Automatic Algorithm Configuration

Thomas Stützle

IRIDIA, CoDE, Université Libre de Bruxelles
Brussels, Belgium

stuetzle@ulb.ac.be
iridia.ulb.ac.be/~stuetzle

Outline

1. Context
2. Automatic algorithm configuration
3. Automatic configuration methods
4. Applications
5. Concluding remarks
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 on 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, neighborhoods, pre-processing, strategies, ...
  - many design choices have associated numerical parameters
  - example: multi-objective ACO algorithms with 22 parameters (plus several still hidden ones)
Example: Ant Colony Optimization

Modeling side

Algorithm side

Problem

Solution components

Pheromone model

Probabilistic solution construction

Local search

Pheromone update
Probabilistic solution construction

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
  - \( \ldots \) many more \( \ldots \)
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, ...
  - numerical parameters may be *conditional* to specific values of categorical or ordinal parameters

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

Designing optimization algorithms

Challenges

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

Traditional design approach

- trial–and–error design guided by expertise/intuition
  - prone to over-generalizations, implicit independence assumptions, limited exploration of design alternatives

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

Automated algorithm configuration

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

Offline configuration and online parameter control

Offline configuration

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

Online parameter control

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

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

Typical performance measures

- maximize solution quality (within given computation time)
- minimize computation time (to reach optimal solution)

Approaches to configuration

- numerical optimization techniques
  - e.g. MADS [Audet&Orban, 2006], various [Yuan et al., 2012]
- heuristic search methods
  - e.g. meta-GA [Grefenstette, 1985], ParamILS [Hutter et al., 2007, 2009], gender-based GA [Ansótegui at al., 2009], linear GP [Oltean, 2005], REVAC(++) [Eiben & students, 2007, 2009, 2010] ...
- experimental design techniques
  - e.g. CALIBRA [Adenso–Díaz, Laguna, 2006], [Ridge&Kudenko, 2007], [Coy et al., 2001], [Ruiz, Stützle, 2005]
- 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
Approaches to configuration

- numerical optimization techniques
  - e.g. MADS [Audet&Orban, 2006], various [Yuan et al., 2012]
- heuristic search methods
  - e.g. meta-GA [Grefenstette, 1985], ParamILS [Hutter et al., 2007, 2009], gender-based GA [Ansótegui et al., 2009], linear GP [Oltean, 2005], REVAC(++) [Eiben & students, 2007, 2009, 2010] ...
- experimental design techniques
  - e.g. CALIBRA [Adenso-Díaz, Laguna, 2006], Ridge&Kudenko, 2007, Coy et al., 2001, Ruiz, Stützle, 2005
- 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*

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

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

Some applications

International time-tabling competition
- winning algorithm configured by F-race
- interactive injection of new configurations

Vehicle routing and scheduling problem
- first industrial application
- improved commercialized algorithm

F-race in stochastic optimization
- evaluate “neighbours” using F-race (solution cost is a random variable!)
- good performance if variance of solution cost is high
Iterated race

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 race)

---

**Iterated race: an illustration**

- sample configurations from initial distribution
- While not terminate()
  1. apply race
  2. modify the distribution
  3. sample configurations with selection probability
Sampling distributions

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

- Truncated normal distribution

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

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

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

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

- Discrete probability distribution

$$\Pr^z\{X_d = x_j\} = \begin{bmatrix} x_1 & x_2 & \ldots & x_{n_d} \\ 0.1 & 0.3 & \ldots & 0.4 \end{bmatrix}$$

- Updated by increasing probability of parameter value in elite configuration and reducing probabilities of others

The irace Package


- implementation of Iterated Racing in R
  
  Goal 1: flexible

  Goal 2: easy to use

- but no knowledge of R necessary
Other tools: ParamILS, SMAC

ParamILS
- iterated local search in configuration space
- requires discretization of numerical parameters
- [http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/](http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/)

SMAC
- surrogate model assisted search process
- encouraging results for large configuration spaces

*capping*: effective speed-up technique for configuration target run-time

Applications of automatic configuration tools

- configuration of “black-box” solvers
  - e.g. mixed integer programming solvers, continuous optimizers
- supporting tool in algorithm engineering
  - e.g. metaheuristics for probabilistic TSP, re-engineering PSO
- bottom-up generation of heuristic algorithms
  - e.g. heuristics for SAT, FSP, etc.; metaheuristic framework
- design configurable algorithm frameworks
  - e.g. Satenstein, MOACO, UACOR
Example, configuration of “black-box” solvers

