu search to its meta-parameters. *Soft Computing*, 18(11):2177–2190, 2014. doi: 10.1007/s00500-013-1192-6.

E. Ridge and D. Kudenko. Tuning the performance of the MMAS heuristic. In T. Stützle, M. Birattari, and H. H. Hoos, editors, *International Workshop on Engineering Stochastic Local Search Algorithms (SLS 2007)*, volume 4638 of *Lecture Notes in Computer Science*, pages 46–60. Springer, Heidelberg, Germany, 2007.

R. Ruiz and C. Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494, 2005.

S. K. Smit and A. E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the 2009 Congress on Evolutionary Computation (CEC 2009)*, pages 399–406. IEEE Press, Piscataway, NJ, 2009.

S. K. Smit and A. E. Eiben. Beating the 'world champion' evolutionary algorithm via REVAC tuning. In H. Ishibuchi et al., editors, *Proceedings of the 2010 Congress on Evolutionary Computation (CEC 2010)*, pages 1–8. IEEE Press, Piscataway, NJ, 2010. doi: 10.1109/CEC.2010.5586026.

J. A. Vázquez-Rodríguez and G. Ochoa. On the automatic discovery of variants of the NEH procedure for flow shop scheduling using genetic programming. *Journal of the Operational Research Society*, 62(2):381–396, 2010.

S. Wessing, N. Beume, G. Rudolph, and B. Naujoks. Parameter tuning boosts performance of variation operators in multiobjective optimization. In R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pages 728–737. Springer, Heidelberg, Germany, 2010. doi: 10.1007/978-3-642-15844-5_73.

L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, June 2008.

References V

Z. Yuan, M. A. Montes de Oca, T. Stützle, and M. Birattari. Continuous optimization algorithms for tuning real and integer algorithm parameters of swarm intelligence algorithms. *Swarm Intelligence*, 6(1):49–75, 2012.

Z. Yuan, M. A. Montes de Oca, T. Stützle, H. C. Lau, and M. Birattari. An analysis of post-selection in automatic configuration. In C. Blum and E. Alba, editors, *Proceedings of GECCO 2013*, page to appear. ACM Press, New York, NY, 2013.

S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

Automatic (Offline) Configuration of Algorithms

Thomas Stützle

stuetzle@ulb.ac.be

http://iridia.ulb.ac.be/~stuetzle

Manuel López-Ibáñez

manuel.lopez-ibanez@ulb.ac.be

http://iridia.ulb.ac.be/~manuel

ULB UNIVERSITÉ LIBRE DE BRUXELLES, UNIVERSITÉ D'EUROPE

IRIDIA Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle