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!

Design choices and parameters everywhere

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

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

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

**Offline configuration**

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
### Experimental Design, ANOVA
- CALIBRA [Adenso-Díaz & Laguna, 2006]
- Others [Coy et al., 2001; Ridge & Kudenko, 2007; Ruiz & Maroto, 2005]

### Numerical Optimization
- MADS [Audet & Orban, 2006], CMA-ES, BOBYQA [Yuan et al., 2012]

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

### Model-Based
- SPO [Bartz-Beielstein et al., 2005, 2010b], SMAC [Hutter et al., 2011]

### Sequential Statistical Testing
- F-race, iterated F-race [Balaprakash et al., 2007; Birattari et al., 2002], irace [López-Ibáñez et al., 2011]

---

**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

---

**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
The danger of over-tuning

![Graph showing comparison of BasicILS and 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]

Numerical optimization techniques

MADS / OPAL
- Mesh-adaptive direct search applied to parameter tuning of other direct-search methods [Audet & Orban, 2006]
- later extension to OP (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

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 Ip_solve (47 parameters) for various instance distributions of MIP encoded optimization problems
  - speed-ups ranged between a factor of 1 (none) to 153

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

**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

**How to discard?**

- **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)
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]

### 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

### 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]
  - Elitist variant proposed by López-Ibáñez, Dubois-Lacoste, Pérez Cáceres, Stützle, and Birattari [2016]
- A software package implementing the latest variants.
Iterated Racing: Sampling distributions

**Numerical parameter** \( X_d \in [x_d, x_d] \)

⇒ *Truncated normal distribution*

\[
N(\mu_d, \sigma_d^j) \in [x_d, x_d]
\]

- \( \mu_d \): value of parameter \( d \) in elite configuration \( z \)
- \( \sigma_d^j \): decreases with the number of iterations

**Categorical parameter** \( X_d \in \{x_1, x_2, \ldots, x_n_d\} \)

⇒ *Discrete probability distribution*

\[
\Pr\{X_d = x_j\} = \begin{bmatrix}
0.1 & 0.3 & \cdots & 0.4
\end{bmatrix}
\]

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

---

Iterated Racing: Elitist Iterated Racing

- irace may “lose” the best-so-far configuration
  ⇒ Each new iteration (race) forgets the results of the previous one
- Protect the best configurations (*elites*) from being discarded unless all their results are considered
- after race \( i \), elites were evaluated in \( I_e \) instances
- race \( i + 1 \) will start with \( I_{new} \cup I_e \) instances
- irace remembers the values of the elites on \( I_e \)
- elites can only be discarded after alive configurations are evaluated on at least all \( I_{new} \cup I_e \)
  (similar to ParamILS’s domination concept, but more strict)
- non-elites are discarded as usual
The irace Package

- Implementation of Iterated Racing in R
  - Goal 1: Flexible
  - Goal 2: Easy to use
- R package available at CRAN
- Use it through the command-line: (see `irace --help`)
  ```
  irace --max-experiments 1000 --param-file parameters.txt
  ```
- No knowledge of R needed

The irace Package: version 2.3

- Elitist irace by default
- New interfaces with more intuitive names
- A detailed user-guide / tutorial:
  - [https://cran.r-project.org/web/packages/irace/vignettes/irace-package.pdf](https://cran.r-project.org/web/packages/irace/vignettes/irace-package.pdf)
- Available from CRAN (GNU/Linux, Windows, OSX)
  - [https://cran.r-project.org/package=irace](https://cran.r-project.org/package=irace)

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
```

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

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

The irace Package: Options

- `maxExperiments`: maximum number of runs of the target algorithm (tuning budget)
- `digits`: number of decimal places to be considered for the real parameters (default: 4)
- `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: target-runner

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

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

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

- An R function

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

The irace Package: Other features

- Initial configurations
  - “seed” irace with the default configuration
- Parallel evaluation: MPI, multiple cores, Grid Engine / qsub
- Forbidden configurations:
  ```
  popsize < 5 & LS == "SA"
  ```
- Recovery file: allows resuming an interrupted irace run
- Test instances
  - Specify not only the training instances but also test instances for comparing results
An overview of applications of irace

Parameter tuning
- Exact MIP solvers (CPLEX, SCIP with > 200 parameters)
- single-objective optimization metaheuristics
- 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 [Miranda et al., 2014] (mlr R package, Bischl et al.)

Automatic design of control software for robots [Francesca et al., 2015]

irace (and others) works great for
- Complex parameter spaces:
  - numerical, categorical, ordinal, subordinate (conditional)
- Large parameter spaces (few hundred parameters)
- Heterogeneous instances
- Medium to large tuning budgets (thousands of target runs)
- Target 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 target runs)
- Very large tuning budgets (millions of target runs)
- Target runs require days

Conclusions

“For procedures that require parameter tuning, the available data must be partitioned into a training and a test set. Tuning should be performed in the training set only.”


“The performance of swarm intelligence algorithms […] is often strongly dependent on the value of the algorithm parameters. Such values should be set using either sound statistical procedures […] or automatic parameter tuning procedures.”

[Swarm Intelligence Journal (Springer)]
Example #1

Automated Offline Design of Algorithms

Example: Tuning ACOTSP

Example: ACOTSP

Thomas Stütze. ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem, 2002.

Command-line program:

