Automatic (Offline) Configuration of Algorithms

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

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

Part I

Automatic Algorithm Configuration (Overview)

Outline of Part I: Automatic Algorithm Configuration (Overview)

1. Context
2. Automatic Algorithm Configuration
3. The Configuration Problem
4. Methods for Automatic Algorithm Configuration

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

**Today’s 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, neighborhood, 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**

[Diagram showing probabilistic solution construction with ants and pheromone trails]

**ACO design choices and numerical parameters**

- **Solution construction**
  - Choice of pheromone model
  - Choice of heuristic information
  - Choice of constructive procedure
  - Numerical parameters
    - $\alpha, \beta$: influence the weight of pheromone and heuristic information, respectively
    - $\rho_0$: 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 ...
Usual parameter types

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

- **ordinal** parameters
  - neighborhood sizes, tightness of lower bounds, …

- **numerical** parameters
  - weighting factors, population sizes, temperature, hidden constants, …
  - integer or real-valued parameters
  - numerical parameters may be conditional to specific values of categorical or ordinal parameters

**Remark:** With configuration we refer to the task of 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

Towards automatic (offline) algorithm configuration

- **Traditional approaches**
  - trial–and–error design guided by expertise/intuition
    - prone to over-generalizations, implicit independence assumptions, limited exploration of design alternatives
  - indications through theoretical studies
    - often based on over-simplifications, specific assumptions, few parameters

- **Automated 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
Why automatic algorithm configuration?

- Improvement over manual, ad-hoc methods for tuning
- Reduction of development time and human intervention
- Increase number of considerable degrees of freedom
- Empirical studies, comparisons of algorithms
- Support for end users of algorithms

Towards a shift of paradigm in algorithm design

Offline configuration

- 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
  - Online control strategies have parameters

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

\[ \langle \Theta, I, P_I, P_C, t, C, T \rangle \]

- \( \Theta \) 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 \( I \).
- \( t: I \rightarrow \mathbb{R} \) 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 \( t(i) \).

- \( C \subset \mathbb{R} \) 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 \( \theta \) on instance \( i \).
- \( C(\theta) = C(\theta|\Theta, I, P_I, P_C, t) \) is the criterion that needs to be optimized with respect to \( \theta \).
- \( T \) 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 \( \hat{\theta} \), i.e.,

\[
\hat{\theta} = \arg \min_{\theta \in \Theta} C(\theta)
\]  

(1)

- if we consider the expected value, we have

\[
C(\theta) = E_{l,c}[c] = \int c \, dP_{C}(c|\theta, i) \, dP_{l}(i).
\]  

(2)

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

Estimation of expected cost (a remark)

Given

- \( n \) runs for estimating expected cost of configuration \( \theta \)
- a large number of instances

Question

- how many runs on how many instances to minimize variance of estimate?

Answer

- one run on each of \( n \) instances (Birattari, 2004, 2009)

Approaches to configuration

- numerical optimization techniques
  - e.g. MADS (Audet & Orban, 2006), various (Yuan et al., 2012)
- heuristic search methods
  - e.g. ParamILS (Hutter et al., 2007b, 2009), gender-based GA (Ansotegui et al., 2009), linear GP (Oltean, 2005), meta-GA (Grefenstette, 1986), 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, 2010b), 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

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
The racing approach

- start with a set of initial candidates
- consider a stream of instances
- sequentially evaluate candidates
- discard inferior candidates as sufficient evidence is gathered against them
- ... repeat until a winner is selected or until computation time expires

The F-Race algorithm

Statistical testing
- family-wise tests for differences among configurations
  - Friedman two-way analysis of variance by ranks
- 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():
- apply race
- modify the distribution
- sample configurations with selection probability

more details, see part 2 of the tutorial

Example application: configuring 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

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

Tuning in-the-loop Design of Continuous Optimizers
(Montes de Oca et al., 2011)

- Soft Computing special issue on large-scale continuous
  optimization: 19 functions scaled from 50 to 1000 dimensions
- re-optimization of existing incremental Particle Swarm
  Optimization algorithm with local search
- bottom-up algorithm re-engineering process across 7 steps
- Iterated F-race systematically used at each step to specify
  up to 10 parameters
- configuration is done on small dimensional functions (10
  dimensions) across all 19 benchmark functions
- final result: state-of-the-art algorithm for large-scale
  continuous function optimization (tested up to 1000
  dimensions)

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 to be discretized

Initialization
- Select best configuration among default configuration 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

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

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 the CPLEX (76 parameters), Gurobi (25 parameters) and lp solve (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

