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

Agenda

Part I: Automatic Configuration of Algorithms (~53.2 minutes)

Part II: Iterated Racing (irace) (~47.7 minutes)

Questions, discussion, . . . (during presentation and at end)

Coffee . . . (16:03 —)
Part I

Automatic Algorithm Configuration
(Overview)

Solving complex optimization problems

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

- **Exact (systematic search) algorithms**
  - branch&bound, branch&cut, constraint programming, . . .
  - guarantees of optimality but often time/memory consuming
  - powerful general-purpose software available

- **Approximation algorithms**
  - heuristics, local search, metaheuristics, hyperheuristics . . .
  - rarely provable guarantees but often fast and accurate
  - typically special-purpose software

*Very active research on hybrids of exact/approximate algorithms!*
Modern high-performance optimizers involve a large number of design choices and parameter settings.

- **Exact solvers**
  - Design choices: alternative models, pre-processing, variable selection, value selection, branching rules...
  - + numerical parameters
  - SCIP solver: more than 200 parameters that influence search

- **(Meta)-heuristic solvers**
  - Design choices: solution representation, operators, neighborhoods, pre-processing, strategies, ... + numerical parameters
  - Multi-objective ACO algorithms with 22 parameters (see part 2)

---

### ACO, Probabilistic solution construction

![ACO diagram](image)
ACO design choices and numerical parameters

- solution construction
  - choice of constructive procedure
  - choice of pheromone model
  - choice of heuristic information
  - numerical parameters
    - $\alpha, \beta$ influence the weight of pheromone and heuristic information, respectively
    - $q_0$ determines greediness of construction procedure
    - $m$, the number of ants

- pheromone update
  - which ants deposit pheromone and how much?
  - numerical parameters
    - $\rho$: evaporation rate
    - $\tau_0$: initial pheromone level

- local search
  - ... many more ...
Parameter types

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

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

- **numerical** parameters
  - integer or real-valued parameters
  - weighting factors, population sizes, temperature, hidden constants, ... 

Parameters may be **conditional** to specific values of other parameters

*Configuring algorithms involves setting categorical, ordinal and numerical parameters*

---

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

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

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

Human expert + trial-and-error/statistics

Towards more systematic approaches

Traditional approaches

- Trial–and–error design guided by expertise/intuition
  - prone to over-generalizations, limited exploration of design alternatives, human biases

- Guided by theoretical studies
  - often based on over-simplifications, specific assumptions, few parameters

Can we make this approach more principled and automatic?
Towards automatic (offline) algorithm configuration

Automatic algorithm configuration

- apply powerful search techniques to design algorithms
- use computation power to explore algorithm design spaces
- free human creativity for higher level tasks
- at least, assist algorithm designer in the design process

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 tuning / Algorithm configuration

- Learn best parameters before solving an instance
- Configuration done on training instances
- Performance measured over test (≠ training) instances

Online tuning / Parameter control / Reactive search

- Learn parameters while solving an instance
- No training phase
- Limited to very few crucial parameters

*Offline configuration techniques can be helpful to configure online parameter control strategies*
The (offline) algorithm configuration problem (Birattari, 2009)

A configuration instance (scenario) is defined by a tuple

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

\( \Theta \): set of potential algorithm configurations (possibly infinite)
\( \mathcal{I} \): set of instances (possibly infinite), from which instances are sampled with certain probability
\( C \): \( \Theta \times \mathcal{I} \rightarrow \mathbb{R} \): cost measure, where:
\( C(\theta, i) \): cost of configuration \( \theta \in \Theta \) on instance \( i \in \mathcal{I} \)
\( c(\theta, i) \): cost after running one time configuration \( \theta \) on instance \( i \)
\( c_\theta \): function of the cost \( C \) of a configuration \( \theta \) with respect to the distribution of the random variable \( \mathcal{I} \)

Goal: find the best configuration \( \theta^* \) such that:

