Automated Configuration of Optimisation Algorithms

Manuel López-Ibáñez

manuel.lopez-ibanez@manchester.ac.uk
http://lopez-ibanez.eu

University of Manchester, UK

The University of Manchester
Alliance Manchester Business School

NATCOR, 10 April 2024, Nottingham

Agenda

Part I: Automatic Algorithm Configuration

Part II: AAC and Automatic Algorithm Design

Part III: The irace Package

Part I

Automatic Algorithm Configuration (Overview)

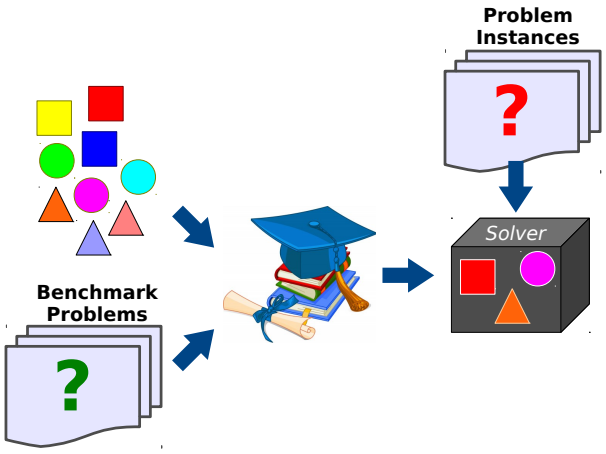
Mario's Pizza Delivery Problem [Birattari, 2004]

Mario collects phone orders for 30 minutes.
Mario 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 time for solving, say one minute (online)
- Limited time to implement an optimization algorithm, say one week (offline)

Traditional design of optimization algorithms

Human expert + intuition + trial-and-error/statistics



Design choices and parameters everywhere

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

- Exact solvers
  - Design choices: alternative models, pre-processing, variable selection, value selection, branching rules ...
    + numerical parameters
  - IBM CPLEX: 63 parameters that control the optimization
- (Meta)-heuristic solvers
  - Design choices: solution representation, operators, neighborhoods, pre-processing, strategies, ... + numerical parameters
  - Many are *hidden*

Design choices and parameters everywhere

Modern high-performance optimizers software involve a large number of design choices and (hyper)-parameter settings

Domain	Software	Parameters	
ML	WEKA	768	[Kotthoff et al., 2016]
	Auto-sklearn	110	[Feurer et al., 2015]
Code optimization	GCC	172 flags + 195 numerical	[Pérez Cáceres et al., 2017b]
Databases	Cassandra	23	[Silva-Muñoz et al., 2021]

Design choices and parameters everywhere

- *Categorical* parameters

  localsearch $\in$ { tabu search, SA, ILS }
- *Ordinal* parameters

  neighborhoods $\in$ { small, medium, large }
- *Numerical* parameters (integer and real-valued)

  population sizes, acceptance temperature, hidden constants, ...
- *Conditional* parameters are only active for specific values of other parameters:

  `temperature` only enabled if `localsearch == "SA"`

Configuring an algorithm means
setting its categorical, ordinal and numerical parameters

Algorithm configuration and design is hard

Challenges

- ✘ Many alternative design choices and parameter settings
- ✘ Nonlinear interactions among algorithm components and/or parameters
- ✘ Algorithms are stochastic
- ✘ Problem instances used for design (benchmark instances) are not identical to the ones found in the real-world
- ✘ Performance assessment is difficult (statistical analysis)

Traditional algorithm design and configuration

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?

The algorithm configuration problem

1. Find the best algorithm configuration
   given a set of *training problem instances*
2. Repeatedly use this algorithm configuration to solve
   unseen problem instances

A problem with many names:

offline parameter *tuning*,
automatic algorithm configuration,
hyper-parameter optimization,
hyper-heuristics, genetic programming,
meta-optimisation, programming by optimisation [Hoos, 2012], ...

Offline configuration vs. Online control

Offline tuning / Algorithm configuration

- Learn best configuration before *solving* the real problem instance
- Configuration done on training problem instances
- Performance measured over test ($\neq$ training) instances

Online tuning / Parameter control / Reactive search

- Learn best configuration *while* solving each instance
- No training phase but more expensive *while* solving
- Very popular in continuous optimization
- Ultimate goal: parameter-free algorithms

All online methods have parameters that are configured offline

The algorithm configuration problem

(extended from [Birattari, 2009])

A configuration scenario is defined by a tuple: $\langle \Theta, \mathcal{I}, m, \mathsf{S}, B \rangle$

- Θ : set of potential algorithm configurations (possibly infinite)
- $\mathcal{I}$: set of relevant problem instances (possibly infinite), from which instances are sampled with some probability
- $m(\theta, i)$: performance metric $m\colon \Theta \times \mathcal{I} \to \mathbb{R}$ of running *once* configuration $\theta \in \Theta$ on instance $i \in \mathcal{I}$. Typically noisy/stochastic!
- S : function (statistic) of $m(\theta, i)$ with respect to the distribution of the random variable $\mathcal{I}$, e.g., expected value of $m(\theta, i)$ over the distribution of $\mathcal{I}$
- B : budget for finding the best configuration, e.g., number of evaluations of $m(\theta, i)$

Find the best algorithm configuration θ^* such that:

$$\theta^* = \arg\min_{\theta \in \Theta} \mathsf{S}\big[m(\theta, i) \mid i \in \mathcal{I}\big]$$

Towards automatic algorithm configuration

Idea: Apply optimization + ML methods to configure algorithms

Automatic algorithm configuration and design

- apply optimization + ML methods to configure and design algorithms
- use computation power to explore algorithm design spaces
- free human creativity for higher level tasks

AC is a mixed-integer stochastic black-box optimization problem

Mixed-decision variables

- discrete (categorical, ordinal and integer) and real-valued
- conditional parameters, box-constraints and other constraints

Stochasticity

- of the target algorithm
- of the problem instances

Black-box

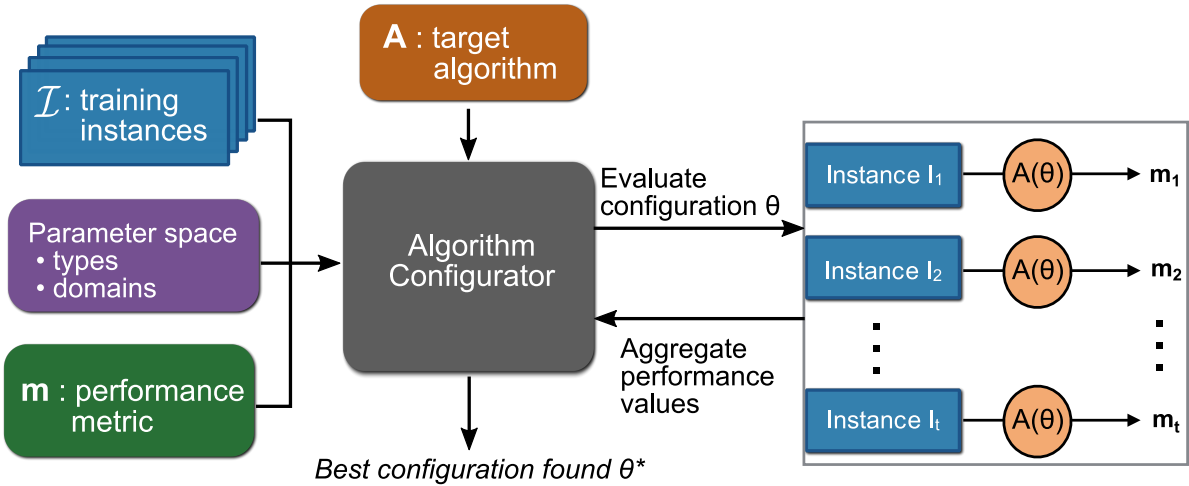
evaluation requires running a configuration on an instance

Typical tuning goals

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

AC requires specialized methods !

(Offline) Automatic Algorithm Configuration



Methods for Automatic Algorithm Configuration

experimental design, ANOVA

CALIBRA [Adenso-Díaz & Laguna, 2006]

others [Ridge & Kudenko, 2007; Coy et al., 2001; 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., 2007, 2009],
gender-based GA [Ansótegui et al., 2009], linear GP [Oltean, 2005],
REVAC(++) [Nannen & Eiben, 2006; Smit & Eiben, 2009, 2010] ...