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

Surrogate Model-assisted approaches

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

Algorithmic scheme
1: Generate initial set of configurations \( \Theta_0 \); evaluate \( \Theta_0 \), choose best-so-far configuration \( \theta^* \), \( i := 1, \Theta_i := \Theta_0 \)
2: while computation budget available do
3: Learn surrogate model \( \mathcal{M} : \Theta \mapsto R \) based on \( \Theta_i \)
4: Use model \( \mathcal{M} \) to select promising configurations \( \Theta_p \)
5: Evaluate configurations in \( \Theta_p \), update \( \theta^* \), \( \Theta_i := \Theta_i \cup \Theta_p \)
6: \( i := i + 1 \)
7: Output: \( \theta^* \)

Sequential parameter optimization (SPO) framework (Bartz-Beielstein et al., 2005, 2010b)

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

Main design decisions
- uses in most variants Gaussian stochastic processes as model \( \mathcal{M} \)
- promising candidates are selected based on expected improvement criterion
- intensification mechanism increases number of evaluations of best-so-far configuration \( \theta^* \)

Practicalities
- SPO is implemented in the comprehensive SPOT R-package
- most applications, however, to numerical parameter tuning, single instances
SMAC extends surrogate model-assisted configuration to complex algorithm configuration tasks and across multiple instances.

Main design decisions:
- Uses random forests as model $M$ 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).
- Uses instance features to improve the performance predictions.
- Intensification mechanism similar to that of FocusedILS.
- Further extensions through introduction of capping.

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).
- 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.
- New post-selection mechanism appears promising (Yuan et al., 2012, 2013).

Part II: Iterated Racing (irace)

1. What is Iterated Racing and irace?
2. The irace Package
3. Example: ACOTSP
4. A more complex example: MOACO framework
5. Automatically Improving the Anytime Behavior
6. From Grammars to Parameters
7. An overview of applications of irace
8. Questions
What is Iterated Racing and irace?

Iterated Racing (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

Require:
- Training instances: \( \{ l_1, l_2, \ldots \} \sim I \)
- Parameter space: \( X \)
- Cost measure: \( C: \Theta \times I \rightarrow \mathbb{R} \)
- Tuning budget: \( B \)

1: \( \Theta_1 := \text{SampleUniform}(X) \)
2: \( \Theta_{\text{elite}} := \text{Race}(\Theta_1, B_1) \)
3: \( i := 2 \)
4: while \( B_{\text{used}} \leq B \) do
5: \( \Theta_{\text{new}} := \text{UpdateAndSample}(X, \Theta_{\text{elite}}) \)
6: \( \Theta_i := \Theta_{\text{new}} \cup \Theta_{\text{elite}} \)
7: \( \Theta_{\text{elite}} := \text{Race}(\Theta_i, B_i) \)
8: \( i := i + 1 \)
9: Output: \( \Theta_{\text{elite}} \)

Iterated Racing: Sampling distributions

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

⇒ Truncated normal distribution

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

\( \mu_d^z \) = value of parameter \( d \) in elite configuration \( z \)

\( \sigma_d^z \) = decreases with the number of iterations

Categorical parameter \( X_d \in \{ x_1, x_2, \ldots, x_{n_d} \} \)

⇒ Discrete probability distribution

\[ \text{Pr}^z \{ X_d = x_j \} = \begin{bmatrix} 0.1 & 0.3 & \ldots & 0.4 \end{bmatrix} \]

- Updated by increasing probability of parameter value in elite configuration
- Other probabilities are reduced
Iterated Racing: Soft-restart

- irace may converge too fast
  ⇒ the same configurations are sampled again and again
- Soft-restart

Compute distance between sampled candidate configurations

- 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: \( \sigma_d^i \) is “brought back” to its value at two iterations earlier, approx. \( \sigma_d^{i-2} \)

Resample

---

Iterated Racing: Other features

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

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

---

The irace Package

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

Instances Parameter space Configuration of irace

hookRun calls with \( i, \theta \) returns \( c(i, \theta) \)
The irace Package:Instances

- TSP instances
  $dir \text{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} \\
  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

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

The irace Package: Parameter space

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

$\text{cat parameters.txt} \\
\# Name \quad \text{Label/switch} \quad \text{Type} \quad \text{Domain} \quad \text{Condition} \\
\text{LS} \quad "--localsearch " \quad \text{c} \quad \{\text{SA, TS, II}\} \\
\text{rate} \quad "--rate=" \quad \text{o} \quad \{\text{low, med, high}\} \\
\text{population} \quad "--pop " \quad \text{i} \quad \{1, 100\} \quad | \quad \text{LS} == "\text{SA}" \\
\text{temp} \quad "--temp " \quad \text{r} \quad \{0.5, 1\} \\

- For real parameters, number of decimal places is controlled by option \text{digits} (--digits)

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

The irace Package: Options

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

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

The irace Package: hook-run

- A script/program that calls the software to be tuned:
  ./\text{hook-run instance candidate-number candidate-parameters} ...

