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 '14, Jul 12-16 2014, Vancouver, BC, Canada
ACM 978-1-4503-2881-4/14/07.
http://dx.doi.org/10.1145/2598394.2605365

Part I

Automatic Algorithm Configuration (Overview)

Outline of Part I: Automatic Algorithm Configuration (Overview)

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, ...
  - powerful general-purpose software available
  - guarantees of optimality but often time/memory consuming
- Approximate algorithms
  - heuristics, local search, metaheuristics, hyperheuristics ...
  - typically special-purpose software
  - rarely provable guarantees but often fast and accurate

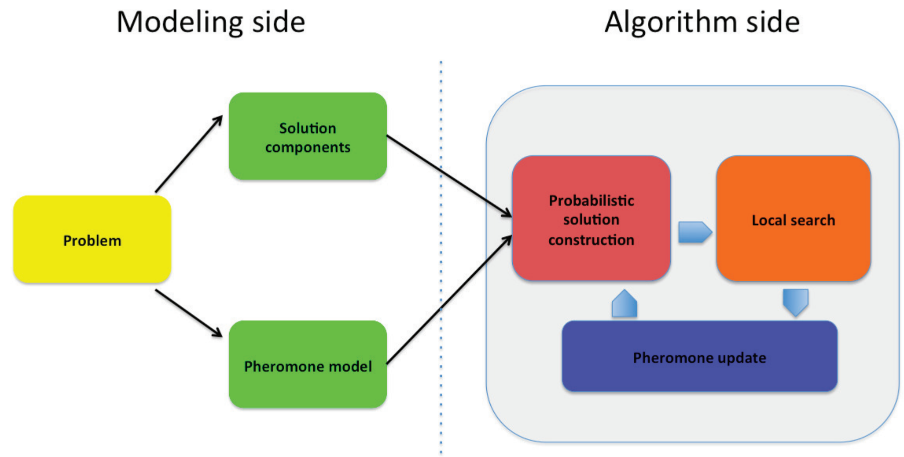
Much active research on hybrids between exact and approximate algorithms!

Design choices and parameters everywhere

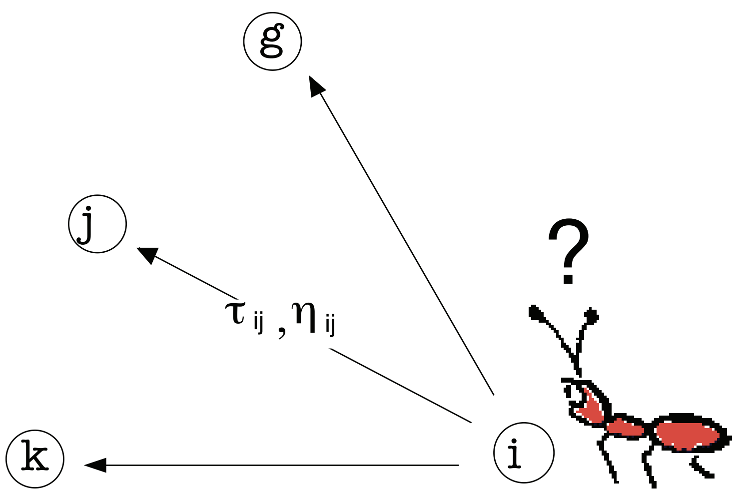
Todays high-performance optimizers involve a large number of design choices and parameter settings

- exact solvers
  - design choices include alternative models, pre-processing, variable selection, value selection, branching rules ...
  - many design choices have associated numerical parameters
  - example: SCIP 3.0.1 solver (fastest non-commercial MIP solver) has more than 200 relevant parameters that influence the solver's search mechanism
- approximate algorithms
  - design choices include solution representation, operators, neighborhoods, pre-processing, strategies, ...
  - many design choices have associated numerical parameters
  - example: multi-objective ACO algorithms with 22 parameters (plus several still hidden ones): see part 2 of tutorial

Example: Ant Colony Optimization



Probabilistic solution construction



ACO design choices and numerical parameters

- solution construction
  - choice of pheromone model
  - choice of heuristic information
  - choice of constructive procedure
  - numerical parameters
    - α, β influence the weight of pheromone and heuristic information, respectively
    - q_0 determines greediness of construction procedure
    - m, the number of ants
- pheromone update
  - which ants deposit pheromone and how much?
  - numerical parameters
    - ρ: evaporation rate
    - τ_0: initial pheromone level
- local search
  - ... many more ...

Parameter types

- *categorical* parameters **design**
  - choice of constructive procedure, choice of recombination operator, choice of branching strategy,...
- *ordinal* parameters **design**
  - neighborhoods, lower bounds, ...
- *numerical* parameters ***tuning, calibration***
  - integer or real-valued parameters
  - weighting factors, population sizes, temperature, hidden constants, ...
  - numerical parameters may be *conditional* to specific values of categorical or ordinal parameters

Design and configuration of algorithms involves setting categorical, ordinal, ***and*** *numerical parameters*

Algorithm design is difficult

Challenges

- many alternative design choices
- nonlinear interactions among algorithm components and/or parameters
- performance assessment is difficult

Optimization algorithm designers are aware that optimization algorithms require proper settings of parameters

Traditional approaches

- trial–and–error design guided by expertise/intuition
  $\rightsquigarrow$ prone to over-generalizations, implicit independence assumptions, limited exploration of design alternatives
- indications through theoretical studies
  $\rightsquigarrow$ 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