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

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

**Configurator**

- **Parameters Definition**
  - name
  - type
  - possible values

- Calls with candidate configuration
- Returns solution cost

**Best configuration to be used**

---

### AC is a stochastic optimization problem

#### Decision variables
- discrete (categorical, ordinal, integer) and continuous

#### Stochasticity
- of the target algorithm
- of the problem instances

#### Typical tuning goals
- maximize solution quality within given time
- minimize run-time to decision / optimal solution

AC requires specialized methods
Approaches to configuration

- experimental design techniques
  - e.g. CALIBRA [Adenso–Díaz, Laguna, 2006], [Ridge&Kudenko, 2007], [Coy et al., 2001], [Ruiz, Stützle, 2005]
- numerical optimization techniques
  - e.g. MADS [Audet&Orban, 2006], various [Yuan et al., 2012]
- heuristic search methods
  - e.g. meta-GA [Grefenstette, 1985], ParamILS [Hutter et al., 2007, 2009], gender-based GA [Ansótegui at al., 2009], linear GP [Oltean, 2005], REVAC(++) [Eiben & students, 2007, 2009, 2010] ...
- model-based optimization approaches
  - e.g. SPO [Bartz-Beielstein et al., 2005, 2006, .. ], SMAC [Hutter et al., 2011, ..]
- sequential statistical testing
  - e.g. F-race, iterated F-race [Birattari et al, 2002, 2007, ..]

General, domain-independent methods required: (i) applicable to all variable types, (ii) multiple training instances, (iii) high performance, (iv) scalable
The racing approach

(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
- \( \ldots \) repeat until a winner is selected or until computation time expires

How to discard?

Statistical testing!

- \textbf{F-Race}: Friedman two-way analysis of variance by ranks
  + Friedman post-hoc test (Conover, 1999)

- Alternative: paired t-test with/without p-value correction (against the best)
Some (early) applications of F-race

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

2003 International time-tabling competition (Chiarandini et al., 2006)
- winning algorithm configured by F-race
- interactive injection of new configurations

F-race in stochastic optimization (Birattari et al., 2006)
- evaluate “neighbors” using F-race
  (solution cost is a random variable)
- very good performance if variance of solution cost is high

Sampling configuration

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

How to sample algorithm configurations?

- Full factorial
- Random sampling
- Iterative refinement of a sampling model
  \[ \Rightarrow \text{Iterated F-Race (I/F-Race)} \] (Balaprakash et al., 2007)
Iterative race: an illustration

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

more details, see part 2 of the tutorial

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

Smit & Eiben (2010) configured another variant of IPOP-CMAES for three different objectives
ParamILS is an iterated local search method that works in the parameter space.

**Main design choices for ParamILS**

**Parameter encoding:** only categorical parameters, numerical parameters need to be discretized.

**Initialization:** select best configuration among default and several random configurations.

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

**Perturbation:** change several randomly chosen parameters.

**Acceptance criterion:** always select the better configuration.
Main design choices for ParamILS

Evaluation of incumbent

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

Adaptive Capping

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

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

- SAT-based verification (Hutter et al., 2007a)
  - SPEAR solver with 26 parameters
    ⇒ speed-ups of up to 500 over default configuration

- Configuration of commercial MIP solvers (Hutter et al., 2010)
  - CPLEX (63 parameters), Gurobi (25 parameters) and lp_solve (47 parameters) for various instance distributions of MIP encoded optimization problems
  - speed-ups ranged between a factor of 1 (none) to 153

Mixed integer programming (MIP) solvers (Hutter, Hoos, Leyton-Brown, and Stützle, 2009; Hutter, Hoos, and Leyton-Brown, 2010)

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

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

FocusedILS, 10 runs, 2 CPU days, 63 parameters
Gender-based genetic algorithm

(Ansótegui et al., 2009)

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

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

Promising results

Relevance Estimation and Value Calibration (REVAC)

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

REVAC is an EDA for tuning numerical parameters

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

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

MADS / OPAL
- Mesh-adaptive direct search applied to parameter tuning of other direct-search methods (Audet & Orban, 2006)
- later extension to OPAL (OPtimization of ALgorithms) framework (Audet et al., 2010)
- Limited experiments