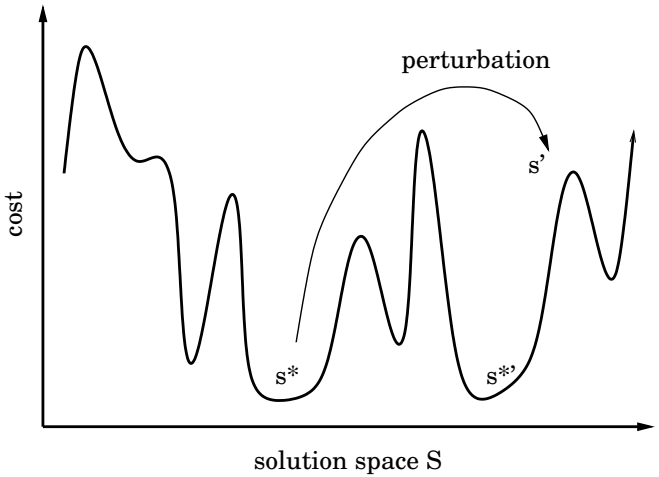
model-based

SPO [Bartz-Beielstein et al., 2005, 2010],
SMAC [Hutter et al., 2011], mlrMBO [Bischl et al., 2017]

sequential statistical testing

F-race, iterated F-race [Birattari et al., 2002; Balaprakash et al., 2007]
irace [López-Ibáñez et al., 2011] [López-Ibáñez et al., 2016]

ParamILS is an iterated local search method that works in the parameter space



Main design choices for ParamILS

Evaluation of configurations

- *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 minimizing computation time

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 $\mathcal{M}: \Theta \mapsto \mathbb{R}$
5: generate set of possible candidate configurations Θ
6: use model $\mathcal{M}$ to filter promising configurations $\Theta_p \subseteq \Theta$
7: evaluate configurations in Θ_p
8: $\Theta_0 := \Theta_0 \cup \Theta_p$
9: update $\theta^* \in \Theta_0$
10: **output:** θ^*

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
  $\Rightarrow$ Aggregate predictions from $\mathcal{M}$ 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 $\Rightarrow$ adaptive capping

The racing approach

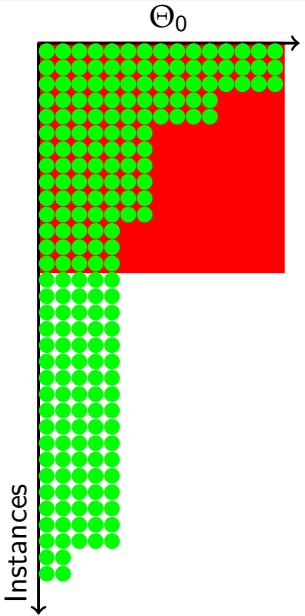
[Maron & Moore, 1997; Birattari et al., 2002]

Racing is a method for the *selection of the best*
among a given set of algorithm configurations

- ✔ Reduce effort evaluating low performance configurations
- ✔ Focus effort on selecting the best configurations

The racing approach

[Maron & Moore, 1997; Birattari et al., 2002]



Racing is a method for the *selection of the best* among a given set of algorithm configurations

- 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

How to discard?

Statistical tests!

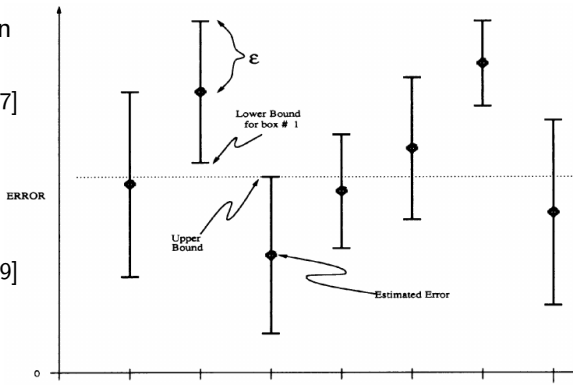
- Paired t-test with/*without* p-value correction (against the best)

  [Maron & Moore, 1997]

- *F-Race:* Friedman two-way analysis of variance by ranks
  + Friedman post-hoc test

  [Conover, 1999]

- *Bayesian:* Bayesian nonparametric statistics
  [Benavoli et al., 2015]



Taken from Maron & Moore [1997]

Sampling configurations

Racing (F-race, t-race, ...) is a method for the
selection of the best among a given set of algorithm configurations

How to define this set of configurations?

- Full factorial
- Random sampling
- Iterative update of a probabilistic sampling model
  ($\approx$ Estimation of Distribution Algorithm)

  $\Rightarrow$ *Iterated F-Race (I/F-Race)* [Balaprakash et al., 2007]

Iterated Racing

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

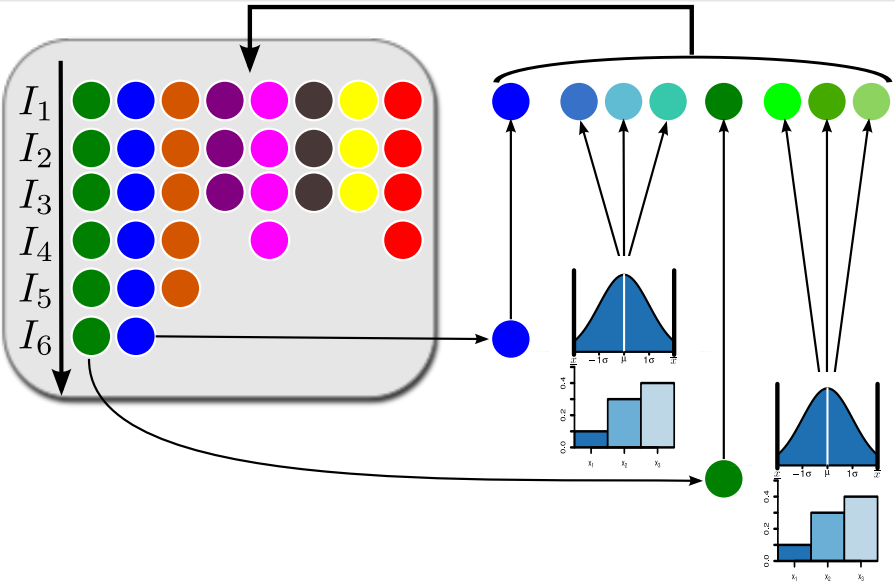
I/F-race: Balaprakash, Birattari & Stützle [2007], Birattari, Yuan, Balaprakash & Stützle [2010]

irace (v1): López-Ibáñez, Dubois-Lacoste, Stützle & Birattari [2011]

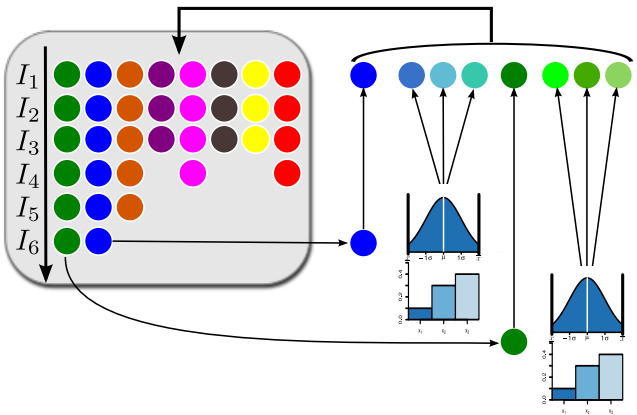
elitist irace (v2): López-Ibáñez, Dubois-Lacoste, Pérez Cáceres, Stützle & Birattari [2016]

elitist irace + adaptive capping (v3): Pérez Cáceres, López-Ibáñez, Hoos & Stützle [2017a]

Iterated Racing



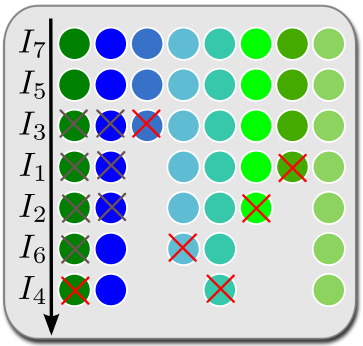
✘ Each new iteration (race) forgets the results of the previous one
⇒ Iterated F-race may "*lose*" the best-so-far configuration



Elitist Iterated Racing

[López-Ibáñez et al., 2016]

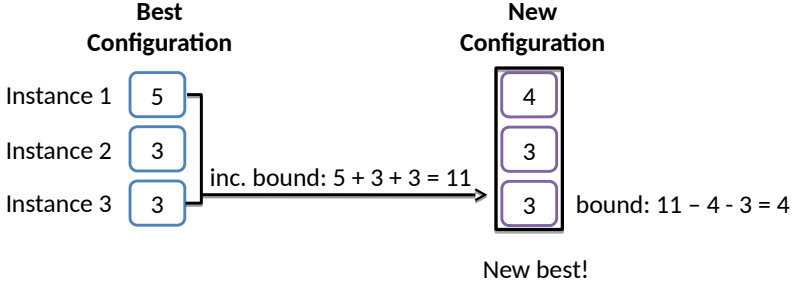
- ✘ Each new iteration (race) forgets the results of the previous one
  $\Rightarrow$ Iterated F-race may "*lose*" the best-so-far configuration
- ✔ Protect the best configurations (*elites*) from being discarded unless all their results are considered



- Extension of irace to better handle run-time minimization
- Configuration θ_1 evaluated on I_1 *dominates* θ_2 evaluated on I_2 if

$$I_2 \subset I_1 \quad \textit{and} \quad \sum_{i \in I_2} m(\theta_1, i) \leq \sum_{i \in I_2} m(\theta_2, i)$$

- Adaptive bound: $\kappa_i^{\text{new}} = \sum_{k=1}^{i} m(\theta_{\text{best}}, k) - \sum_{k=1}^{i-1} m(\theta_{\text{new}}, k)$
- Dominance elimination and adaptive capping:



An overview of applications of irace

- Parameter tuning
  - Exact MIP solvers (CPLEX, SCIP [López-Ibáñez & Stützle, 2014])
  - single-objective optimization metaheuristics
  - multi-objective optimization metaheuristics
  - anytime optimization (improve time-quality trade-offs)
  - command-line flags of GCC compiler [Pérez Cáceres et al., 2017b]
- Automatic algorithm design
  - From a flexible framework of algorithm components
  - From a grammar description
- Machine learning
  - Automatic model selection for survival analysis [Lang et al., 2014]
  - **mlr** R package [Bischl et al., 2013, 2016]
- Design of control software for robots [Francesca et al., 2015]
- Theoretical research [Friedrich et al., 2018; Dang & Doerr, 2019; Hall et al., 2019]

1 919 citations in Google Scholar, 189 000 downloads

Example #1

Automatically Improving the Anytime Behavior of Optimization Algorithms with irace

Manuel López-Ibáñez and Thomas Stützle.
Automatically improving the anytime behaviour of optimisation algorithms.
European Journal of Operational Research, 2014. doi: 10.1016/j.ejor.2013.10.043.

