Structure at the meta-level:
Observations on the structure of design spaces of high-performance solvers for hard combinatorial problems

Holger H. Hoos

BETA Lab
Department of Computer Science
University of British Columbia
Canada

based on joint work with
Chris Fawcett, Frank Hutter, Kevin Leyton-Brown, Thomas Stützle
Acknowledgements

Collaborators:

- Domagoj Babić
- Sam Bayless
- Chris Fawcett
- Quinn Hsu
- Frank Hutter
- Erez Karpas
- Chris Nell
- Eugene Nudelman
- Steve Ramage
- Gabriele Röger
- Marius Schneider
- James Styles
- Dave Tompkins
- Mauro Vallati
- Lin Xu
- Thomas Bartz-Beielstein (FH Köln, Germany)
- Marco Chiarandini (Syddansk Universitet, Denmark)
- Alfonso Gerevini (Università degli Studi di Brescia, Italy)
- Malte Helmert (Universität Basel, Switzerland)
- Alan Hu
- Kevin Leyton-Brown
- Kevin Murphy
- Alessandro Saetti (Università degli Studi di Brescia, Italy)
- Thomas Stützle (Université Libre de Bruxelles, Belgium)

Research funding:

- NSERC, Mprime, GRAND, CFI
- IBM, Actenum Corp.

Computing resources:

- Arrow, BETA, ICICS clusters
- Compute Canada / WestGrid
Take-home message:

- exploiting structure in problem instances permits practical solution of hard problems
  \(\sim\) instance-level structure

- structure in space of algorithms (\(+\) human creativity) facilitates effective construction of good solvers for hard problems
  \(\sim\) meta-level structure

- meta-level structure may differ substantially from instance-level structure

- PbO (rich algorithm design space \(+\) automated configuration) permits (partial) automation of building effective solvers; efficacy depends on exploitation of meta-level structure
application context
Holger Hoos: Structure at the Meta-Level
Programming by Optimisation (PbO)

- program \(\sim\) (large) space of programs
- encourage software developers to
  - avoid premature commitment to design choices
  - seek & maintain design alternatives
- automatically find performance-optimising designs for given use context(s)
Programming by Optimization

When creating software, developers commonly explore different ways of achieving certain tasks. These alternatives are often eliminated or abandoned early in the process, based on the idea that the flexibility they afford would be difficult or impossible to exploit later. This article challenges this view, advocating an approach that encourages developers not only to avoid premature commitment to certain design choices but also to actively develop and promote alternatives for parts of the design. In this approach, called Programming by Optimization (PBO), developers specify a potentially large design space of programs that accomplishes a given task, from which versions of the program optimized for various use contexts are generated automatically, including parallel versions derived from the same sequential sources. We outline a simple, generic programming language extension that supports the specification of such design spaces and discuss ways specific programs can be derived from these design spaces.

---

Communications of the ACM, 55(2), pp. 70–80, February 2012

www.prog-by-opt.net
Levels of PbO:

**Level 4:** Make no design choice prematurely that cannot be justified compellingly.

**Level 3:** Strive to provide design choices and alternatives.

**Level 2:** Keep and expose design choices considered during software development.

**Level 1:** Expose design choices hardwired into existing code (magic constants, hidden parameters, abandoned design alternatives).

**Level 0:** Optimise settings of parameters exposed by existing software.
Success in optimising speed:

<table>
<thead>
<tr>
<th>Application, Design choices</th>
<th>Speedup</th>
<th>PbO level</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT-based software verification (SPEAR), 41</td>
<td>4.5–500 ×</td>
<td>2–3</td>
</tr>
<tr>
<td>Hutter, Babić, HH, Hu (2007)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AI Planning (LPG), 62</td>
<td>3–118 ×</td>
<td>1</td>
</tr>
<tr>
<td>Vallati, Fawcett, Gerevini, HH, Saetti (2011)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mixed integer programming (CPLEX), 76</td>
<td>2–52 ×</td>
<td>0</td>
</tr>
<tr>
<td>Hutter, HH, Leyton-Brown (2010)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... and solution quality:

University timetabling, 18 design choices, PbO level 2–3
⇒ new state of the art; UBC exam scheduling
Fawcett, Chiarandini, HH (2009)

Machine learning / Classification, 803 design choices, PbO level 0–1
⇒ outperforms specialised model selection & hyper-parameter optimisation methods from machine learning
Thornton, Hutter, HH, Leyton-Brown (2012)
Outline

1. Introduction
2. Design spaces & design optimisation
3. Which choices matter? Global perspectives
4. Which choices matter? A local perspective
5. Speculation and open questions
Design optimisation

Simplest case: Configuration / tuning

- Standard optimisation techniques
  (e.g., CMA-ES – Hansen & Ostermeier 01; MADS – Audet & Orban 06)

- Advanced sampling methods
  (e.g., REVAC, REVAC++ – Nannen & Eiben 06–09)

- Racing
  (e.g., F-Race – Birattari, Stützle, Paquete, Varrentrapp 02;
   Iterative F-Race – Balaprakash, Birattari, Stützle 07)

- Model-free search
  (e.g., ParamILS – Hutter, HH, Stützle 07;
   Hutter, HH, Leyton-Brown, Stützle 09)

- Sequential model-based optimisation
  (e.g., SPO – Bartz-Beielstein 06; SMAC – Hutter, HH, Leyton-Brown 11–12)
Iterated Local Search

Local Search
Iterated Local Search

Perturbation
Iterated Local Search

Local Search
Iterated Local Search

Local Search
Iterated Local Search

Selection (using Acceptance Criterion)
Iterated Local Search

Perturbation
ParamILS

- iterated local search in configuration space
- initialisation: pick best of default + $R$ random configurations
- subsidiary local search: iterative first improvement, change one parameter in each step
- perturbation: change $s$ randomly chosen parameters
- acceptance criterion: always select better configuration
- number of runs per configuration increases over time; ensure that incumbent always has same number of runs as challengers
Sequential Model-based Optimisation
e.g., Jones (1998), Bartz-Beielstein (2006)

- **Key idea:**
  - use predictive performance model (response surface model) to find good configurations
- perform runs for selected configurations (initial design) and fit model (e.g., noise-free Gaussian process model)
- iteratively select promising configuration, perform run and update model
Sequential Model-based Optimisation

- parameter response
- measured

Holger Hoos: Structure at the Meta-Level
Sequential Model-based Optimisation

- parameter response
- measured
- model
Sequential Model-based Optimisation

- parameter response
- measured
- model
- predicted best
Sequential Model-based Optimisation

- parameter response
- measured
- model
Sequential Model-based Optimisation

- Parameter response
- Measured
- Model
- Predicted best
Sequential Model-based Optimisation
Sequential Model-based Optimisation
Sequential Model-based Optimisation

- parameter response
- measured
- model
Sequential Model-based Optimisation

- parameter response
- measured
- model
- predicted best

new incumbent found!
Sequential Model-based Algorithm Configuration (SMAC)
Hutter, HH, Leyton-Brown (2011)

- uses *random forest model* to predict performance of parameter configurations
- predictions based on algorithm parameters and instance features, aggregated across instances
- finds promising configurations based on *expected improvement criterion*, using multi-start local search and random sampling
- initialisation with single configuration (algorithm default or randomly chosen)
Observation: Some design choices matter more than others depending on . . .

- algorithm under consideration
- given use context

Knowledge which choices / parameters matter may . . .