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

Model-based Approaches (SPOT, SMAC)

Idea: Use surrogate models to predict performance

Algorithmic scheme
1: generate and evaluate initial set of configurations $\Theta_0$
2: choose best-so-far configuration $\theta^* \in \Theta_0$
3: while tuning budget available do
4: learn surrogate model $M: \Theta \mapsto R$
5: use model $M$ to generate promising configurations $\Theta_p$
6: evaluate configurations in $\Theta_p$
7: $\Theta_0 := \Theta_0 \cup \Theta_p$
8: update $\theta^* \in \Theta_0$
9: output: $\theta^*$
Sequential parameter optimization (SPO) toolbox
(Bartz-Beielstein et al., 2005, 2010b)

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

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

Sequential model-based algorithm configuration (SMAC)
(Hutter et al., 2011)

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

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

- Only numerical parameters:
  - Homogeneous instances ⇒ numerical optimizers, e.g., BOBYQA (few parameters), CMA-ES (many)
  - Expensive homogeneous instances ⇒ SMAC, SPO
  - Heterogeneous instances ⇒ numerical optimizers + (racing or post-selection)

- Categorical and numerical parameters
  - Tuning goal is time ⇒ SMAC
  - Tuning goal is quality ⇒ IRACE

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

AClib: A Benchmark Library for Algorithm Configuration


http://www.aclib.net/

- Standard benchmark for experimenting with configurators
- 182 heterogeneous scenarios
- SAT, MIP, ASP, time-tabling, TSP, multi-objective, machine learning
- Extensible ⇒ new scenarios welcome!
Why automatic algorithm configuration?

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

**Scaling to expensive instances**

*What if my problem instances are too difficult/large?*

- Cloud computing / Large computing clusters


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


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

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

✓ it can be done (Hutter et al., 2009) but …
✘ it is costly and iterating further leads to diminishing returns

Towards a paradigm shift in algorithm design

Thomas Stützle and Manuel López-Ibáñez Automatic (Offline) Configuration of Algorithms
What is Iterated Racing and irace?

- **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)

- **A software package** implementing the variant proposed by López-Ibáñez, Dubois-Lacoste, Stützle, and Birattari (2011)
Iterated Racing

Sampling new configurations according to a probability distribution

Selecting the best configurations from the newly sampled ones by means of racing

Updating the probability distribution in order to bias the sampling towards the best configurations
Iterated Racing: Sampling distributions

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

$\Rightarrow$ *Truncated normal distribution*

$$\mathcal{N}(\mu^z_d, \sigma^i_d) \in [\bar{x}_d, \underline{x}_d]$$

$\mu^z_d =$ value of parameter $d$ in elite configuration $z$

$\sigma^i_d =$ decreases with the number of iterations

**Categorical parameter** $X_d \in \{x_1, x_2, \ldots, x_{n_d}\}$

$\Rightarrow$ *Discrete probability distribution*

$$\Pr\{X_d = x_j\} = \begin{array}{cccc} x_1 & x_2 & \ldots & x_{n_d} \\ 0.1 & 0.3 & \ldots & 0.4 \end{array}$$

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

---

Iterated Racing: Soft-restart

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

1. **Compute distance** between sampled candidate configurations

2. If distance is zero, **soft-restart** the sampling distribution of the parents

   **Numerical parameters** : $\sigma^i_d$ is “brought back” to its value at two iterations earlier, approx. $\sigma^{i-2}_d$

   **Categorical parameters** : “smoothing” of probabilities, increase low values, decrease high values.

3. **Resample**
Iterated Racing: Other features

1. Initial configurations
   - Seed irace with the default configuration
     or configurations known to be good for other problems

2. Parallel evaluation
   - Configurations within a race can be evaluated in parallel
     using MPI, multiple cores, Grid Engine / qsub clusters