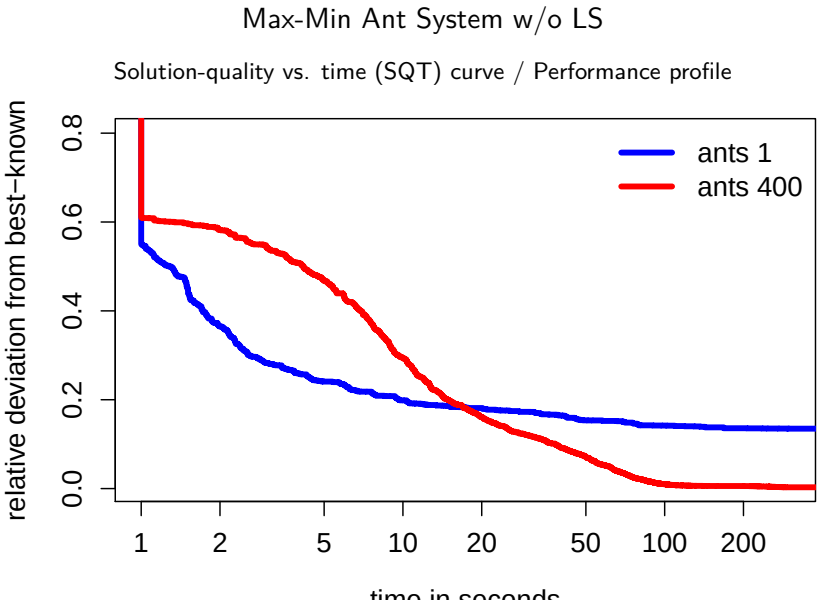
Automatically Improving the Anytime Behavior

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

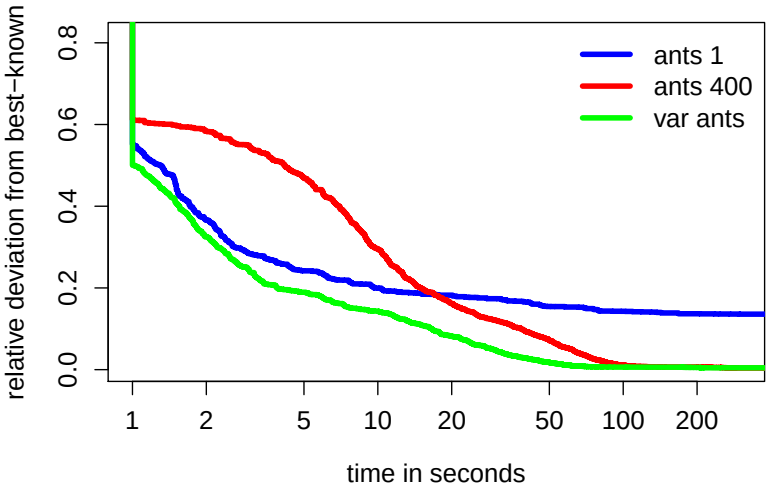
Good Anytime Behavior [Zilberstein, 1996]

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


Max-Min Ant System w/o LS
Solution-quality vs. time (SQT) curve / Performance profile
relative deviation from best−known
0.0
0.2
0.4
0.6
0.8
ants 1
ants 400
1
2
5
10
20
50
100
200
time in seconds

Automatically Improving the Anytime Behavior

Algorithms with good *"anytime" behaviour* produce as high quality result as possible at any moment of their execution [Zilberstein, 1996]



Improving Anytime Behaviour

How to improve the anytime behaviour of MMAS?

☞ Online parameter variation:

- Start with 1 ant, add 1 ant every iteration until 400 ants
- Start with $\beta = 10$, switch to $\beta = 2$ after 100 iterations
- ...

✘ More parameters!

✘ How to compare SQT curves?

Automatically Improving the Anytime Behavior

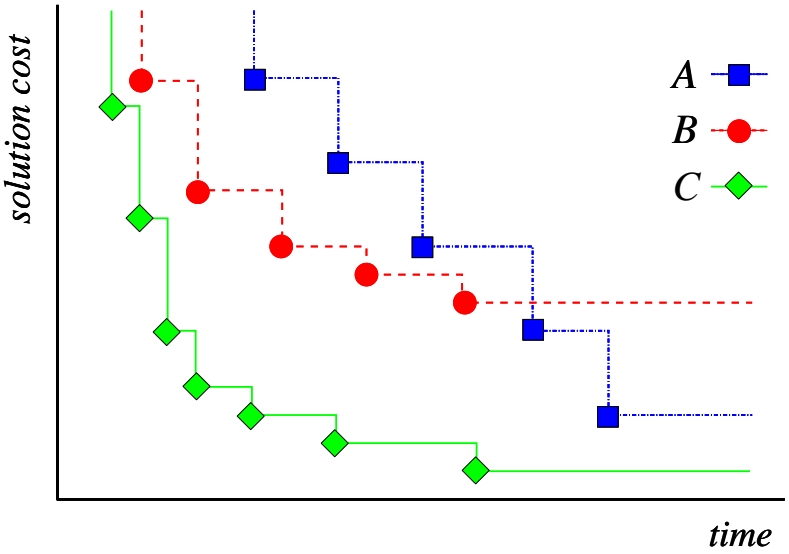
Online parameter control

- ✘ Which parameters to adapt? How? $\Rightarrow$ More parameters!
- ✔ Use irace (offline) to select the best parameter control strategies

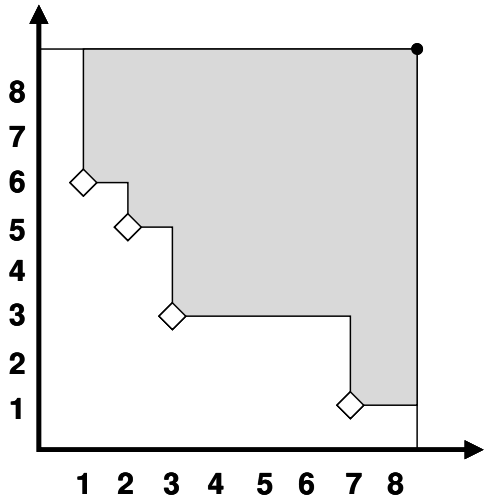
Improve Anytime Behavior

- ✔ More robust to different termination criteria
- ✘ How can irace compare SQT curves?