- An R function:
  \text{hook.run} <- \text{function(instance, candidate, extra.params = NULL, config = list())} \\
  \{ \\
  \text{...} \\
  \} \\

\text{Flexibility: If there is something you cannot tune, let us know!}

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

```
$ 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"
elitists   "--elitists " i         (1, 750) | algorithm == "eas"
nlsls      "--nlsls "   i         (5, 50) | localsearch in c(1,2,3)
      "--dlb "   c         (0, 1) | localsearch in c(1,2,3)

$ 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

```r
$ 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 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

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, ...)
- 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

Results: Multi-objective components

<table>
<thead>
<tr>
<th>Instance</th>
<th>MOACO (1)</th>
<th>MOACO (2)</th>
<th>MOACO (3)</th>
<th>MOACO (4)</th>
<th>MOACO (5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>euclidAB100.tsp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>euclidAB300.tsp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>euclidAB500.tsp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary

- 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
- Another example:
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.

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 optimality

Hypervolume measure ≈ Anytime behaviour

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

![Graph showing RPD from best-known for different configurations](image)

Example #4

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

**Top-down approaches**
- Flexible frameworks:
  - **SATenstein** (KhudaBukhsh et al., 2009)
  - **MOACO framework** (López-Ibáñez and Stützle, 2012)
- 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**


<code>program</code> ::= <choosebins><br>  remove_pieces_from_bins()<br>  <repack><br><choosebins> ::= <type> | <type> <choosebins><br><type> ::= highest_filled(<num>, <ignore>, <remove>)<br>  | lowest_filled(<num>, <ignore>, <remove>)<br>  | random_bins(<num>, <ignore>, <remove>)<br>  | gap_lessthan(<num>, <threshold>, <ignore>, <remove>)<br>  | num_of_pieces(<num>, <numpieces>, <ignore>, <remove>)<br><num> ::= 2 | 5 | 10 | 20 | 50<br><threshold> ::= average | minimum | maximum<br><numpieces> ::= 1 | 2 | 3 | 4 | 5 | 6<br><ignore> ::= 0.995 | 0.997 | 0.999 | 1.0 | 1.1<br><remove> ::= ALL | ONE<br><repack> ::= best_fit | worst_fit | first_fit

**Parametric representation**

Grammar ⇒ Parameter space

---

**GE representation**

codons = [3, 5, 1, 2, 7, 4]<br>
1. Start at <code>program</code><br>2. Expand until rule with alternatives<br>3. Compute (3 mod 2) + 1 = 2 ⇒ <code>type</code> <choosebins><br>4. Compute (5 mod 5) + 1 = 1 ⇒ highest_filled(<num>, )<br>5. ... until complete expansion or maximum number of wrappings<br>

<code>program</code> ::= highest_filled(<num>, <ignore>)<br>  <choosebins><br>  remove_pieces_from_bins()<br>  <repack><br><num> ::= 2 | 5 | 10 | 20 | 50

---

**Parametric representation**

Grammar ⇒ Parameter space

- Rules without alternatives ⇒ no parameter

---

Grammar ⇒ Parameter space
Parametric representation

Grammar ⇒ Parameter space?

- Rules with numeric terminals ⇒ numerical parameters

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

becomes

--num [0, 100]

- Rules with alternative choices ⇒ categorical parameters

\[
?type> ::= \text{highest\_filled}(<\text{num}>), \text{lowest\_filled}(<\text{num}>), \text{random\_bins}(<\text{num}>)
\]

becomes

--type \{highest\_filled, lowest\_filled, random\_bins\}

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

\[
\text{choosebins} ::= \text{type} | \text{type} \text{ choosebins}
\]

\[
\text{type} ::= \text{highest}(<\text{num}>), \text{lowest}(<\text{num}>)
\]

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Details</th>
<th># params</th>
</tr>
</thead>
<tbody>
<tr>
<td>GE</td>
<td>(Burke et al., 2012)</td>
<td>30</td>
</tr>
<tr>
<td>irace-ge</td>
<td>codons range [0, 5]</td>
<td>30</td>
</tr>
<tr>
<td>irace-param5</td>
<td>max. 5 repeated rules</td>
<td>91</td>
</tr>
<tr>
<td>irace-param3</td>
<td>max. 3 repeated rules</td>
<td>55</td>
</tr>
<tr>
<td>rand-ge</td>
<td>250 random IGs, 10 instances</td>
<td>30</td>
</tr>
<tr>
<td>rand-param</td>
<td>Same but using irace-param5 rep</td>
<td>91</td>
</tr>
</tbody>
</table>
Results (Uniform1000)

Results (Scholl)

From Grammars to Parameters: Summary

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


✓ Not limited to IG!


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

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!

Questions

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

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, Franco Mascia, Marco Montes de Oca, Zhi Yuan.

The research leading to the results presented in this paper has received funding from diverse projects:
- the European Research Council under the European Union’s Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n° 246939
- the META-X project, an Action de Recherche Concertée funded by the Scientific Research Directorate of the French Community of Belgium
- the 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.
References I


References II


References III


References IV