3. Forbidden configurations (new in 1.05)
   - popsize < 5 & LS == "SA"

4. Recovery file (new in 1.05)
   - allows resuming a previous irace run

The irace Package


- 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

Instances
Parameter space
Configuration of irace

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

Thomas Stützle and Manuel López-Ibáñez
Automatic (Offline) Configuration of Algorithms

The irace Package: Instances

- **TSP instances**
  
  
  $ \text{dir Instances/} \\
  3000-01.tsp 3000-02.tsp 3000-03.tsp ...$

- **Continuous functions**
  
  $ \text{cat instances.txt} \\
  \text{function=1 dimension=100} \\
  \text{function=2 dimension=100} \\
  ...$

- **Parameters for an instance generator**
  
  $ \text{cat instances.txt} \\
  \text{I1 --size 100 --num-clusters 10 --sym yes --seed 1} \\
  \text{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)

```bash
$ cat parameters.txt
```

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Label/switch</th>
<th>Type</th>
<th>Domain</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS</td>
<td>--localsearch</td>
<td>c</td>
<td>{SA, TS, II}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rate</td>
<td>--rate=</td>
<td>o</td>
<td>{low, med, high}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>population</td>
<td>--pop</td>
<td>i</td>
<td>(1, 100)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>temp</td>
<td>--temp</td>
<td>r</td>
<td>(0.5, 1)</td>
<td>LS == &quot;SA&quot;</td>
<td></td>
</tr>
</tbody>
</table>

- 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

```bash
$ cat parameters.txt
```

<table>
<thead>
<tr>
<th>#</th>
<th>name</th>
<th>switch</th>
<th>type</th>
<th>values</th>
<th>conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>algorithm</td>
<td>&quot;=&quot;</td>
<td>c</td>
<td>(as,mmas,eas,ras,acs)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>localsearch</td>
<td>&quot;--localsearch&quot;</td>
<td>c</td>
<td>(0, 1, 2, 3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>&quot;--alpha &quot;</td>
<td>r</td>
<td>(0.00, 5.00)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>beta</td>
<td>&quot;--beta &quot;</td>
<td>r</td>
<td>(0.00, 10.00)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rho</td>
<td>&quot;--rho &quot;</td>
<td>r</td>
<td>(0.01, 1.00)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ants</td>
<td>&quot;--ants &quot;</td>
<td>i</td>
<td>(5, 100)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>q0</td>
<td>&quot;--q0 &quot;</td>
<td>r</td>
<td>(0.0, 1.0)</td>
<td>algorithm == &quot;acs&quot;</td>
</tr>
<tr>
<td></td>
<td>rasrank</td>
<td>&quot;--rasranks &quot;</td>
<td>i</td>
<td>(1, 100)</td>
<td>algorithm == &quot;ras&quot;</td>
</tr>
<tr>
<td></td>
<td>elitistsants</td>
<td>&quot;--elitistsants &quot;</td>
<td>i</td>
<td>(1, 750)</td>
<td>algorithm == &quot;eas&quot;</td>
</tr>
<tr>
<td></td>
<td>nnls</td>
<td>&quot;--nnls &quot;</td>
<td>i</td>
<td>(5, 50)</td>
<td>localsearch %in% c(1,2,3)</td>
</tr>
<tr>
<td></td>
<td>dlb</td>
<td>&quot;--dlb &quot;</td>
<td>c</td>
<td>(0, 1)</td>
<td>localsearch %in% c(1,2,3)</td>
</tr>
</tbody>
</table>
```
#!/bin/bash

INSTANCE=$1
CANDIDATENUM=$2
CAND_PARAMS=$*
STDOUT="c$\{CANDIDATENUM\}.stdout"
FIXED_PARAMS=" --time 20 --tries 1 --quiet "
acotsp $FIXED_PARAMS -i $INSTANCE $CAND_PARAMS 1> $STDOUT

COST=$(grep -oE 'Best \[-+0-9.e\]+' $STDOUT | cut -d ' ' -f2)

if ! [[ "$\{COST\}" =~ ^[-+0-9.e]+$ ]]; then
  error "$\{STDOUT\}: Output is not a number"
fi

echo "$\{COST\}"
exit 0
```

```
$ cat hook-run

GOOD: ACOTSP