Automatically Improving the Anytime Behavior

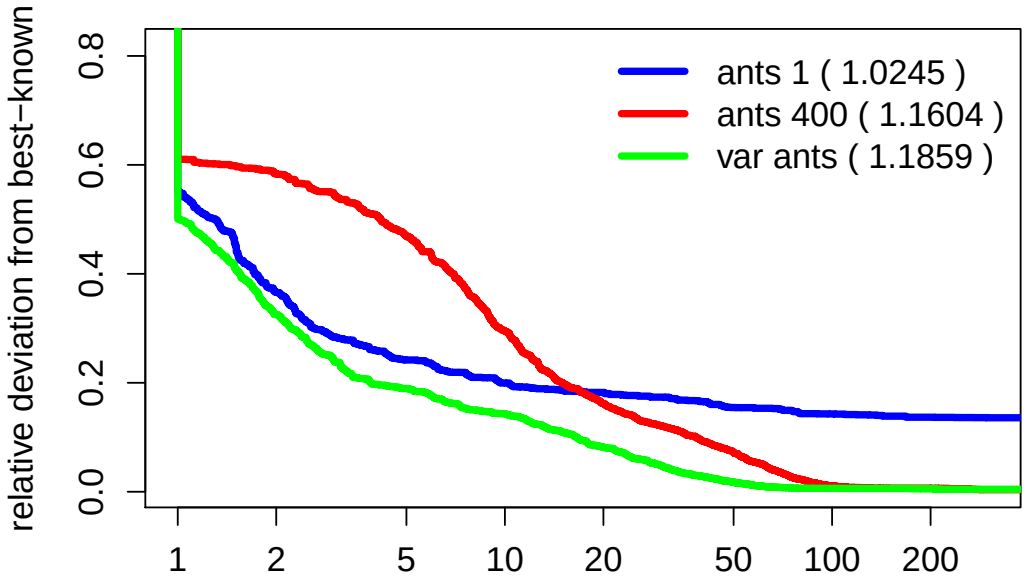


Automatically Improving the Anytime Behavior



Hypervolume measure $\approx$ Anytime behaviour

Automatically Improving the Anytime Behavior



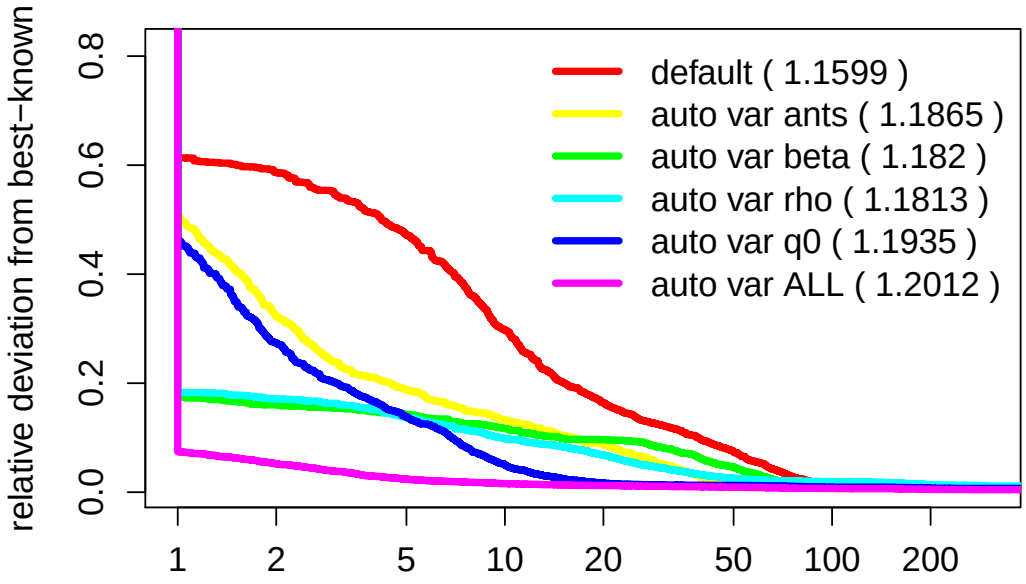
Automatically Improving the Anytime Behavior

irace + hypervolume = automatically improving the anytime behavior of optimization algorithms

1. Run configuration until large stopping time
2. Compute hypervolume of SQT curve
3. Evaluate anytime behavior according to hypervolume

- Hypervolume (multi-objective) optimization
  - ✔ Objectively defined comparison
  - ✔ Well-known performance measure
- Automatic configuration using irace
  - ✔ Most effort done by the computer
  - ✔ Best configurations selected by the computer: *Unbiased*

Scenario #1: Experimental comparison



Scenario #2: SCIP

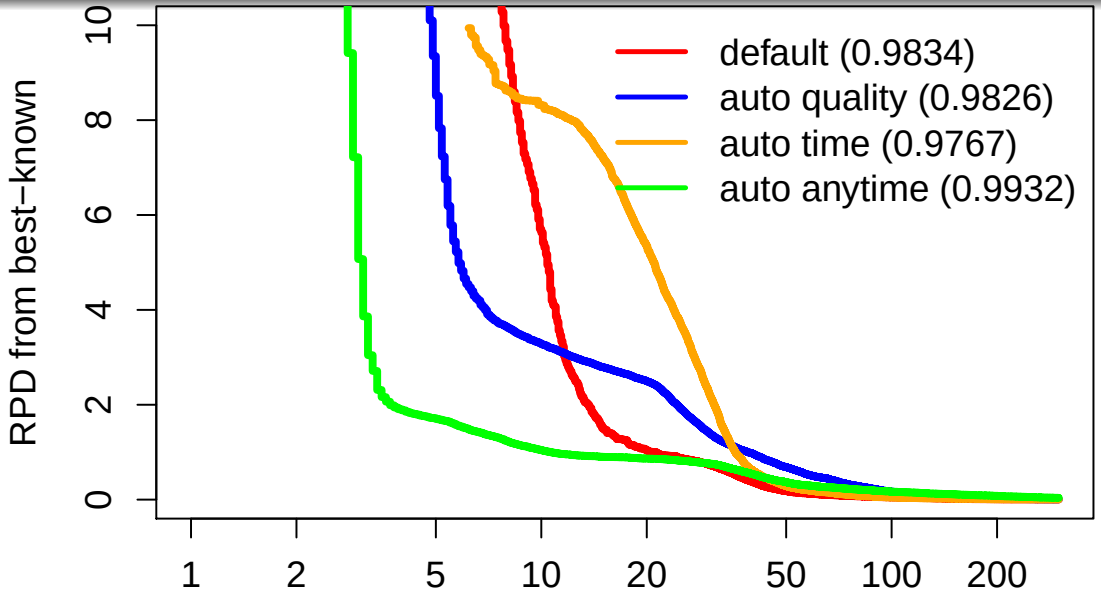
[López-Ibáñez & Stützle, 2014]

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

Scenario #2: SCIP

[López-Ibáñez & Stützle, 2014]



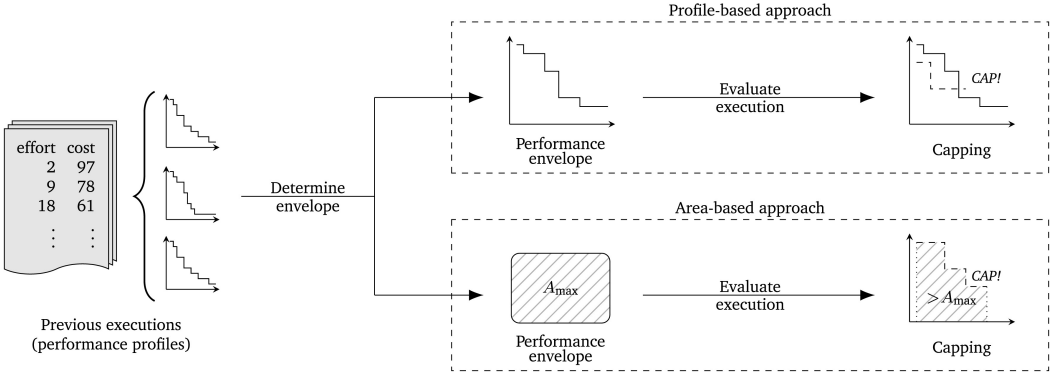
Example #2

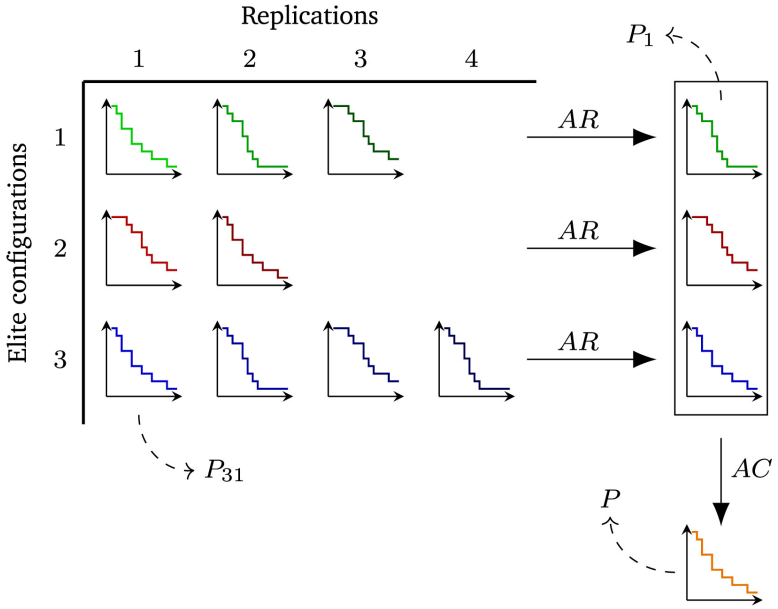
Capping methods for the automatic configuration of optimization algorithms

Marcelo De Souza, Marcus Ritt, and Manuel López-Ibáñez.
Capping methods for the automatic configuration of optimization algorithms.
Computers & Operations Research,2022. doi: 10.016/j.cor.2021.105615.