- guide algorithm development
- facilitate configuration
Forward selection based on empirical performance models

Hutter, HH, Leyton-Brown (2013)

Key idea:

- build regression models of algorithm performance as a function of input parameters (= design choices)

  \( \leadsto \) empirical performance models (EPMs)

- consider only subset of parameters \( S \), ignore all others

- starting with \( S = \emptyset \), iteratively add parameters one at a time

- in each iteration, greedily add parameter resulting in max. improvement in accuracy of regression model
EPMs work:
Hutter, HH, Leyton-Brown (to appear in AIJ)

SPEAR on SAT-encoded IBM software verification problems
true running times $[\log_{10} \text{CPU sec}]$
EPMs work:

Hutter, HH, Leyton-Brown (to appear in AIJ)

SPEAR on SAT-encoded IBM software verification problems
predicted running times $[\log_{10} \text{CPU sec}]$
EPMs work:
Hutter, HH, Leyton-Brown (to appear in AIJ)

CPLEX 12.1 on MIP problems from computational sustainability
true running times [log_{10} CPU sec]
EPMs work:
Hutter, HH, Leyton-Brown (to appear in AIJ)

**CPLEX 12.1 on MIP problems from computational sustainability predicted running times** [log_{10} CPU sec]
Empirical study:

- high-performance solvers for SAT, MIP, TSP (23–76 parameters),
  well-known sets of benchmark data (real-world structure)

- random forest models for performance prediction,
  trained on 1000 randomly sampled configurations per solver
Good prediction accuracies for few parameters:

LK-H on TSPLIB
(RMSE for $\log_{10}$ running times in CPU sec)
Good prediction accuracies for few parameters:

SPEAR on SAT-encoded IBM software verification problems
(RMSE for $\log_{10}$ running times in CPU sec)
How important is each parameter?

Cost of omission
= impact on model accuracy from excluding single parameters

LK-H on TSPLIB
(normalised to 100 for most important parameter)
How important is each parameter?

Cost of omission
= impact on model accuracy from excluding single parameters

SPEAR on SAT-encoded IBM software verification problems
(normalised to 100 for most important parameter)
Key idea:
- build regression model of algorithm performance as a function of all input parameters (= design choices)
  \( \sim \) empirical performance models (EPMs)
- analyse variance in model output (= predicted performance) due to each parameter, parameter interactions
- importance of parameter: fraction of performance variation over configuration space explained by it (main effect)
- analogous for sets of parameters (interaction effects)
Decomposition of variance in a nutshell

For parameters $p_1, \ldots, p_n$ and a function (performance model) $y$:

$$y(p_1, \ldots, p_n) = \mu + f_1(p_1) + f_2(p_2) + \cdots + f_n(p_n) + f_{1,2}(p_1, p_2) + f_{1,3}(p_1, p_3) + \cdots + f_{n-1,n}(p_{n-1}, p_n) + f_{1,2,3}(p_1, p_2, p_3) + \cdots + \cdots$$
Note:

- Straightforward computation of main and interaction effects is intractable.
  (integration over combinatorial spaces of configurations)

- For random forest models, marginal performance predictions and variance decomposition (up to constant-sized interactions) can be computed exactly and efficiently.
Empirical study:

- 8 high-performance solvers for SAT, ASP, MIP, TSP (4–85 parameters)
- 12 well-known sets of benchmark data (random + real-world structure)
- random forest models for performance prediction, trained on 10,000 randomly sampled configurations per solver + data from 25+ runs of SMAC configuration procedure
Fraction of variance explained by main effects:

- CPLEX on RCW (comp sust) 70.3%
- CPLEX on CORLAT (comp sust) 35.0%
- Clasp on software verification 78.9%
- Clasp on DB query optimisation 62.5%
- CryptoMiniSAT on bounded model checking 35.5%
- CryptoMiniSAT on software verification 31.9%
Fraction of variance explained by main + 2-interaction effects:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPLEX on RCW (comp sust)</td>
<td>70.3% + 12.7%</td>
</tr>
<tr>
<td>CPLEX on CORLAT (comp sust)</td>
<td>35.0% + 8.3%</td>
</tr>
<tr>
<td>Clasp on software verification</td>
<td>78.9% + 14.3%</td>
</tr>
<tr>
<td>Clasp on DB query optimisation</td>
<td>62.5% + 11.7%</td>
</tr>
<tr>
<td>CryptoMiniSAT on bounded model checking</td>
<td>35.5% + 20.8%</td>
</tr>
<tr>
<td>CryptoMiniSAT on software verification</td>
<td>31.9% + 28.5%</td>
</tr>
</tbody>
</table>
Note:
may pick up variation caused by poorly performing configurations