### Mixed-integer programming solvers

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

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

<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>

Focused ILS, 10 runs, 2 CPU days, 63 parameters
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, Stützle, 2013]
General Local Search Structure: ILS

\[ s_0 := \text{initSolution} \]
\[ s^* := \text{ls}(s_0) \]

\[
\textbf{repeat}
\]
\[ s' := \text{perturb}(s^*,\text{history}) \]
\[ s^{*'} := \text{ls}(s') \]
\[ s^* := \text{accept}(s^*, s^{*'}, \text{history}) \]
\textbf{until} termination criterion met

- many SLS methods instantiable from this structure
- abilities
  - hybridization
  - recursion
  - problem specific implementation at low-level

Grammar

```plaintext
<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>)
  | firstImprDescend(<comparator>, <stop>)
<sa> ::= ILS(<pbs_move>, no_ls, <metropolis>, <stop>)
<rii> ::= ILS(<pbs_move>, no_ls, probRandom, <stop>)
<pb> ::= ILS(<pbs_variable_move>, firstImprDescend(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\}
```
<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>
<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}

<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}
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

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>ALS1</th>
<th>ALS2</th>
<th>ALS3</th>
<th>soa−IG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>26600</td>
<td>27000</td>
<td>27400</td>
<td>26600</td>
</tr>
<tr>
<td>10000</td>
<td>24200</td>
<td>24600</td>
<td>25000</td>
<td>24200</td>
</tr>
<tr>
<td>15000</td>
<td>21800</td>
<td>22200</td>
<td>22600</td>
<td>21800</td>
</tr>
<tr>
<td>20000</td>
<td>19400</td>
<td>19800</td>
<td>20200</td>
<td>19400</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>ALS1</th>
<th>ALS2</th>
<th>ALS3</th>
<th>soa−IG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>29000</td>
<td>29400</td>
<td>29800</td>
<td>29000</td>
</tr>
<tr>
<td>10000</td>
<td>26600</td>
<td>27000</td>
<td>27400</td>
<td>26600</td>
</tr>
<tr>
<td>15000</td>
<td>24200</td>
<td>24600</td>
<td>25000</td>
<td>24200</td>
</tr>
<tr>
<td>20000</td>
<td>21800</td>
<td>22200</td>
<td>22600</td>
<td>21800</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>ALS1</th>
<th>ALS2</th>
<th>ALS3</th>
<th>soa−IG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>33000</td>
<td>33400</td>
<td>33800</td>
<td>33000</td>
</tr>
<tr>
<td>10000</td>
<td>30600</td>
<td>31000</td>
<td>31400</td>
<td>30600</td>
</tr>
<tr>
<td>15000</td>
<td>28200</td>
<td>28600</td>
<td>29000</td>
<td>28200</td>
</tr>
<tr>
<td>20000</td>
<td>25800</td>
<td>26200</td>
<td>26600</td>
<td>25800</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>ALS1</th>
<th>ALS2</th>
<th>ALS3</th>
<th>soa−IG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>410000</td>
<td>420000</td>
<td>430000</td>
<td>410000</td>
</tr>
<tr>
<td>10000</td>
<td>386000</td>
<td>390000</td>
<td>400000</td>
<td>386000</td>
</tr>
<tr>
<td>15000</td>
<td>362000</td>
<td>366000</td>
<td>376000</td>
<td>362000</td>
</tr>
<tr>
<td>20000</td>
<td>338000</td>
<td>342000</td>
<td>352000</td>
<td>338000</td>
</tr>
</tbody>
</table>

Results are competitive or superior to state-of-the-art algorithm

Example, design configurable algorithm framework

Multi-objective ant colony optimization (MOACO)
Multi-objective Optimization

- many real-life problems are multiobjective
- no a priori knowledge $\leadsto$ Pareto-optimality

MOACO framework
López-Ibáñez, Stütze, 2012

- algorithm framework for multi-objective ACO algorithms
- can instantiate MOACO algorithms from literature
- 10 parameters control the multi-objective part
- 12 parameters control the underlying pure “ACO” part

Example of a top-down approach to algorithm configuration
MOACO framework

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

Automatic configuration multi-objective ACO
Automatic configuration multi-objective ACO

Why automatic algorithm configuration?

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

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

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

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

Towards a shift of paradigm in algorithm design
Towards a shift of paradigm in algorithm design
Conclusions

Status

- using automatic configuration tools is rewarding in terms of development time and algorithm performance
- interactive usage of configurators allows humans to focus on creative part of algorithm design
- many application opportunities also in other areas than optimization

Future work

- more powerful configurators
- more and more complex applications
- best practice