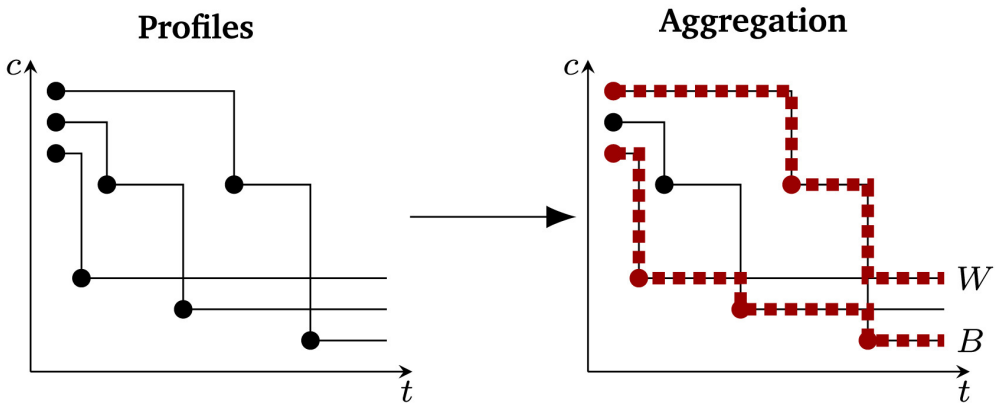
Capping methods for automatic algorithm configuration [De Souza et al., 2022]

- Adaptive capping only useful for decision algorithms (*time-to-target*)
- In many scenarios, we optimize cost over time until maximum termination time
  - ✘ Running bad configurations until maximum time is *wasteful*
  - ✔ Terminate (*cap*) bad configurations as soon as possible



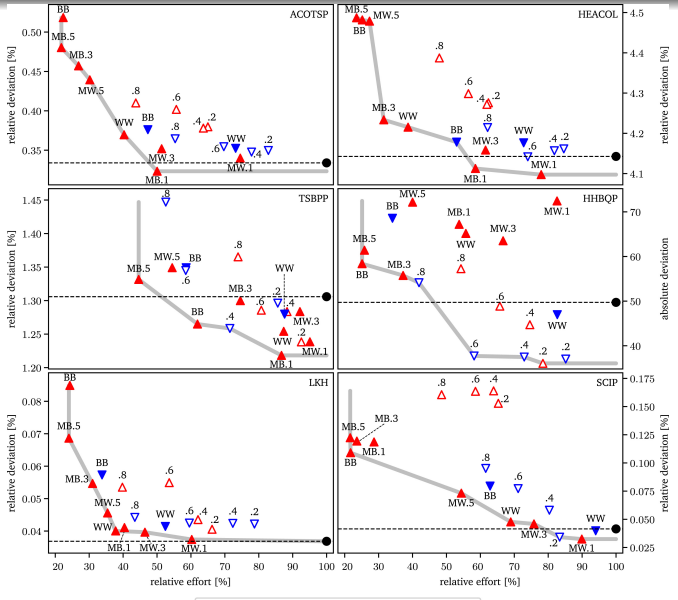

Replications
1
2
3
4
Elite configurations
1
2
3
P_{31}
AR
AR
AR
P_1
AC
P

Best and Worst Profile-based Aggregation Methods:



Capping methods for automatic algorithm configuration [De Souza et al., 2022]

P = profile-based
A = area-based
E = elitist
D = adaptive
B = best
W = worst
M = model



Capping methods for automatic algorithm configuration [De Souza et al., 2022]

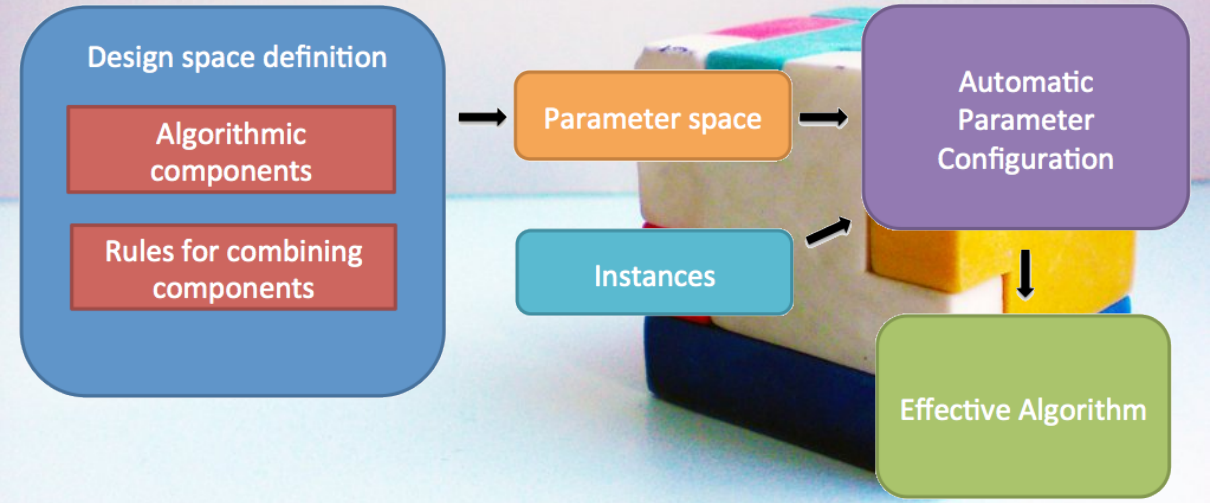
Category	Capping method	Relative effort [%]	Quality loss [%]					
			ACOTSP	HEACOL	TSBPP	HHBQP	LKH	SCIP
Conservative	AD.4	76.1	0.02	0.02	−0.05	−12.24	0.00	0.02
Aggressive	PEMB.1	53.0	−0.01	−0.03	−0.09	17.47	0.00	0.08

- AEBB for scenarios where the configuration budget is time
- Python add-on for irace: https://github.com/souzamarcelo/capopt

Part II

AAC and Automatic Algorithm Design

Automatic Algorithm Design: General approach



Main approaches for automatic algorithm design

Top-down approaches

- *Fixed algorithm template* with fixed alternatives for template parameters
- Examples:

MIP solvers: *CPLEX* [Hutter et al., 2010],
SCIP [López-Ibáñez & Stützle, 2014]
SATenstein [KhudaBukhsh et al., 2009]
MOACO framework [López-Ibáñez & Stützle, 2012]
AutoMOEA [Bezerra, López-Ibáñez & Stützle, 2016]
Fast-GA (ParadisEO) [Aziz-Alaoui, Doerr, Dreo, 2021]

Bottom-up approaches

- Library of algorithm components
- Rules for composing algorithms from components,
  e.g., via *grammars*
- Examples:

GP + trees [Vázquez-Rodríguez & Ochoa, 2010]
GE + grammars [Burke et al., 2012]
irace + *grammars* [Mascia et al., 2014]
[De Souza & Ritt, 2018]
(EMILI) [Pagnozzi & Stützle, 2019]

Example #3

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

Automatic design of hybrid SLS algorithms

[Marmion, Mascia, López-Ibáñez & Stützle, 2013]

1. *Decompose* single-point LS methods into components
2. Devise generalized LS (GLS) meta-metaheuristic
3. Describe possible GLS instantiations by a *grammar*
4. Convert the grammar to a *parametric representation*
   - ✔ Allows use of standard automatic configuration tools (*irace*)
   - ✔ Shows good performance when compared to grammatical evolution
     [Mascia, López-Ibáñez, Dubois-Lacoste & Stützle, 2014]

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, 2013.

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.

Manuel López-Ibáñez, Marie-Eléonore Marmion, and Thomas Stützle.
Automatic Design of Hybrid Metaheuristics from Algorithmic Components. *TR/IRIDIA/2017-012*, 2017.

Generalized Local Search

GLS Algorithm

1: $s_0 :=$ `Initialisation()`
2: `best_found` $:=$ **GLS(**s_0, `Perturb`, `Local_search`, `Acceptance`, `Stop`**)**
3: **return** `best_found`

Function GLS(s_0, `Perturb`, `Local_search`, `Acceptance`, `Stop`)

1: $s^* :=$ `Local_search`(s_0)
2: **while not** `Stop()` **do**
3: $\quad s' :=$ `Perturb`(s^*)
4: $\quad s'' :=$ `Local_search`(s')
5: $\quad s^* :=$ `Acceptance`(s'', s^*)
6: **return** s^*

- ✔ Many SLS methods may be instantiated from this template
- ✔ Hybridization when `Local_search` is another GLS

Classical SLS as instantiations of GLS

	Perturbation	**Local Search**	**Acceptance**
SA	One-move	$\varnothing$	Metropolis
PII	One-move	$\varnothing$	Metropolis (fixed temp.)
RII	One-move	$\varnothing$	Probabilistic
VNS	Variable-move	"any"	Variable-Accept
IG	Deconst-Construc	"any"	"any"
ILS	any	any	any
TS	$\varnothing$	TS-ls	Always

Grammar

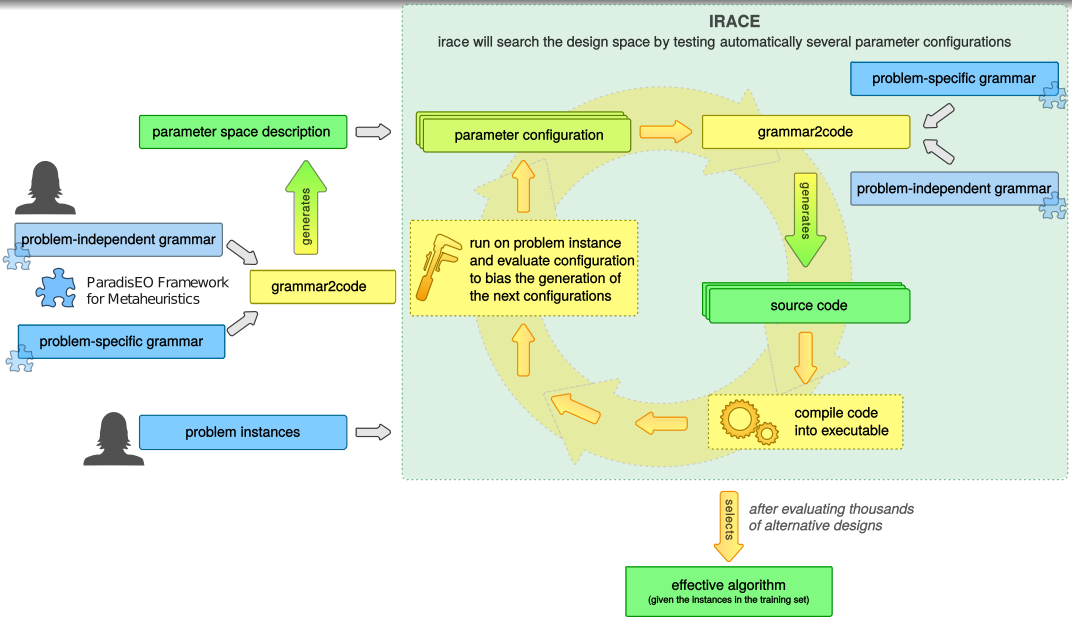
<start> ::= <Initialisation>; <GLS>; **return best_found;**
<GLS> ::= <known_MH> | <Hybrid>
<known_MH> ::= <ILS> | <SA> |<PII> | <RII> | <IG> | <VNS> | <TS>
<Hybrid> ::= **GLS(** <perturbation>, <GLS>, <acceptance>, <stop> **)**