```
$ ./acotsp -i instance -t 20 --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-acotsp.txt

# Name  Label/switch  Type  Domain  Condition
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, 1.00) | algorithm == "acs"
rasrank     "--rasranks"  i    (1, 100) | algorithm == "ras"
elitistants "--elitistants "i    (1, 750) | algorithm == "eas"
nlsls       "--nnls "     i    (5, 50)  | localsearch %in% c(1,2,3)
dlb         "--dlb "     c    (0, 1)   | localsearch %in% c(1,2,3)
```
Example: ACOTSP

```bash
#!/bin/bash
CONFIG_ID=$1
INSTANCE_ID=$2
SEED=$3
INSTANCE=$4
CONFIG_PARAMS=$*
FIXED_PARAMS=" --time 1 --tries 1 --quiet "
STDOUT="c$CONFIG_ID-$INSTANCE_ID.stdout"
acotsp $FIXED_PARAMS -i $INSTANCE --seed $SEED $CONFIG_PARAMS > $STDOUT
COST=$(grep -oE 'Best [-+0-9.e]+' $STDOUT | cut -d' ' -f2)
echo "$COST"
exit 0
```

$ cat target-runner

Example: ACOTSP

```bash
$ dir Instances/
3000-01.tsp 3000-02.tsp 3000-03.tsp ...
$ cat scenario.txt
trainInstancesDir = "/Instances"
maxExperiments = 1000
digits = 2

✓ Good to go:

$ irace --parallel 2 --debug-level 1

- --parallel to execute in parallel
- --debug-level to see what irace is executing

Example: ACOTSP: and more

- Initial configurations:
  ```bash
  $ cat default.txt
  ```
  ```
algorithmlocalsearch alpha beta rho ants nnls dlb q0
  as 0 1.0 1.0 0.95 10 NA NA NA
  ```

- Logical expressions that forbid configurations:
  ```bash
  $ cat forbidden.txt
  ```
  ```
  (alpha == 0.0) & (beta == 0.0)
  ```