Remark: automatic here contrasts with the traditional manual algorithm design and parameter tuning; it implies that configuration is done by algorithmic tools with minimum manual intervention

Offline configuration and online parameter control

Offline configuration

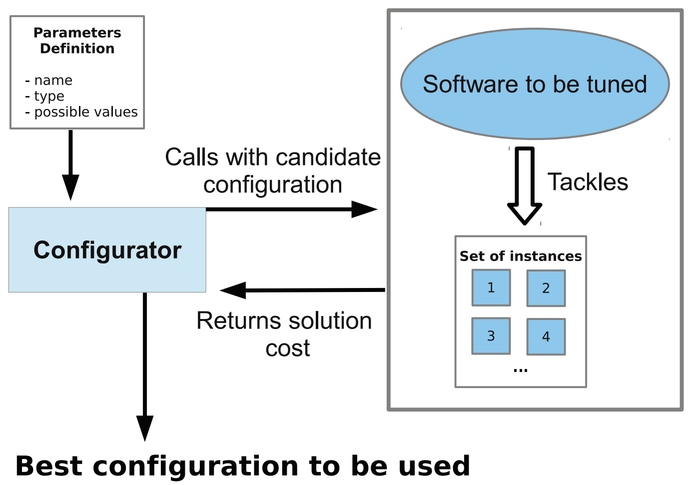
- configure algorithm before deploying it
- configuration done on training instances
- related to algorithm design

Online parameter control

- adapt parameter setting while solving an instance
- typically limited to a set of known crucial algorithm parameters
- related to parameter calibration

Offline configuration techniques can be helpful to configure (online) parameter control strategies

Offline configuration



Configuration is a stochastic optimization problem

Random influences

- stochasticity of the parameterized algorithms
- stochasticity through "sampling" of problem instances

Typical optimization goals

- maximize solution quality (within given time)
- minimize run-time (to decision, optimal solution)

Variables

- discrete (categorical, ordinal, integer) and continuous

Example of application scenario

Mario's Pizza delivery problem (Birattari, 2004; Birattari et al., 2002)

- Mario collects phone orders for 30 minutes
- scheduling deliveries is an optimization problem
- a different instance arises every 30 minutes
- limited amount of time for scheduling, say one minute
- good news: Mario has an SLS algorithm!
- . . . but the SLS algorithm must be tuned
- You have a limited amount of time for tuning it, say one week

Criterion:

Good configurations find good solutions for future instances!

The configuration problem: more formal
(Birattari, 2009; Birattari et al., 2002)

The configuration problem can be defined as a tuple

$$\langle \Theta, I, P_I, P_C, t, \mathcal{C}, T \rangle$$

Θ is the possibly infinite set of candidate configurations.

I is the possibly infinite set of instances.

P_I is a probability measure over the set I.

$t \colon I \to \Re$ is a function associating to every instance the computation time that is allocated to it.

$c(\theta, i, t(i))$ is a random variable representing the cost measure of a configuration $\theta \in \Theta$ on instance $i \in I$ when run for computation time $t(i)$.

- $C \subset \Re$ is the range of c
- P_C is a probability measure over the set C
  $P_C(c|\theta, i)$ give the probability that c is the cost of running configuration θ on instance i.
- $\mathcal{C}(\theta) = \mathcal{C}(\theta|\Theta, I, P_I, P_C, t)$ is the criterion that needs to be optimized with respect to θ.
- T is the total amount of time available for experimenting before delivering the selected configuration.

Performance measure

- solving the tuning problem requires finding a performance optimizing configuration $\bar{\theta}$, i.e.,

$$\bar{\theta} = \arg\min_{\theta \in \Theta} \mathcal{C}(\theta) \qquad (1)$$

- if we consider the expected value, we have

$$\mathcal{C}(\theta) = E_{I,C}[c] = \int c \; dP_C(c|\theta, i) dP_I(i), \qquad (2)$$

- however, analytical solution not possible, hence *estimate expected cost in a Monte Carlo fashion*

Approaches to configuration

- numerical optimization techniques
  - e.g. MADS (Audet & Orban, 2006), various (Yuan et al., 2012)
- heuristic search methods
  - e.g. meta-GA (Grefenstette, 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) ...
- experimental design, ANOVA
  - e.g. CALIBRA (Adenso-Díaz & Laguna, 2006), (Ridge & Kudenko, 2007; Coy et al., 2001; Ruiz & Maroto, 2005)
- model-based optimization approaches
  - e.g. SPO (Bartz-Beielstein et al., 2005, 2010), SMAC (Hutter et al., 2011)
- sequential statistical testing
  - e.g. F-race, iterated F-race (Birattari et al., 2002; Balaprakash et al., 2007)

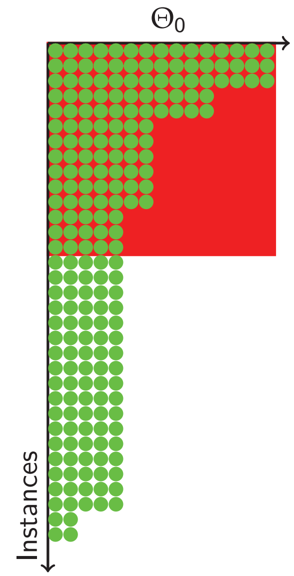
Remark: we focus on methods that are (i) applicable to all variable types, (ii) can deal with many variables, and (iii) use configuration across multiple training instances