<ILS> ::= **GLS(**<perturbation>, <local_search>, <acceptance>**)**
<PII> ::= **GLS(**<pert_one_move>, **LSNone**, **AcceptMetropolis(**<temperature>**))**
<SA> ::= **GLS(**<pert_one_move>, **LSNone**, <accept_metropolis>**)**
<RII> ::= **GLS(**<pert_one_move>, **LSNone**, **AcceptProbabilistic(** <probability> **))**
<VNS> ::= **GLS(**<pert_VNS>, <local_search>, <accept_VNS>**)**
<IG> ::= **GLS(**<ps:pert_destruct_construct>, <local_search>, <acceptance>**)**
<TS> ::= **GLS(PertNone**, <ls_TS>, **AcceptAlways)**

<accept_VNS> ::= **AcceptBetter** | <accept_skewed>
<pert_VNS> ::= <ps:pert_repeatable> **(**<pert_strength_dyn_incr>**)**
<perturbation> ::= **PertNone** | **PertRestart** | <pert_one_move>
| <pert_k_moves> | <pert_variable>
<pert_k_moves> ::= <ps:pert_repeatable> **(**<pert_strength_value>**)**
<pert_variable> ::= <ps:pert_repeatable> **(**<pert_strength_schedule>**)**
<local_search> ::= **LSNone**| <ls_first_imp> | <ls_best_imp > | <...>
<acceptance> ::= **AcceptAlways** | **AcceptBetter** | **AcceptBetterEqual**
| <accept_probab> | <accept_threshold> | <accept_metropolis>
<accept_metropolis> ::= **AcceptMetropolis(**<temp_init>, <temp_factor>, <temp_final>,
<temp_reheat_mode>**)**

System overview

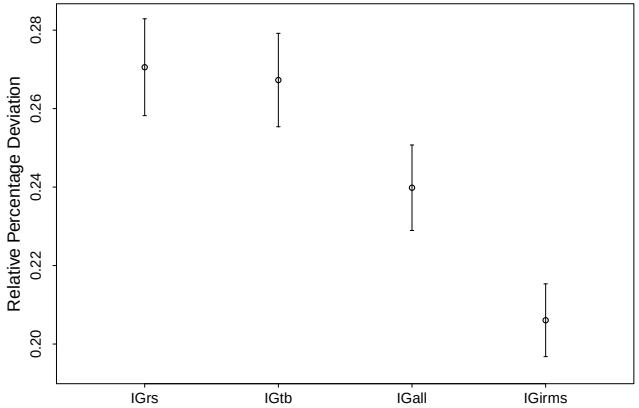
[Marmion, Mascia, López-Ibáñez & Stützle, 2013]



Flow-shop problem with makespan objective [Pagnozzi & Stützle, 2019]

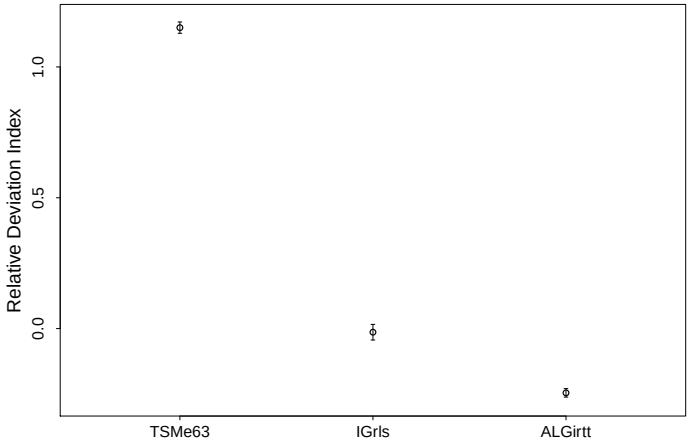
- **IGrs** [Ruiz & Stützle, 2007]
- **IGtb**: IGrs + tie-breaking rule by [Fernandez-Viagas & Framiñán, 2014]
- **IGall**: IG + LS on partial solutions [Dubois-Lacoste et al., 2017]

- **IGirms**: automatically designed by irace
  - max. three levels of recursion
  - biased / unbiased grammar resulting in 262 and 502 parameters, respectively
  - budget: 200 000 runs of $0.03 \cdot n \cdot m$ CPU-sec



Flow-shop problem with total tardiness objective [Pagnozzi & Stützle, 2019]

- **TSMe63**: Trajectory Scheduling Method [Li et al., 2015]
- **IGrls**: [Karabulut, 2016]
- **ALGirtt**: automatically designed by irace
  - max. three levels of recursion
  - budget: 200 000 runs of $0.03 \cdot n \cdot m$ sec



Results are clearly superior to state-of-the-art

Why automatic algorithm configuration and design?

1. More scientific, more principled
2. The end of the up-the-wall game
3. Computing power is exponentially cheaper
4. AC tools are becoming better
5. More interesting, fun and useful

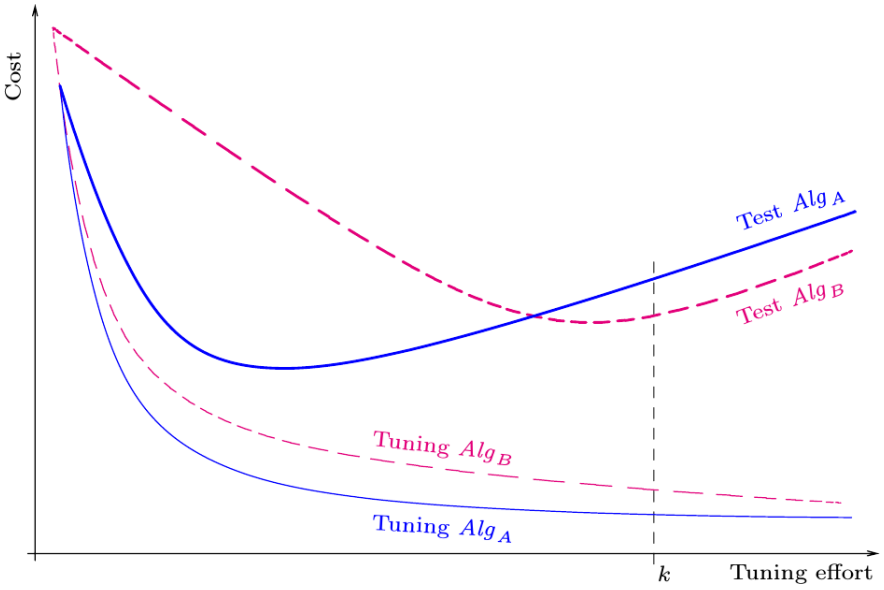
Reason #1: More scientific, more principled

- ✔ Reproducible results
- ✔ Fairer comparisons (best-effort)
- ✔ Avoid / reduce human biases
- ✔ Codify good practices

> *For procedures that require parameter tuning, the available data must be partitioned into a training and a test set. Tuning should be performed in the training set only.*
>
> [Journal of Heuristics: Policies on Heuristic Search Research]

> *The performance of swarm intelligence algorithms* [...] *is often strongly dependent on the value of the algorithm parameters. Such values should be set using either sound statistical procedures* [...] *or automatic parameter tuning procedures.*
>
> [Swarm Intelligence Journal (Springer)]

Over-tuning



(Taken from Birattari [2009])

Reason #2: The End of the Game

> *The Journal of Heuristics does not endorse the up-the-wall game.*
>
> [Journal of Heuristics: Policies on Heuristic Search Research]

> *True innovation in metaheuristics research therefore does not come from yet another method that performs better than its competitors, certainly if it is not well understood why exactly this method performs well.* *[Sörensen, 2015]*

- Finding a state-of-the-art algorithm is "easy":

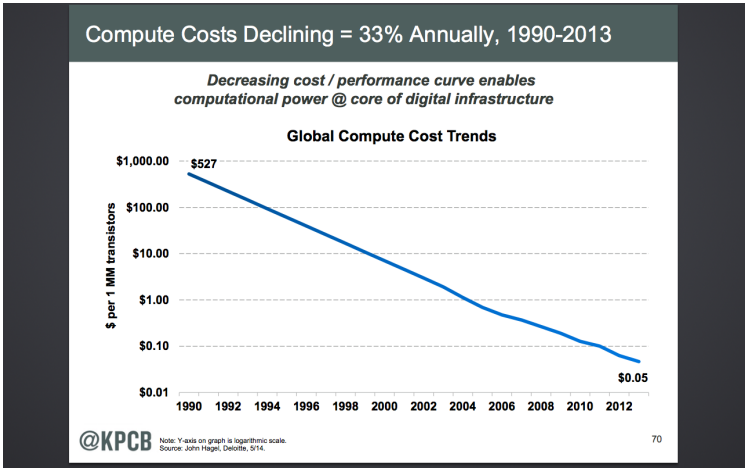
  problem modeling + algorithmic components + computing power

- *What* novel components? *Why* they work? *When* they work?