Example #1

- Configuring known algorithms

```
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)

### Benchmark set Default Configured Speedup

<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

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

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

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

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 optimality

Hypervolume measure ≈ Anytime behaviour

Manuel López-Ibáñez and Thomas Stützle.
Automatically improving the anytime behaviour of optimisation algorithms.
Scenario #1: Brute-Force Approach

- Choose *many* parameter settings
- Run lots of experiments
- Visually compare SQT plots

After about one year:

+ Strategies for varying ants, $\beta$, or $q_0$ that significantly improve the anytime behaviour of MMAS on the TSP.
- Extremely time consuming
- Subjective / Bias

Scenario #2: Experimental comparison

<table>
<thead>
<tr>
<th>Parameter Setting</th>
<th>Relative Deviation from Best-Known</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>0.9834</td>
</tr>
<tr>
<td>auto var ants</td>
<td>0.9826</td>
</tr>
<tr>
<td>auto var beta</td>
<td>0.9826</td>
</tr>
<tr>
<td>auto var rho</td>
<td>0.9767</td>
</tr>
<tr>
<td>auto var $q_0$</td>
<td>0.9932</td>
</tr>
<tr>
<td>auto var ALL</td>
<td>1.2012</td>
</tr>
</tbody>
</table>

Scenario #2: SCIP

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

- 200 parameters controlling search, heuristics, thresholds, . . .
  1 000 training + 1 000 testing instances
- Single run timeout: 300 seconds
- irace budget (*maxExperiments*): 5 000 runs
Example #3

**Integration in algorithm (re-)engineering process**

- 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 may help the designer gain insight useful for further development*

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

[Montes de Oca et al., 2011]

- Automated design from (flexible) algorithm frameworks

---

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

[Montes de Oca et al., 2011]

- Comparison on 100D, median values across 19 functions

---
General approach

Automated Offline Design of Algorithms

Main approaches

Top-down approaches
- develop flexible framework following a fixed algorithm template with alternatives
- apply high-performing configurators
- Examples: Satenstein, MOACO, AutoMOEA, MIP Solvers

Bottom-up approaches
- flexible framework implementing algorithm components
- define rules for composing algorithms from components e.g. through grammars
- frequently usage of genetic programming, grammatical evolution etc.

Example #4

A more complex example: MOACO framework

- A flexible framework of multi-objective ACO 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


The automatic design of multi-objective ant colony optimization algorithms.

Manuel López-Ibáñez and Thomas Stützle.
A more complex example: MOACO framework

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

\[ \text{irace + hypervolume = automatic configuration of multi-objective solvers!} \]

Results: Multi-objective components

Example #5

Top-down design approach: MOEA framework

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

- Other examples:
  - Single-objective top-down frameworks for MIP: CPLEX, SCIP
  - Single-objective top-down framework for SAT: SATenstein
    
  [KhudaBukhsh, Xu, Hoos, and Leyton-Brown, 2009]
  
  - 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]
An even more complex example: MOEA framework

- Replicate as many well-known MOEAs as possible from the same template
- The template has a number of configurable algorithmic components
- Each component can be configured by choosing one option from various alternatives
- Aim to maximise the number of different configurations that are valid MOEAs

AutoMOEA: A MOEA template

1: pop := Initialization()
2: if type(pop_ext) != none then
3:  pop_ext := pop
4: repeat
5:  pool := BuildMatingPool(pop)
6:  pop_new := Variation(pool)
7:  pop_new := Evaluation(pop_new)
8:  pop := Replacement(pop, pop_new)
9:  if type(pop_ext) == bounded then
10:     pop_ext := Archiving(pop_ext, pop_new)
11:  else if type(pop_ext) == unbounded then
12:     pop_ext := pop_ext ∪ pop
13: until termination criteria met
14: if type(pop_ext) == none then
15:    return pop
16: else
17:    return pop_ext

AutoMOEA: Main components

<table>
<thead>
<tr>
<th>Component</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuildMatingPool</td>
<td>⟨PreferenceMat, Selection⟩</td>
</tr>
<tr>
<td>Replacement</td>
<td>⟨PreferenceRep, Removal⟩</td>
</tr>
<tr>
<td>Archiving</td>
<td>⟨PreferenceExt, RemovalExt⟩</td>
</tr>
<tr>
<td>Preference</td>
<td>⟨Fitness, Diversity⟩</td>
</tr>
</tbody>
</table>

“On Set-Based Multiobjective Optimization” [Zitzler, Thiele, and Bader, 2010]

- Set-partitioning
  - dominance count
  - dominance rank
  - dominance strength
  - dominance depth
  - dominance depth-rank