Approaches to configuration

- numerical optimization techniques
  - e.g. MADS (Audet & Orban, 2006), various (Yuan et al., 2012)
- heuristic search methods
  - e.g. meta-GA (Grefenstette, 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) . . .
- experimental design, ANOVA
  - e.g. CALIBRA (Adenso-Díaz & Laguna, 2006), (Ridge & Kudenko, 2007; Coy et al., 2001; Ruiz & Maroto, 2005)
- model-based optimization approaches
  - e.g. SPO (Bartz-Beielstein et al., 2005, 2010), *SMAC* (Hutter et al., 2011)
- sequential statistical testing
  - e.g. F-race, *iterated F-race* (Birattari et al., 2002; Balaprakash et al., 2007)

Remark: we focus on methods that are (i) applicable to all variable types, (ii) can deal with many variables, and (iii) use configuration across multiple training instances

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 F-Race algorithm

Statistical testing

1. family-wise tests for differences among configurations
   - Friedman two-way analysis of variance by ranks
2. if Friedman rejects H_0, perform pairwise comparisons to best configuration
   - apply Friedman post-test (Conover, 1999)

Some (early) applications of F-race

International time-tabling competition (Chiarandini et al., 2006)
- winning algorithm configured by F-race
- interactive injection of new configurations

Vehicle routing and scheduling problem (Becker et al., 2005)
- first industrial application
- improved commercialized algorithm

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

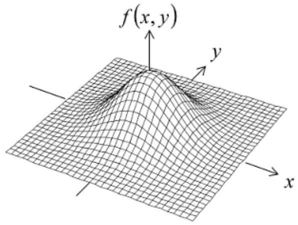
Sampling configurations

F-race is a method for the *selection of the best* configuration and independent of the way the set of configurations is sampled

Sampling configurations and F-race

- full factorial design
- random sampling design
- iterative refinement of a sampling model (iterated F-race)

Iterative race: an illustration



- sample configurations from initial distribution

While not terminate()

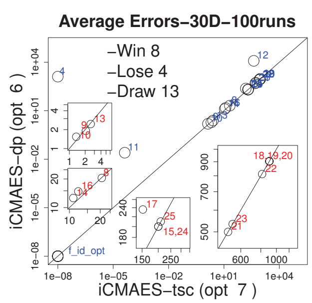
1. apply race
2. modify the distribution
3. sample configurations with selection probability

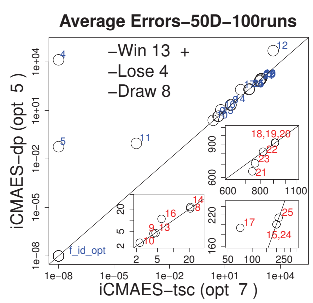
more details, see part 2 of the tutorial

Example application: configuring IPOP-CMAES

(Liao et al., 2013)

- IPOP-CMAES is state-of-the-art continuous optimizer
- configuration done on benchmark problems (instances) distinct from test set (CEC'05 benchmark function set) using seven numerical parameters





Related approach: Smit & Eiben (2010) configured another variant of IPOP-CMAES for three different objectives

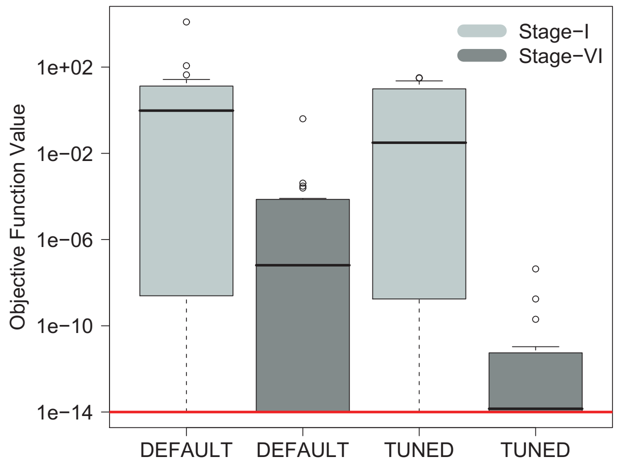
Tuning in-the-loop (re)design of continuous optimizers

(Montes de Oca et al., 2011)

- re-design of an incremental PSO algorithm for large-scale continuous optimization
- six steps
  - local search, call and control strategy of LS, PSO rules, bound constraint handling, stagnation handling, restarts
- iterated F-race used at each step to configure up to 10 parameters
- configuration done on 19 functions of dimension 10
- scaling examined until dimension 1000

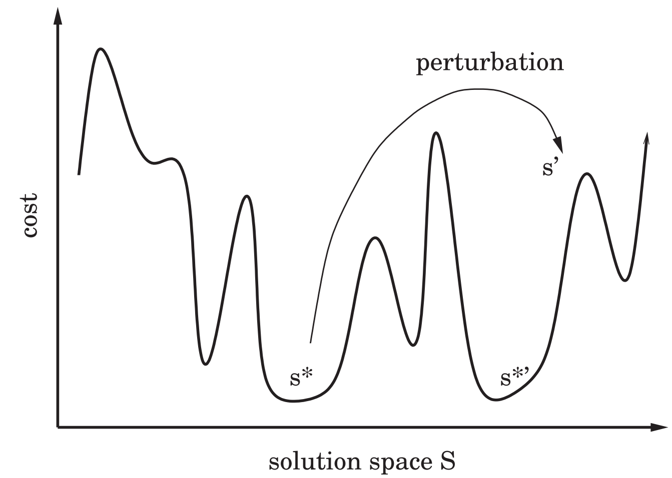
configuration results can help designer to gain insight useful for further development