```

```
$ cat tune-conf

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

✓ Good to go:

$ mkdir acotsp-arena
$ irace

```

```
Example: ACOTSP

$ cat tune-conf

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

✓ Good to go:

$ mkdir acotsp-arena
$ irace

```

```
Example: ACOTSP

$ cat hook-run

GOOD: ACOTSP

```

```
```

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

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

![Graph showing Pareto front]

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

Results: Multi-objective components

<table>
<thead>
<tr>
<th>Method</th>
<th>euclidAB100.tsp</th>
<th>euclidAB300.tsp</th>
<th>euclidAB500.tsp</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOAQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BicriterionAnt (1 col)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BicriterionAnt (3 col)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPETants</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PACO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mACO−1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mACO−2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mACO−3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mACO−4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOACO (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOACO (2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOACO (3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOACO (4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOACO (5)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We propose a new MOACO algorithm that...

We propose an approach to automatically design MOACO algorithms:

- Synthesize state-of-the-art knowledge into a flexible MOACO framework
- 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, Satenstein
  (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
“Anytime” algorithms aim to produce as high quality results as possible, independent of the computation time allowed.

Automatically Improving the Anytime Behavior

Hypervolume measure $\approx$ Anytime behaviour

Automatically Improving the Anytime Behavior

Scenario #1
Online parameter adaptation to make an algorithm more robust to different termination criteria
- 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 \textit{optimality}

\textbf{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 (\textit{maxExperiments}): 5 000 runs
Automatically Improving the Anytime Behavior

Example #4

Tuning in algorithm engineering:
Tuning in the loop
re-design of an incremental PSO algorithm for large-scale continuous optimization

- steps: (1) local search, (2) call and control strategy of LS, (3) PSO rules, (4) bound constraint handling, (5) stagnation handling, (6) 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

Thomas Stützle and Manuel López-Ibáñez

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

Tune known algorithms; example IPOP-CMAES

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

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

Average Errors–50D–100runs
- Win 13
- Lose 4
- Draw 8
Comparison of continuous optimizers
[Liao, Molina, Stützle, 2012]

### Solvers
- IPOP-CMAES (evolution strategy)
- MA-LSch-CMA (memetic algorithm)
- UACOR (ant colony optimization)
- IPSOLS (particle swarm optimization)
- IABCLS (artificial bee colony)
- SaDE (self-adaptive differential evolution)
- MOS (differential evolution, local search hybrid)

### Benchmark sets
- CEC’05 benchmark set (25 functions, scalable)
- Soft Computing special issue (19 functions, scalable)

**Ranking default vs. tuned, example CEC’05**

![Bar charts comparing default vs. tuned configurations for CEC'05 benchmark set](chart.png)

- **CEC'05_50D_Default**
  - MA-LSch-CMA: 2.78
  - IPOP-CMA-ES: 2.96
  - UACOR: 3.54
  - SaDE: 4.55
  - IABCLS: 4.53
  - IPSOLS: 4.78
  - MOS: 4.92

- **CEC'05_50D_Tuned**
  - MA-LSch-CMA: 2.48
  - IPOP-CMA-ES: 3.26
  - UACOR: 3.8
  - SaDE: 4.53
  - IABCLS: 4.7
  - IPSOLS: 4.82
  - MOS: 4.92
Ranking on SOCO vs. CEC’05 benchmark sets

Configuring continuous function classes
(Yuan et al., 2012)

- family of continuous functions
- example, Rastrigin

\[ nA + \sum_{i=1}^{n} (x_i^2 - A \cos(\omega x_i)) \]

- generate different values for \( A, \omega, n \) such that some properties are maintained (e.g. landscape features)
- configure continuous algorithms for it
- some extensions on the way …
From Grammars to Parameters: 
How to use irace to design algorithms from a grammar description?