- Pareto-compliant quality measures
  - binary indicator ($I_e$ or $I_H$)
  - exclusive hypervolume contribution ($I^1_H$)
  - shared hypervolume contribution ($I^2_H$)

- Diversity measures
  - niche sharing
  - k-th nearest neighbor (kNN)
  - crowding distance

- Algorithm
  - Fitness
  - Diversity
  - NSGA-II: dominance depth, crowding distance
  - SPEA2: dom. strength, kNN
  - IBEA: binary indicator
  - HypE: $I^1_H$
  - SMS-EMOA: dom. depth-rank, $I^1_H$
AutoMOEA: Main components

<table>
<thead>
<tr>
<th>Component</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuildMatingPool</td>
<td>Preference_{Mat}, Selection</td>
</tr>
<tr>
<td>Replacement</td>
<td>Preference_{Rep}, Removal</td>
</tr>
<tr>
<td>Archiving</td>
<td>Preference_{Ext}, Removal_{Ext}</td>
</tr>
<tr>
<td>Preference</td>
<td>Fitness, Diversity</td>
</tr>
</tbody>
</table>

AutoMOEA: Automatic Design

Automatic configuration (irace) + Flexible algorithmic framework (AutoMOEA) = Automatic design of state-of-the-art MOEAs

<table>
<thead>
<tr>
<th>BuildMatingPool</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetPart</td>
<td>Quality</td>
</tr>
<tr>
<td>MOGA</td>
<td>rank</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>depth</td>
</tr>
<tr>
<td>SPEA2</td>
<td>strength</td>
</tr>
<tr>
<td>IBEA</td>
<td>binary indicator</td>
</tr>
<tr>
<td>HypE</td>
<td>—</td>
</tr>
<tr>
<td>SMS-EMOA</td>
<td>—</td>
</tr>
<tr>
<td>DTLZ 2-obj</td>
<td>—</td>
</tr>
<tr>
<td>DTLZ 3-obj</td>
<td>depth-rank $l_H$</td>
</tr>
<tr>
<td>DTLZ 5-obj</td>
<td>rank</td>
</tr>
<tr>
<td>WFG 2-obj</td>
<td>rank</td>
</tr>
<tr>
<td>WFG 3-obj</td>
<td>count</td>
</tr>
<tr>
<td>WFG 5-obj</td>
<td>count</td>
</tr>
</tbody>
</table>

AutoMOEA: Automatic Design

<table>
<thead>
<tr>
<th>BuildMatingPool</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetPart</td>
<td>Quality</td>
</tr>
<tr>
<td>MOGA</td>
<td>rank</td>
</tr>
<tr>
<td>NSGA-II</td>
<td>depth</td>
</tr>
<tr>
<td>SPEA2</td>
<td>strength</td>
</tr>
<tr>
<td>IBEA</td>
<td>binary indicator</td>
</tr>
<tr>
<td>HypE</td>
<td>—</td>
</tr>
<tr>
<td>SMS-EMOA</td>
<td>—</td>
</tr>
<tr>
<td>DTLZ 2-obj</td>
<td>—</td>
</tr>
<tr>
<td>DTLZ 3-obj</td>
<td>depth-rank $l_H$</td>
</tr>
<tr>
<td>DTLZ 5-obj</td>
<td>rank</td>
</tr>
<tr>
<td>WFG 2-obj</td>
<td>rank</td>
</tr>
<tr>
<td>WFG 3-obj</td>
<td>count</td>
</tr>
<tr>
<td>WFG 5-obj</td>
<td>count</td>
</tr>
</tbody>
</table>

AutoMOEA: Automatic Design

Automatic configuration (irace) + Flexible algorithmic framework (AutoMOEA) = Automatic design of state-of-the-art MOEAs

- Fair to compare with untuned traditional MOEAs?
- Why is our setup representative?
- Different AutoMOEAs for termination criterion in FEs or seconds
- How do you define “state-of-the-art”?
- What is a “novel” MOEA?

AutoMOEA: Automatic Design

Automatic configuration (irace) + Flexible algorithmic framework (AutoMOEA) = Automatic design of state-of-the-art MOEAs

- Finding a state-of-the-art algorithm is “easy”:
  - problem modeling + algorithmic components + computing power

AutoMOEA: Automatic Design

Automatic configuration (irace) + Flexible algorithmic framework (AutoMOEA) = Automatic design of state-of-the-art MOEAs

- Finding a state-of-the-art algorithm is “easy”:
  - problem modeling + algorithmic components + computing power

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

- Finding a state-of-the-art algorithm is “easy”:
  - problem modeling + algorithmic components + computing power

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

- Finding a state-of-the-art algorithm is “easy”:
  - problem modeling + algorithmic components + computing power

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

- Finding a state-of-the-art algorithm is “easy”:
  - problem modeling + algorithmic components + computing power
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]

Automatic design of hybrid SLS algorithms
[Marmion, Mascia, López-Ibáñez, Stützle, 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, and Stützle, 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 through recursion
  - problem specific implementation at low-level
  - separation of generic and problem-specific components
Example instantiations of some metaheuristics

<table>
<thead>
<tr>
<th>perturb</th>
<th>ls</th>
<th>accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA</td>
<td>random move</td>
<td>∅     Metropolis</td>
</tr>
<tr>
<td>PII</td>
<td>random move</td>
<td>∅     Metropolis, fixed T</td>
</tr>
<tr>
<td>TS</td>
<td>∅     TS</td>
<td>∅</td>
</tr>
<tr>
<td>ILS</td>
<td>any    any</td>
<td>any</td>
</tr>
<tr>
<td>IG</td>
<td>destruct/construct</td>
<td>any</td>
</tr>
<tr>
<td>GRASP</td>
<td>rand. greedy sol.</td>
<td>any</td>
</tr>
</tbody>
</table>

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>) | firstImpDescent(<comparator>, <stop>)
<sa> ::= ILS(<pbs_move>, no_ls, <metropolis>, <stop>)
<rii> ::= ILS(<pbs_move>, no_ls, probRandom, <stop>)