Tuning in-the-loop (re)design of continuous optimizers
(Montes de Oca et al., 2011)



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
- ParamILS assumes all parameters to be categorical
- numerical parameters are discretized

Initialization
- select best configuration among default and r random configurations

Local search
- neighborhood is a 1-exchange neighborhood, where exactly one parameter changes a value at a time
- neighborhood is searched in random order

Perturbation
- change t randomly chosen variables

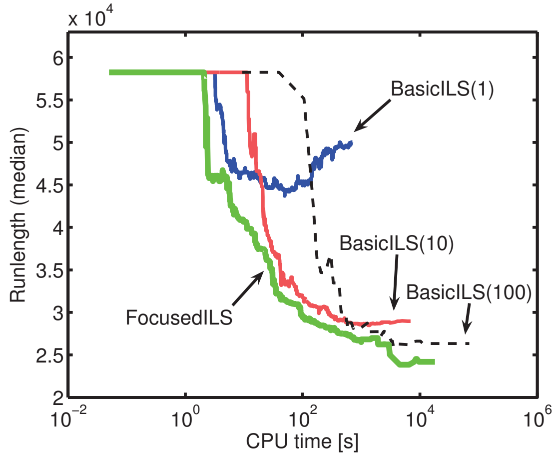
Main design choices for ParamILS

Acceptance criterion
- always select the better configuration

Evaluation of incumbent
- *BasicILS*: each configuration is evaluated on a same number of N instances
- *FocusedILS*: the number of instances on which the best configuration is evaluated increases at run time (intensification)

Adaptive Capping
- mechanism for early pruning the evaluation of poor candidate configurations
- particularly effective when configuring algorithms for minimization of computation time

ParamILS: BasicILS vs. FocusedILS



example: comparison of BasicILS and FocusedILS for configuring the SAPS solver for SAT-encoded quasi-group with holes, taken from (Hutter et al., 2007b)

Applications of ParamILS

ParamILS was widely used in various configuration tasks, many of which have tens and more parameters to be set.

- SAT-based verification (Hutter et al., 2007a)
  - configured SPEAR solver with 26 parameters, reaching for some instance classes speed-ups of up to 500 over defaults
- configuration of commercial MIP solvers (Hutter et al., 2010)
  - configured CPLEX (63 parameters), Gurobi (25 parameters) and lpsolve (47 parameters) for various instance distributions of MIP encoded optimization problems
  - speed-ups obtained ranged between a factor of 1 (that is, none) to 153 depending on problem and solver

Mixed integer programming (MIP) solvers

[Hutter, Hoos, Leyton-Brown, Stützle, 2009, Hutter, Hoos Leighton-Brown, 2010]

- MIP modelling widely used for tackling optimization problems
- powerful commercial (e.g. CPLEX) and non-commercial (e.g. SCIP) solvers available
- large number of parameters (tens to hundreds)

Benchmark set	Default	Configured	Speedup
`Regions200`	72	10.5 (11.4 ± 0.9)	*6.8*
`Conic.SCH`	5.37	2.14 (2.4 ± 0.29)	*2.51*
`CLS`	712	23.4 (327 ± 860)	*30.43*
`MIK`	64.8	1.19 (301 ± 948)	*54.54*
`QP`	969	525 (827 ± 306)	*1.85*

FocusedILS, 10 runs, 2 CPU days, 63 parameters

Gender-based genetic algorithms (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

Results

- 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
- mutation as uniform variation in specific interval
- "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)
- few tests, only real-valued parameters, limited experiments

Other continuous optimizers (Yuan et al., 2012)

- study of CMAES, BOBYQA, MADS, and irace-model for tuning continuous and quasi-continuous parameters
- BOBYQA best for few; CMAES best for larger number of parameters
- post-selection mechanism appears promising (Yuan et al., 2012, 2013)

Surrogate Model-assisted approaches

Use surrogate models of search landscape to predict the performance of specific parameter configurations

Algorithmic scheme

Generate initial set of configurations Θ_0; evaluate Θ_0, choose best-so-far configuration θ^*, $i := 1$, $\Theta_i := \Theta_0$
while computation budget available **do**
  Learn surrogate model $\mathcal{M} : \Theta \mapsto R$ based on Θ_i
  Use model $\mathcal{M}$ to select promising configurations Θ_p
  Evaluate configurations in Θ_p, update θ^*, $\Theta_i := \Theta_i \cup \Theta_p$,
  $i := i + 1$
Output: θ^*

Sequential parameter optimization (SPO) framework

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

SPO is a prominent method to parameter tuning using a surrogate model-assisted approach

Main design decisions

- uses in most variants Gaussian stochastic processes for $\mathcal{M}$
- promise of candidates is defined through expected improvement criterion
- intensification mechanism increases number of evaluations of best-so-far configuration θ^*

Practicalities

- SPO is implemented in the comprehensive SPOT R-package
- most applications, however, to numerical parameter tuning, single instances

Sequential Model-based Algorithm Configuration (SMAC)

(Hutter et al., 2011)

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

Main design decisions

- uses random forests as model $\mathcal{M}$ to allow modelling categorical variables
- predictions from model for single instances are aggregated across multiple ones
- promising configurations identified through local search on the surrogate model surface (expected improvement criterion)
- can use instance features to improve performance predictions
- intensification mechanism similar to that of FocusedILS
- further extensions through introduction of capping

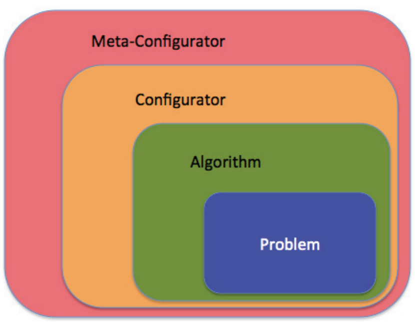
Why automatic algorithm configuration?