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

Top-down approaches
- Flexible frameworks:
  - SATenstein (KhudaBukhsh et al., 2009)
  - MOACO framework (López-Ibáñez and Stützle, 2012)
  - MIP solvers: CPLEX, SCIP
- Automatic configuration tools:
  - ParamILS (Hutter et al., 2009)
  - irace (Birattari et al., 2010; López-Ibáñez et al., 2011)

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

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

Thomas Stützle and Manuel López-Ibáñez  Automatic (Offline) Configuration of Algorithms
From Grammars to Parameters: How to use irace to design algorithms from a grammar description? See talk on Monday by Franco Mascia

Summary

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


- Not limited to IG!

Automatic design of hybrid SLS algorithms
[Marmion, Mascia, López-Ibáñez, Stütze, 2013]

Approach
- decompose single-point SLS methods into components
- derive generalized metaheuristic structure
- component-wise implementation of metaheuristic part

Implementation
- present possible algorithm compositions by a grammar
- instantiate grammar using a parametric representation
  - allows use of standard automatic configuration tools
  - shows good performance when compared to, e.g., grammatical evolution [Mascia, López-Ibáñez, Dubois-Lacoste, Stütze, 2014]

General Local Search Structure: ILS

\[
\begin{align*}
s_0 & := \text{initSolution} \\
 s^* & := \text{ls}(s_0) \\
 \text{repeat} \\
 & \quad s' := \text{perturb}(s^*, \text{history}) \\
 & \quad s^{*'} := \text{ls}(s') \\
 & \quad s^* := \text{accept}(s^*, s^{*'}, \text{history}) \\
 \text{until} & \quad \text{termination criterion met}
\end{align*}
\]

- many SLS methods instantiable from this structure
- abilities
  - hybridization
  - recursion
  - problem specific implementation at low-level
<algorithm> ::= <initialization> <ils>
<initialization> ::= random | <pbs_initialization>
<ils> ::= ILS(<perturb>, <ls>, <accept>, <stop>)

<perturb> ::= none | <initialization> | <pbs_perturb>
<ils> ::= <ils> | <descent> | <sa> | <rii> | <pii> | <vns> | <ig> | <pbs_ls>
<accept> ::= alwaysAccept | improvingAccept <comparator>
| prob(<value_prob_accept>) | probRandom | <metropolis> | threshold(<value_threshold_accept>) | <pbs_accept>
<descent> ::= bestDescent(<comparator>, <stop>)
| firstImprDescent(<comparator>, <stop>)
<sa> ::= ILS(<pbs_move>, no_ls, <metropolis>, <stop>)
<rii> ::= ILS(<pbs_move>, no_ls, probRandom, <stop>)
<pii> ::= ILS(<pbs_move>, no_ls, prob(<value_prob_accept>), <stop>)
<vns> ::= ILS(<pbs_variable_move>, firstImprDescent(improvingStrictly),
    improvingAccept(improvingStrictly), <stop>)
<ig> ::= ILS(<deconst-construct_perturb>, <ls>, <accept>, <stop>)

<comparator> ::= improvingStrictly | improving
<value_prob_accept> ::= [0, 1]
<value_threshold_accept> ::= [0, 1]
<metropolis> ::= metropolisAccept(<init_temperature>, <final_temperature>,
    <decreasing_temperature_ratio>, <span>)

<init_temperature> ::= {1, 2,..., 10000}
<final_temperature> ::= {1, 2,..., 100}
<decreasing_temperature_ratio> ::= [0, 1]
<span> ::= {1, 2,..., 10000}
Grammar