<pv> ::= ILS(<pbs_move>, no_ls, prob(<value_prob_accept>), <stop>)
<vs> ::= ILS(<pbs_variable_move>, firstImpDescent(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}
Flow-shop problem with makespan objective [Pagnozzi, Stütze, 2017]

- Automatic configuration:
  - max. three levels of recursion
  - biased / unbiased grammar resulting in 262 and 502 parameters, respectively
  - budget: 200 000 trials of $n \cdot m \cdot 0.03$ seconds

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

Flow-shop problem with total completion time objective [Pagnozzi, Stütze, 2017]

- Automatic configuration:
  - max. three levels of recursion
  - budget: 100 000 trials of $n \cdot m \cdot 0.03$ seconds

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

Flow-shop problem with total tardiness objective [Pagnozzi, Stütze, 2017]

- Automatic configuration:
  - max. three levels of recursion
  - budget: 100 000 trials of $n \cdot m \cdot 0.03$ seconds

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

Summary

Contributions
- approach to automate design and analysis of (hybrid) metaheuristics
- not a silver bullet, but needs right components, especially problem-specific ones
- better or equal performance to state-of-the-art for UBQP, TSP-TW, many (flow-shop) scheduling problems
- directly extendible for automated comparison of metaheuristics

Current/future work
- extensions to other methods and templates
- dealing with complexity of hybrid algorithms
- increase generality, tackling wide problem classes
Why automatic algorithm configuration?

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

Towards a paradigm shift in algorithm design

Conclusions

Automatic Configuration

- leverages computing power for software design
- is rewarding w.r.t. development time and algorithm performance

Future work

- more powerful configurators
- more and more complex applications
- paradigm shift in optimization software development

Configurable, flexible (SLS) frameworks XXL

- paradigm shift in SLS algorithm development
- configurable frameworks XXL
- solve = model + configure + search

Many challenges remain on (i) problem representations, (ii) algorithmic structure, (iii) algorithmic components and generation thereof, (iv) automatic configuration techniques, and (v) extensions to other techniques and challenging problems, . . .
Acknowledgments

The tutorial has benefited from collaborations and discussions with our colleagues: Prasanna Balaprakash, Mauro Birattari, Jérémie Dubois-Lacoste, Alberto Franzin, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown, Tianjun Liao, Marie-Élénore Marmon, Franco Mascia, Marco Montes de Oca, Federico Pagozzi, 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° 249593
- PAl project COMEX funded by the Interveruniversity Attraction Poles Programme of the Belgian Science Policy Office and the EU FP7 ICT Project COLOMBO, Cooperative Self-Organizing System for Low Carbon Mobility at Low Penetration Rates (agreement no. 318622)
- the EU FP7 ICT Project CMU-SWARM, Cooperative Self-Organizing Swarm Intelligence
- and the EU FP7 ICT Project COLOMBO, Cooperative Self-Organizing System for Low Carbon Mobility at Low Penetration Rates (agreement no. 318622)

Thomas Stützle acknowledges support of the F.R.S.-FNRS of which he is a senior research associate.

References II

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° 249593
- PAl project COMEX funded by the Interveruniversity Attraction Poles Programme of the Belgian Science Policy Office and the EU FP7 ICT Project COLOMBO, Cooperative Self-Organizing System for Low Carbon Mobility at Low Penetration Rates (agreement no. 318622)

References III

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° 249593
- PAl project COMEX funded by the Interveruniversity Attraction Poles Programme of the Belgian Science Policy Office and the EU FP7 ICT Project COLOMBO, Cooperative Self-Organizing System for Low Carbon Mobility at Low Penetration Rates (agreement no. 318622)

Thomas Stützle acknowledges support of the F.R.S.-FNRS of which he is a senior research associate.

References I

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° 249593
- PAl project COMEX funded by the Interveruniversity Attraction Poles Programme of the Belgian Science Policy Office and the EU FP7 ICT Project COLOMBO, Cooperative Self-Organizing System for Low Carbon Mobility at Low Penetration Rates (agreement no. 318622)
References IV


References V


AClib: A Benchmark Library for Algorithm Configuration


http://www.aclib.net/

- Standard benchmark for experimenting with configurators
- 326 heterogeneous scenarios
- SAT, MIP, ASP, time-tabling, TSP, multi-objective, machine learning
- Extensible ⇒ new scenarios welcome!

 Scaling to expensive instances

What if my problem instances are too difficult/large?

- Cloud computing / Large computing clusters
  - Tune on easy instances, then ordered F-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