Reason #3: Computing power is exponentially cheaper

Algorithm Configuration in the Cloud [Geschwender et al., 2014]

Amazon EC2, 8 cores, 7GB memory, $ 0.58/hour



Reason #4: AC tools are becoming better

- Complex parameter spaces: numerical, categorical, ordinal, subordinate (conditional), constraints
- Large parameter spaces (hundreds of parameters)
- Heterogeneous problem instances
- Medium to large configuration budgets (few hundred to many thousands of runs)
- Individual runs may require from seconds to hours
- Multi-core CPUs, MPI, distributed computation clusters

☞ Modern automatic configuration tools (irace, SMAC, ...) are general, flexible, powerful and easy to use

Reason #5: More interesting, fun, and useful

✘ Classical optimization research:

1. Human-driven design to outperform other algorithmic designs
2. Analysis of the human-designed algorithm

✔ Paradigm shift in optimisation research:

From monolithic algorithms
to flexible frameworks of algorithmic components

1. Humans devise *novel* algorithmic components
2. Data-driven CPU-intensive automatic design
3. Analysis of generated data
4. Human-driven improvement of components

QUIT LIVING IN THE PAST

Part III

The irace Package

The irace Package

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari.
The irace package: Iterated Racing for Automatic Algorithm Configuration.
Operations Research Perspectives, 3:43–58, 2016. doi: 10.1016/j.orp.2016.09.002
https://mlopez-ibanez.github.io/irace/

- Implementation of Iterated Racing in R

  Goal 1: Flexible

  Goal 2: Easy to use

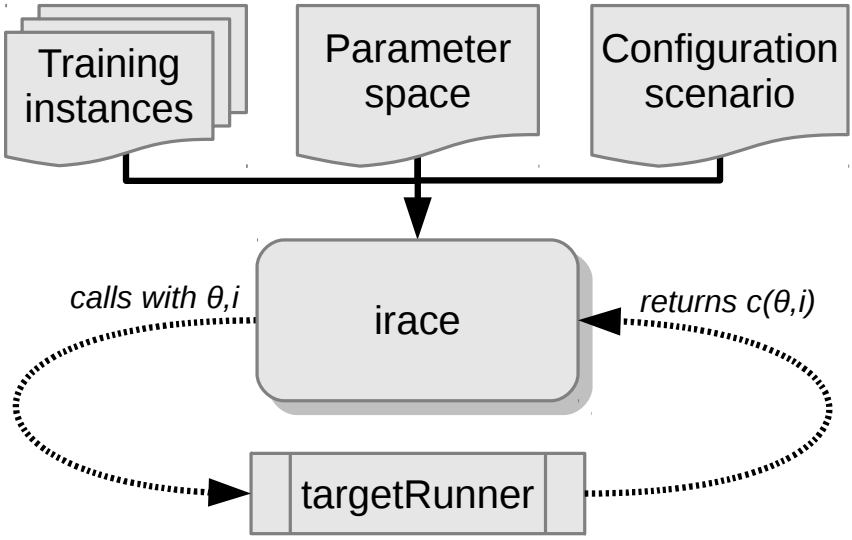
- R package available at CRAN (GNU/Linux, Windows, OSX)

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

- Use it through the command-line: (see `irace --help`)

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

✔ No knowledge of R needed


Training instances
Parameter space
Configuration scenario
irace
calls with θ,i
returns c(θ,i)
targetRunner

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`)
- Logarithmic scale (`,log`) *(irace 3.0)*

```
$ cat parameters.txt
```

```
# Name        Label/switch       Type   Domain          Condition
LS            "--localsearch "     c    (SA, TS, II)
rate          "--rate="            o    (low, med, high)
population    "--pop "           i,log  (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

- *maxExperiments* (*maxTime*): maximum number of runs (or overall time) of the target algorithm (tuning budget)
- *testType*: either F-test or t-test

The irace Package: target-runner

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

```
./target-runner configID instanceID seed instance configuration
```

e.g. :