Simple solution:
cap at default performance (or quantile from distribution of randomly sampled configurations); build model from capped data.
Which choices matter? A local perspective

Note: We are mostly interested in good configurations (obtained from design optimisation)

Questions:
- Which differences between two configurations matter (how much)?
- How close to the default can good performance be obtained?
- How sensitive is an optimised configuration to parameter changes?

Answers may . . .
- guide algorithm development
- facilitate configuration
- improve performance of default configurations
- improve robustness of optimised configurations
Ablation analysis
Fawcett, HH (under review)

Key idea:
- given two configurations, $A$ and $B$, change one parameter at a time to get from $A$ to $B$
  $\leadsto$ ablation path
- in each step, change parameter to achieve maximal gain (or minimal loss) in performance
- for computational efficiency, use racing (F-race) for evaluating parameters considered in each step
Prototypical ablation results:

all parameters are equally important
(note: log scale!)
Prototypical ablation results:

few parameters are most important
Empirical study:

- high-performance solvers for SAT, MIP, AI Planning (26–76 parameters), well-known sets of benchmark data (real-world structure)

- optimised configurations obtained from ParamILS (minimisation of penalised average running time; 10 runs per scenario, 48 CPU hours each)
Ablation between default and optimised configurations:

LPG on Depots planning domain
Ablation between default and optimised configurations:

SPEAR on SAT-encoded IBM software verification problems
Which parameters are important?

SPEAR on SAT-encoded IBM software verification instances:
- `sp-var-dec-heur` (99.92% of overall performance gain!)
- `sp-rand-var-dec-scaling`
- `sp-res-cutoff-cls`
- `sp-first-restart`

LPG (AI Planning):
Importance of parameters varies between planning domains
Ablation between optimised configurations:

CPLEX 12.1 on MIP problems from computational sustainability
⇒ large plateau of good configurations
Ablation between optimised configurations:

SPEAR on SAT-encoded IBM software verification problems

\[ \rightsquigarrow \text{possibility of barriers between good configurations} \]
Speculation and open questions

Optimisation at the meta-level

- candidate solutions are engineering designs
- evaluation is (very) noisy (problem instances)
- evaluation is expensive
- cost of evaluation often depends on quality of candidate solution (e.g., for minimisation of running time)

_different methods, different types of structure, different ways to exploit structure_
Some hypotheses

HH (2012), Fawcett & HH (under review), Hutter, HH, Leyton-Brown (2013)

- parameters interact, but not too much
- individual parameter responses tend to be well-behaved (uni-modal)
- (few) key parameters need to have certain settings, depending on use context (vz backbones)

large, shallow basins around optimised configurations

- for highly parametric algorithms, there are barrier-free paths between optimised configurations (vz neutral paths in landscapes of RNA secondary structures)
Open questions

- Applicability of / insights from standard landscape analysis techniques? (auto-correlation, fitness distance analysis, ...)

- Insights that can be exploited for better design optimisers (configurators)?

- Principles that can guide algorithm developers using PbO to more effectively optimisable designs?
Take-home message:

- exploiting structure in problem instances permits practical solution of hard problems
  \[\sim \text{instance-level structure}\]

- structure in space of algorithms (+ human creativity) facilitates effective construction of good solvers for hard problems
  \[\sim \text{meta-level structure}\]

- meta-level structure may differ substantially from instance-level structure

- PbO (rich algorithm design space + automated configuration) permits (partial) automation of building effective solvers; efficacy depends on exploitation of meta-level structure