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

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

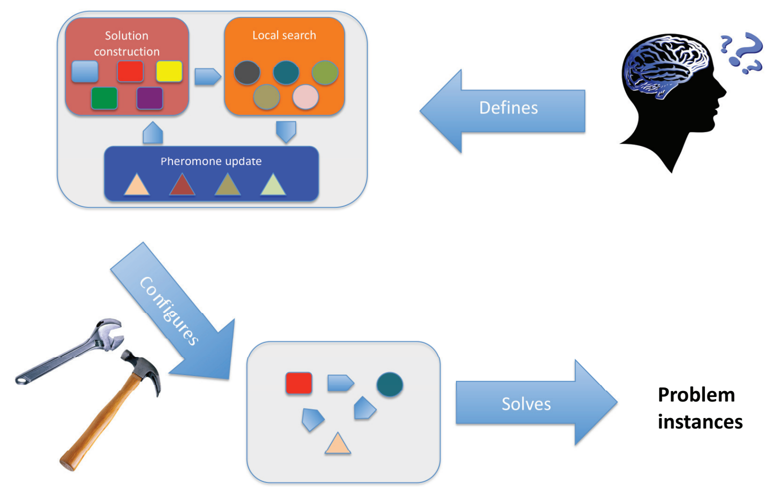
Configuring configurators

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



- can be done (example, see (Hutter et al., 2009)), but . . .
- it is costly and iterating further leads to diminishing returns

Towards a shift of paradigm in algorithm design



Part II

Iterated Racing (irace)

What is Iterated Racing and irace?

Iterated Racing (irace)

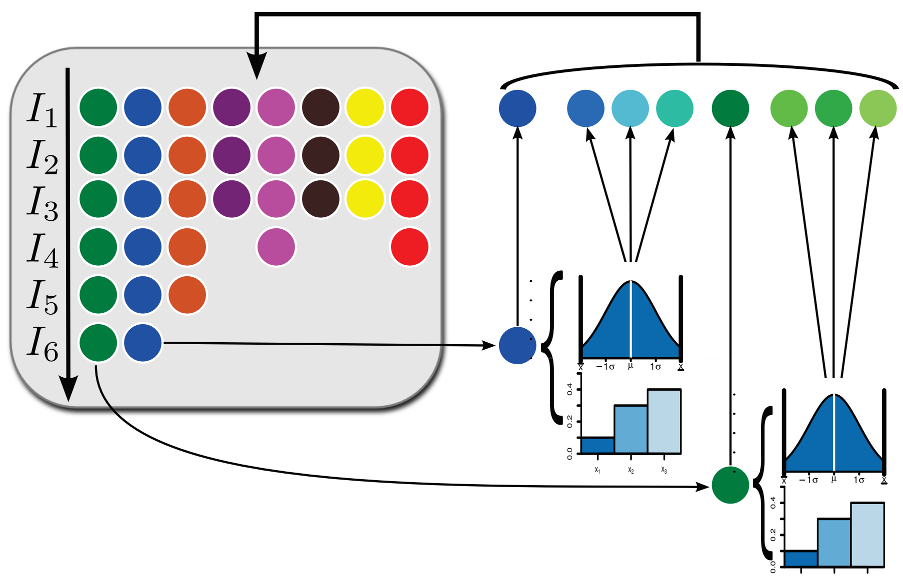
1. A variant of I/F-Race with several extensions
   - I/F-Race proposed by Balaprakash, Birattari, and Stützle (2007)
   - Refined by Birattari, Yuan, Balaprakash, and Stützle (2010)
   - Further refined and extended by López-Ibáñez, Dubois-Lacoste, Stützle, and Birattari (2011)
2. A software package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

Iterated Racing

Iterated Racing ⊇ 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



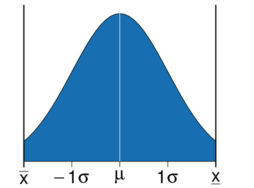
Iterated Racing

Require:
Training instances: $\{I_1, I_2, \dots\} \sim \mathcal{I}$,
Parameter space: X,
Cost measure: $\mathcal{C}\colon \Theta \times \mathcal{I} \to \mathbb{R}$,
Tuning budget: B

$\Theta_1 := \texttt{SampleUniform}(X)$
$\Theta^{\text{elite}} := \texttt{Race}(\Theta_1,\ B_1)$
$i := 2$
while $B_{\text{used}} \leq B$ **do**
  $\Theta^{\text{new}} := \texttt{UpdateAndSample}(X, \Theta^{\text{elite}})$
  $\Theta_i := \Theta^{\text{new}} \cup \Theta^{\text{elite}}$
  $\Theta^{\text{elite}} := \texttt{Race}(\Theta_i,\ B_i)$
  $i := i + 1$
Output: Θ^{elite}

Iterated Racing: Sampling distributions

Numerical parameter $X_d \in [\underline{x}_d, \overline{x}_d]$
$\Rightarrow$ *Truncated normal distribution*

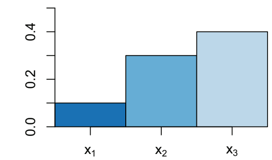


$$\mathcal{N}(\mu_d^z, \sigma_d^i) \in [\underline{x}_d, \overline{x}_d]$$

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

σ_d^i = decreases with the number of iterations

Categorical parameter $X_d \in \{x_1, x_2, \dots, x_{n_d}\}$
$\Rightarrow$ *Discrete probability distribution*



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

x_1	x_2	$\dots$	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
   Categorical parameters : "smoothing" of probabilities, increase low values, decrease high values.