```
./target-runner 2 1 1234567 3000-01.tsp --localsearch SA ...
```

- An R function

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

The irace Package: Other features

1. Initial configurations (e.g., default configuration)
2. Parallel evaluation: multiple CPUs, MPI, batch job clusters (SGE, PBS, Torque, Slurm)
3. Forbidden configurations (+ rejection):

   ```
   popsize < 5 & LS == "SA"
   ```

4. Recovery file: allows resuming an interrupted irace run
5. Test instances
6. Repair configurations before being evaluated
7. Adaptive capping (for runtime minimization)

The irace Package

Last version 3.5 (23/10/2022)

A detailed user-guide / tutorial:

https://cran.r-project.org/web/packages/irace/vignettes/irace-package.pdf

GitHub: https://github.com/MLopez-Ibanez/irace

Google group

https://groups.google.com/d/forum/irace-package

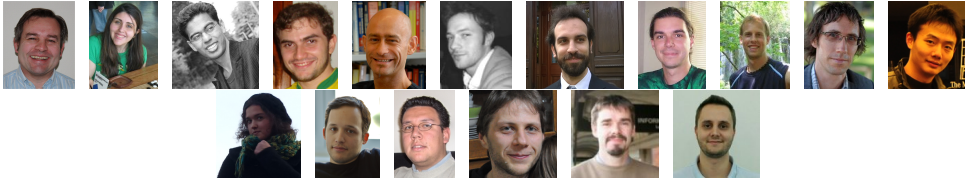
Exercises

https://lopez-ibanez.eu/2024-natcor-aac/

Acknowledgments

This tutorial has benefited from collaborations and discussions with my colleagues:

Thomas Stützle, Leslie Pérez Cáceres, Prasanna Balaprakash, Leonardo Bezerra, Mauro Birattari, Jérémie Dubois-Lacoste, Alberto Franzin, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown, Tianjun Liao, Marie-Eléonore Marmion, Franco Mascia, Marco Montes de Oca, Federico Pagnozzi, Zhi Yuan, Marcus Ritt, Marcelo De Souza



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. doi: 10.1137/0406208.

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. doi: 10.1007/978-3-540-75514-2_9.

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. doi: 10.1007/978-3-642-02538-9_14.

A. Benavoli, G. Corani, F. Mangili, and M. Zaffalon. A Bayesian nonparametric procedure for comparing algorithms. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, volume 37, pages 1264–1272. PMLR, 2015.

L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle. Automatic component-wise design of multi-objective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 20(3):403–417, 2016. doi: 10.1109/TEVC.2015.2474158.

L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle. Automatically designing state-of-the-art multi- and many-objective evolutionary algorithms. *Evolutionary Computation*, 28(2):195–226, 2020a. doi: 10.1162/evco_a_00263.

L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle. Automatic configuration of multi-objective optimizers and multi-objective configuration. In T. Bartz-Beielstein, B. Filipič, P. Korošec, and E.-G. Talbi, editors, *High-Performance Simulation-Based Optimization*, pages 69–92. Springer International Publishing, Cham, Switzerland, 2020b. doi: 10.1007/978-3-030-18764-4_4.

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

References II

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

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, 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. doi: 10.1007/978-3-642-02538-9_13.

B. Bischl, M. Lang, J. Bossek, L. Judt, J. Richter, T. Kuehn, and E. Studerus. *mlr: Machine Learning in R*, 2013. URL http://cran.r-project.org/package=mlr. R package.

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016.

B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang. mlrMBO: A modular framework for model-based optimization of expensive black-box functions. *Arxiv preprint arXiv:1703.03373 [stat.ML]*, 2017. URL http://arxiv.org/abs/1703.03373.

A. Blot, H. H. Hoos, L. Jourdan, M.-E. Kessaci-Marmion, and H. Trautmann. MO-ParamILS: A multi-objective automatic algorithm configuration framework. In P. Festa, M. Sellmann, and J. Vanschoren, editors, *Learning and Intelligent Optimization, 10th International Conference, LION 10*, volume 10079 of *Lecture Notes in Computer Science*, pages 32–47. Springer, Cham, Switzerland, 2016. doi: 10.1007/978-3-319-50349-3_3.

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.

W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, NY, 3rd 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.

N. Dang and C. Doerr. Hyper-parameter tuning for the $(1 + (\lambda, \lambda))$ GA. In M. López-Ibáñez, A. Auger, and T. Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 889–897. ACM Press, New York, NY, 2019. ISBN 978-1-4503-6111-8. doi: 10.1145/3321707.3321725.

M. De Souza and M. Ritt. Automatic grammar-based design of heuristic algorithms for unconstrained binary quadratic programming. In A. Liefooghe and M. López-Ibáñez, editors, *Proceedings of EvoCOP 2018 – 18th European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 10782 of *Lecture Notes in Computer Science*, pages 67–84. Springer, Heidelberg, Germany, 2018. doi: 10.1007/978-3-319-77449-7_5.

References III

M. De Souza, M. Ritt, and M. López-Ibáñez. Capping methods for the automatic configuration of optimization algorithms. *Computers & Operations Research*, 139: 105615, 2022. doi: 10.1016/j.cor.2021.105615.

T. Dean and M. S. Boddy. An analysis of time-dependent planning. In H. E. Shrobe, T. M. Mitchell, and R. G. Smith, editors, *Proceedings of the 7th National Conference on Artificial Intelligence, AAAI-88*, pages 49–54. AAAI Press/MIT Press, Menlo Park, CA, 1988. URL http://www.aaai.org/Conferences/AAAI/aaai88.php.

J. Dubois-Lacoste, F. Pagnozzi, and T. Stützle. An iterated greedy algorithm with optimization of partial solutions for the permutation flowshop problem. *Computers & Operations Research*, 81:160–166, 2017. doi: 10.1016/j.cor.2016.12.021.

V. Fernandez-Viagas and J. M. Framiñán. On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers & Operations Research*, 45:60–67, 2014.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS 28)*, pages 2962–2970, 2015. URL http://papers.nips.cc/book/advances-in-neural-information-processing-systems-28-2015.

G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, F. Mascia, V. Trianni, and M. Birattari. AutoMoDe-Chocolate: Automatic design of control software for robot swarms. *Swarm Intelligence*, 2015. doi: 10.1007/s11721-015-0107-9.

T. Friedrich, F. Quinzan, and M. Wagner. Escaping large deceptive basins of attraction with heavy-tailed mutation operators. In H. E. Aguirre and K. Takadama, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 293–300. ACM Press, New York, NY, 2018. doi: 10.1145/3205455.3205515.

D. Geschwender, F. Hutter, L. Kotthoff, Y. Malitsky, H. H. Hoos, and K. Leyton-Brown. Algorithm configuration in the cloud: A feasibility study. In P. M. Pardalos, M. G. C. Resende, C. Vogiatzis, and J. L. Walteros, editors, *Learning and Intelligent Optimization, 8th International Conference, LION 8*, volume 8426 of *Lecture Notes in Computer Science*, pages 41–46. Springer, Heidelberg, Germany, 2014. doi: 10.1007/978-3-319-09584-4_5.

J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986. doi: 10.1109/TSMC.1986.289288.

G. T. Hall, P. S. Oliveto, and D. Sudholt. On the impact of the cutoff time on the performance of algorithm configurators. In M. López-Ibáñez, A. Auger, and T. Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 907–915. ACM Press, New York, NY, 2019. ISBN 978-1-4503-6111-8. doi: 10.1145/3321707.3321879.

H. H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, Feb. 2012. doi: 10.1145/2076450.2076469.

References IV

F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In R. C. Holte and A. Howe, editors, *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1152–1157. AAAI Press/MIT Press, Menlo Park, CA, 2007.

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. doi: 10.1613/jair.2861.

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, CPAIOR 2010*, volume 6140 of *Lecture Notes in Computer Science*, pages 186–202. Springer, Heidelberg, Germany, 2010. doi: 10.1007/978-3-642-13520-0_23.

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. doi: 10.1007/978-3-642-25566-3.40.

K. Karabulut. A hybrid iterated greedy algorithm for total tardiness minimization in permutation flowshops. *Computers and Industrial Engineering*, 98(Supplement C):300 – 307, 2016.

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 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 517–524. AAAI Press, Menlo Park, CA, 2009.

L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 17:1–5, 2016.

M. Lang, H. Kotthaus, P. Marwedel, C. Weihs, J. Rahnenführer, and B. Bischl. Automatic model selection for high-dimensional survival analysis. *Journal of Statistical Computation and Simulation*, 85(1):62–76, 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.

X. Li, L. Chen, H. Xu, and J. N. D. Gupta. Trajectory scheduling methods for minimizing total tardiness in a flowshop. *Operations Research Perspectives*, 2:13–23, 2015. ISSN 2214–7160. doi: 10.1016/j.orp.2014.12.001.

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.

References V

M. López-Ibáñez and T. Stützle. Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research*, 235(3):569–582, 2014. doi: 10.1016/j.ejor.2013.10.043.

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/link/IridiaTr2011-004.pdf. Published in Operations Research Perspectives López-Ibáñez et al. [2016].

M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. doi: 10.1016/j.orp.2016.09.002.

M.-E. Marmion, F. Mascia, M. López-Ibáñez, and T. Stützle. Automatic design of hybrid stochastic local search algorithms. In M. J. Blesa, C. Blum, P. Festa, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 7919 of *Lecture Notes in Computer Science*, pages 144–158. Springer, Heidelberg, Germany, 2013. ISBN 978-3-642-38515-5. doi: 10.1007/978-3-642-38516-2_12.

O. Maron and A. W. Moore. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Research*, 11(1–5):193–225, 1997. doi: 10.1023/A:1006556606079.

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.

P. Miranda, R. M. Silva, and R. B. Prudêncio. I/S-Race: An iterative multi-objective racing algorithm for the SVM parameter selection problem. In *European Symposium on Artificial Neural Networks, ESSAN*, pages 573–578, 2015.

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.

A. J. Nebro, M. López-Ibáñez, C. Barba-González, and J. García-Nieto. Automatic configuration of NSGA-II with jMetal and irace. In M. López-Ibáñez, A. Auger, and T. Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO Companion 2019*, pages 1374–1381. ACM Press, New York, NY, 2019. ISBN 978-1-4503-6748-6. doi: 10.1145/3319619.3326832.

M. Oltean. Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, 13(3):387–410, 2005. doi: 10.1162/1063656054794815.

F. Pagnozzi and T. Stützle. Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems. *European Journal of Operational Research*, 276:409–421, 2019. doi: 10.1016/j.ejor.2019.01.018.

References VI

L. Pérez Cáceres, M. López-Ibáñez, H. H. Hoos, and T. Stützle. An experimental study of adaptive capping in irace. In R. Battiti, D. E. Kvasov, and Y. D. Sergeyev, editors, *Learning and Intelligent Optimization, 11th International Conference, LION 11*, volume 10556 of *Lecture Notes in Computer Science*, pages 235–250. Springer, Cham, Switzerland, 2017a. doi: 10.1007/978-3-319-69404-7_17.

L. Pérez Cáceres, F. Pagnozzi, A. Franzin, and T. Stützle. Automatic configuration of GCC using irace: Supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2017-009/, 2017b.

E. Ridge and D. Kudenko. Tuning the performance of the MMAS heuristic. In T. Stützle, M. Birattari, and H. H. Hoos, editors, *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics. 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.

R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

M. Silva-Muñoz, A. Franzin, and H. Bersini. Automatic configuration of the Cassandra database using irace. *PeerJ Computer Science*, 7:e634, 2021. doi: 10.7717/peerj-cs.634.

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, Piscataway, NJ, 2009. IEEE Press.

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, Piscataway, NJ, 2010. IEEE Press. doi: 10.1109/CEC.2010.5586026.

K. Sörensen. Metaheuristics—the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18, 2015. doi: 10.1111/itor.12001.

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.

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.

T. Zhang, M. Georgiopoulos, and G. C. Anagnostopoulos. S-Race: A multi-objective racing algorithm. In C. Blum and E. Alba, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2013*, pages 1565–1572. ACM Press, New York, NY, 2013. ISBN 978-1-4503-1963-8.

T. Zhang, M. Georgiopoulos, and G. C. Anagnostopoulos. Multi-objective model selection via racing. *IEEE Transactions on Cybernetics*, 46(8):1863–1876, 2016.

S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996. doi: 10.1609/aimag.v17i3.1232.

E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003. doi: 10.1109/TEVC.2003.810758.

Offline configuration vs. Online control

Offline tuning / Algorithm configuration

- Learn best configuration before *solving* the real problem instance
- Configuration done on training problem instances
- Performance measured over test ($\neq$ training) instances

Online tuning / Parameter control / Reactive search

- Learn best configuration *while* solving each instance
- No training phase
- Very popular in continuous optimization
- Ultimate goal: parameter-free algorithms

All online methods have parameters that are configured offline

MO tuning vs. Tuning MO algorithms [Bezerra et al., 2020b]

Multi-objective AAC

- Multiple metrics to evaluate an algorithm configuration

  Optimization: solution quality, computation time, memory ...

  ML: precision, recall, training time, prediction time ...

- AAC produces mutually nondominated *set* of configurations

  S-Race [Zhang et al., 2013], I/S-Race [Miranda et al., 2015],
  SPRINT [Zhang et al., 2016], MO-ParamILS [Blot et al., 2016]

MO tuning vs. Tuning MO algorithms

[Bezerra et al., 2020b]

Multi-objective AAC

- Multiple metrics to evaluate an algorithm configuration
- AAC produces mutually nondominated *set* of configurations

AAC for multi-objective algorithms

- Running a configuration outputs a *set* of mutually nondominated solutions (and/or *anytime* behavior)
- Unary quality metrics (hypervolume, epsilon, IGD+) evaluate the output

  [Zitzler et al., 2003]
- Uses *single-objective* AAC methods and produces a single best

  [López-Ibáñez & Stützle, 2012; Bezerra et al., 2016; Nebro et al., 2019; Bezerra et al., 2020a]

Multi-objective AAC of multi-objective algorithms is also possible!