<algorithm> ::= <initialization> <ils>
<initialization> ::= random | <pbs_initialization>
<ils> ::= ILS(<perturb>, <ls>, <accept>, <stop>)
<perturb> ::= none | <initialization> | <pbs_perturb>
<ls> ::= <ils> | <descent> | <sa> | <rii> | <pii> | <vns> | <ig> | <pbs_ls>
<accept> ::= alwaysAccept | improvingAccept <comparator>
| prob(<value_prob_accept>) | probRandom | <metropolis>
| threshold(<value_threshold_accept>) | <pbs_accept>
<descent> ::= bestDescent(<comparator>, <stop>)
| firstImprDescent(<comparator>, <stop>)
<sa> ::= ILS(<pbs_move>, no_ls, <metropolis>, <stop>)
<rii> ::= ILS(<pbs_move>, no_ls, probRandom, <stop>)
<pii> ::= ILS(<pbs_move>, no_ls, prob(<value_prob_accept>), <stop>)
<vns> ::= ILS(<pbs_variable_move>, firstImprDescent(improvingStrictly),
improvingAccept(improvingStrictly), <stop>)
<ig> ::= ILS(<deconst-construct_perturb>, <ls>, <accept>, <stop>)
<comparator> ::= improvingStrictly | improving
[value_prob_accept] ::= [0, 1]
[value_threshold_accept] ::= [0, 1]
[metropolis] ::= metropolisAccept(<init_temperature>, <final_temperature>,
<decreasing_temperature_ratio>, <span>)
[init_temperature] ::= {1, 2, ..., 10000}
[final_temperature] ::= {1, 2, ..., 100}
[decreasing_temperature_ratio] ::= [0, 1]
[span] ::= {1, 2, ..., 10000}

System overview

problem-specific grammar

parameter instantiation
grammar2code

problem-independent grammar

performance measured and used to generate new parameter instantations

compiled and executed on test instances
Flow-shop problem with weighted tardiness

- Automatic configuration:
  - 1, 2 or 3 levels of recursion (r)
  - 80, 127, and 174 parameters, respectively
  - budget: \( r \times 10,000 \) trials each of 30 seconds

\[
\begin{array}{c|c|c|c}
\text{Algorithms} & \text{ALS1} & \text{ALS2} & \text{ALS3} \\
\hline
\text{Fitness value} & 24,200 & 24,600 & 25,000 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\text{Algorithms} & \text{ALS1} & \text{ALS2} & \text{ALS3} \\
\hline
\text{Fitness value} & 26,600 & 27,000 & 27,400 \\
\end{array}
\]

dgreen results are competitive or superior to state-of-the-art

An overview of applications of irace

Done already

- Parameter tuning
  - single-objective optimization metaheuristics
  - MIP solvers (SCIP) with \( > 200 \) parameters.
  - multi-objective optimization metaheuristics
  - anytime algorithms (improve time-quality trade-offs)

- Automatic algorithm design
  - From a flexible framework of algorithm components
  - From a grammar description

- Machine learning
  - Automatic model selection for high-dimensional survival analysis (Lang et al., 2014)
  - Hyperparameter tuning (mlr R package, Bischl et al.)
An overview of applications of irace

<table>
<thead>
<tr>
<th>irace (and others) works great for</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Complex parameter spaces:</td>
</tr>
<tr>
<td>numerical, categorical, ordinal,</td>
</tr>
<tr>
<td>subordinate (conditional)</td>
</tr>
<tr>
<td>- Large parameter spaces (few</td>
</tr>
<tr>
<td>hundred parameters)</td>
</tr>
<tr>
<td>- Heterogeneous instances</td>
</tr>
<tr>
<td>- Medium to large tuning budgets</td>
</tr>
<tr>
<td>(thousands of runs)</td>
</tr>
<tr>
<td>- Individual runs require from</td>
</tr>
<tr>
<td>seconds to hours</td>
</tr>
<tr>
<td>- Multi-core CPUs, MPI, Grid-Engine</td>
</tr>
<tr>
<td>clusters</td>
</tr>
</tbody>
</table>

What we haven’t deal with yet

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

We are looking for interesting benchmarks / applications!

Talk to us!
Acknowledgments

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

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

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

- European Research Council under the European Union’s Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n° 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 III


References IV