   Numerical parameters : σ_d^i is "brought back" to its value at two iterations earlier, approx. σ_d^{i-2}
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

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

- R package available at CRAN:

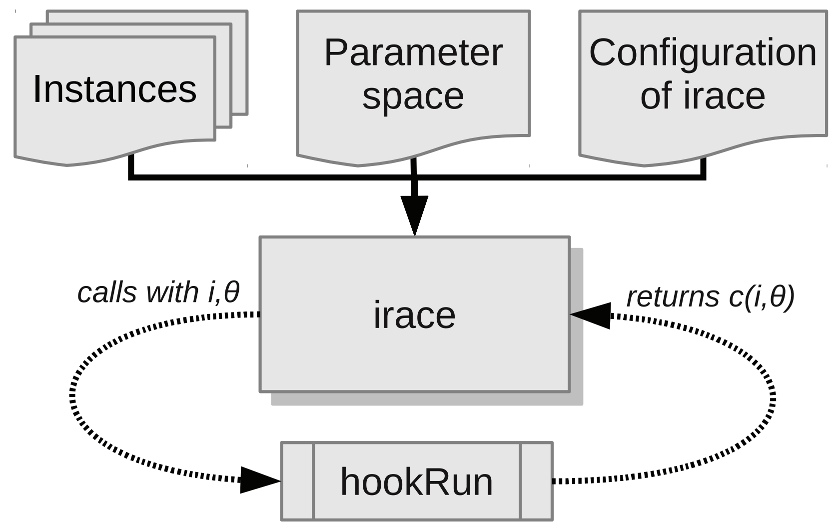
  http://cran.r-project.org/package=irace

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

- Use it from inside R . . .

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


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
CANDIDATE=$2
shift 2 || exit 1
CAND_PARAMS=$*
STDOUT="c${CANDIDATE}.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

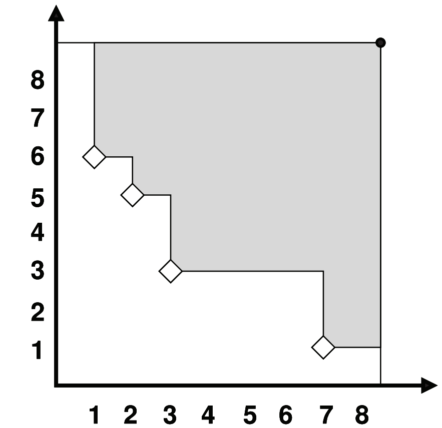
A more complex example: MOACO framework

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 a Pareto front!



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

A more complex example: MOACO framework

- Use a common reference point $(2.1, 2.1, \dots)$
- Normalize the objectives range to $[1, 2]$ per instance without predefined maximum / minimum

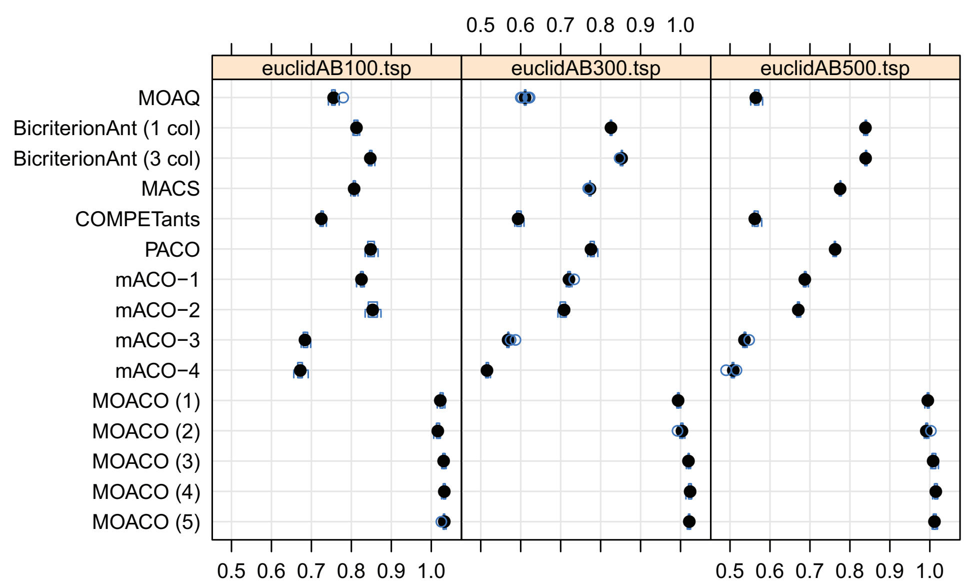
☞ We need all Pareto fronts for computing the normalization!

✘ We cannot simply use `hook-run`

✔ We use `hook-evaluate` !
  - `hook-evaluate` ≈ `hook-run`
  - Executes *after* all `hook-run` for a given instance
  - Returns the cost value instead of `hook-run`

`./hook-evaluate` *`instance candidate-number total-candidates`*

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)*

Example #3

Automatically Improving the Anytime Behavior of Optimization Algorithms with irace

Automatically Improving the Anytime Behavior

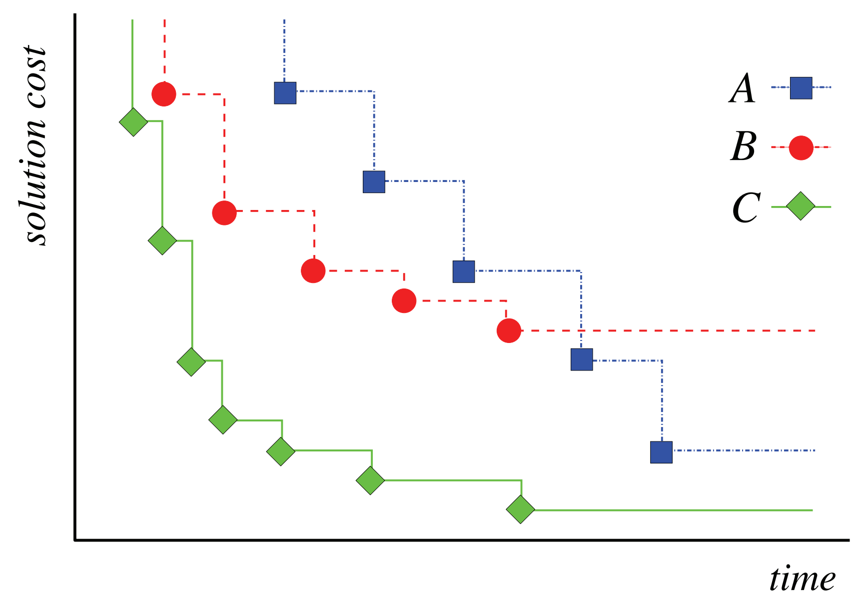
Anytime Algorithm (Dean & Boddy, 1988)

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

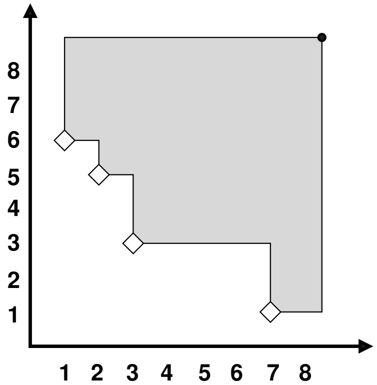
Good Anytime Behavior (Zilberstein, 1996)

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

Automatically Improving the Anytime Behavior



Automatically Improving the Anytime Behavior



Hypervolume measure ≈ Anytime behaviour

Manuel López-Ibáñez and Thomas Stützle. **Automatically improving the anytime behaviour of optimisation algorithms**. Technical Report TR/IRIDIA/2012-012, IRIDIA, Université Libre de Bruxelles, Belgium, 2012.

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, . . .)

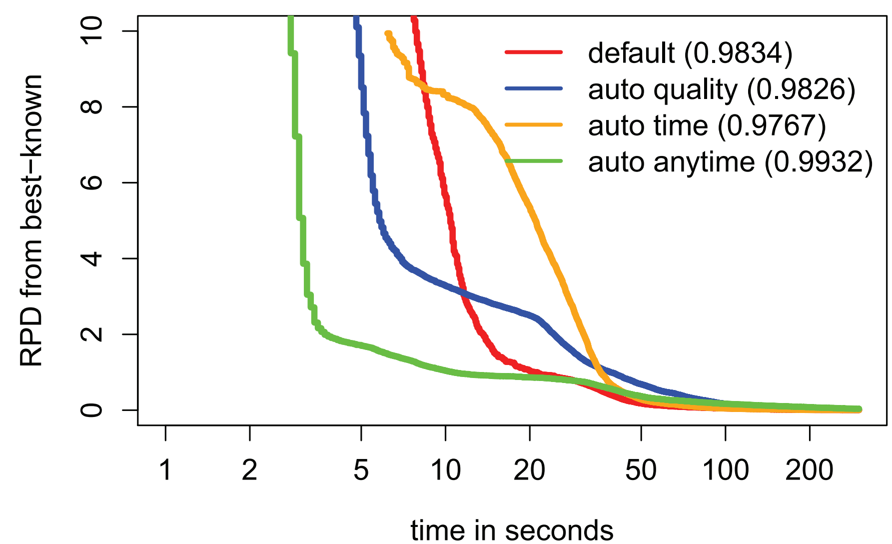
- Hundred of parameters
- Tuned by default for solving fast to *optimality*

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 GE and a grammar description (Burke et al., 2012)

Bottom-up approaches using irace?

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 numeric terminals ⇒ numerical parameters

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

becomes

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


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

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

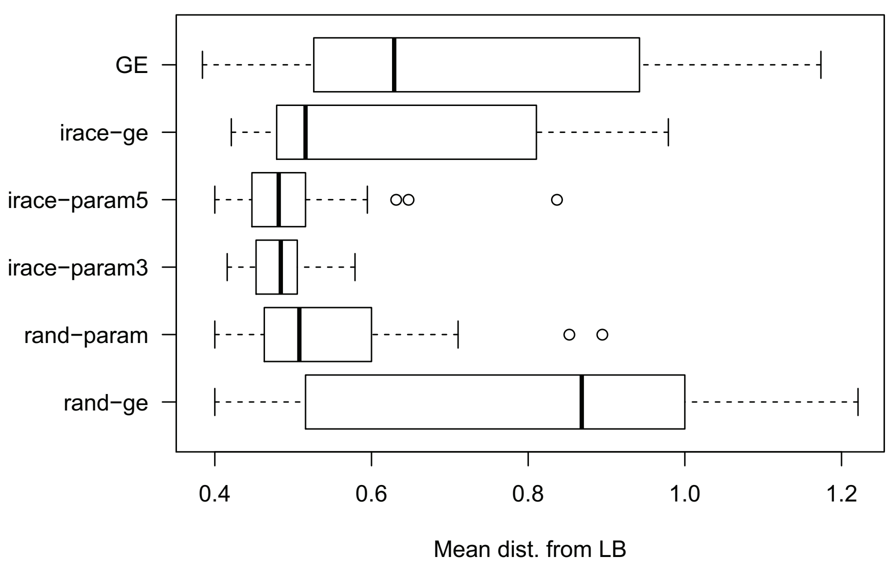
can be represented by

```
--type1   {highest, lowest}
--num1    (1, 5)
--type2   {highest, lowest, ""}
--num2    (1, 5)                   if type2 ≠ ""
--type3   {highest, lowest, ""}    if type2 ≠ ""
...       ...
```

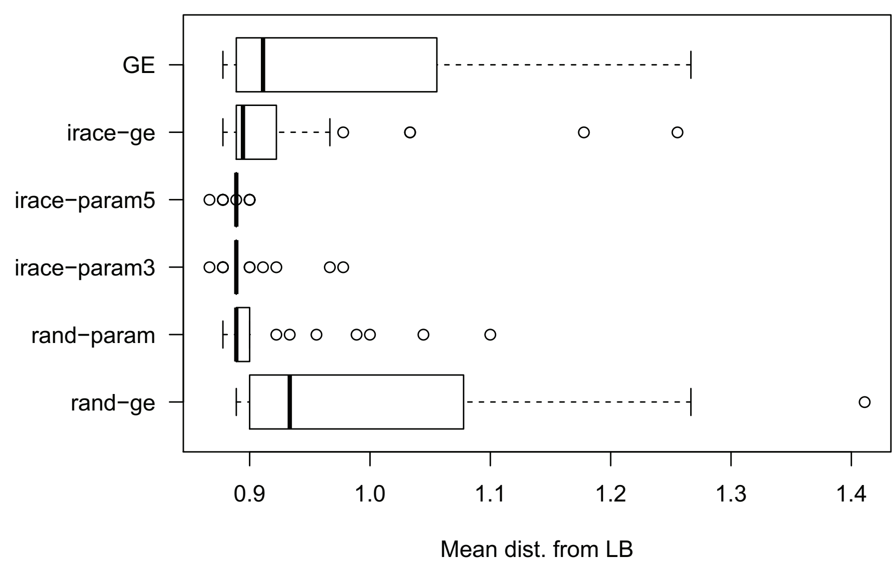

From Grammars to Parameters

Method	Details	# params
GE	(Burke et al., 2012)	30
irace-ge	codons range $[0, 5]$	30
irace-param5	max. 5 repeated rules	91
irace-param3	max. 3 repeated rules	55
rand-ge	250 random IGs, 10 instances	30
rand-param	Same but using irace-param5 rep	91

Results (Uniform1000)



Results (Scholl)



From Grammars to Parameters: 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. **From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness.** In *Learning and Intelligent Optimization, 7th International Conference, LION 7*, 2013.

✔ 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*, 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

An overview of applications of irace

irace works great for

- Complex parameter spaces:
  numerical, categorical, ordinal, subordinate (conditional)
- Large parameter spaces (up to 200 parameters)
- Heterogeneous instances
- Medium to large tuning budgets (thousands of runs)
- Individuals 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

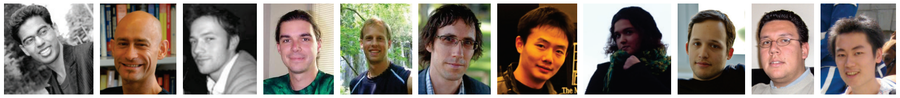
- Extremely large parameter spaces (thousands of parameters)
- Extremely heterogeneous instances
- Small tuning budgets (500 or less runs)
- Very large tuning budgets (millions of runs)
- Individuals runs require days
- Parameter tuning of decision algorithms / minimize time

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, 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
- META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium
- FRFC project "*Méthodes de recherche hybrides pour la résolution de problèmes complexes*"
- and the EU FP7 ICT Project COLOMBO, Cooperative Self-Organizing System for Low Carbon Mobility at Low Penetration Rates (agreement no. 318622)

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.

References II

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.

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.

References III

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. doi: 10.1145/2001576.2001847.

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

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.

References IV

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.

T. Liao, M. A. Montes de Oca, and T. Stützle. Computational results for an automatically tuned CMA-ES with increasing population size on the CEC'05 benchmark set. *Soft Computing*, 17(6):1031–1046, 2013.

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.

M. A. Montes de Oca, D. Aydin, and T. Stützle. An incremental particle swarm for large-scale continuous optimization problems: An example of tuning-in-the-loop (re)design of optimization algorithms. *Soft Computing*, 15(11):2233–2255, 2011. doi: 10.1007/s00500-010-0649-0.

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.

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.

References V

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.

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