AI-Augmented Algorithms
How I Learned to Stop Worrying and Love Choice

Lars Kotthoff
University of Wyoming
larisko@uwyo.edu

Boulder, 16 January 2019
Outline

▷ Big Picture
▷ Motivation
▷ Choosing Algorithms
▷ Tuning Algorithms
▷ (NCAR-relevant) Applications
▷ Outlook and Resources
Big Picture

▷ advance the state of the art through meta-algorithmic techniques
▷ rather than inventing new things, use existing things more intelligently – automatically
▷ invent new things through combinations of existing things
Big Picture

- advance the state of the art through meta-algorithmic techniques
- rather than inventing new things, use existing things more intelligently – automatically
- invent new things through combinations of existing things

https://xkcd.com/720/
Motivation – What Difference Does It Make?
Performance Differences

Leveraging the Differences

Performance Improvements

Common Theme

Performance models of black-box processes
▷ also called surrogate models
▷ substitute expensive underlying process with cheap approximate model
▷ build approximate model using machine learning techniques based on results of evaluations of the underlying process
▷ no knowledge of what the underlying process is required (but can be helpful)
▷ may facilitate better understanding of the underlying process through interrogation of the model
Choosing Algorithms
Algorithm Selection

Given a problem, choose the best algorithm to solve it.

Algorithm Selection

Algorithm Selection

Portfolio

Algorithm 1
Algorithm 2
Algorithm 3

Training Instances

Instance 1
Instance 2
Instance 3

Feature Extraction

Performance Model

Instance 4: Algorithm 2
Instance 5: Algorithm 3
Instance 6: Algorithm 3

Feature Extraction

Instance 4
Instance 5
Instance 6
Algorithm Portfolios

▷ instead of a single algorithm, use several complementary algorithms
▷ idea from Economics – minimise risk by spreading it out across several securities
▷ same for computational problems – minimise risk of algorithm performing poorly
▷ in practice often constructed from competition winners or other algorithms known to have good performance

“algorithm” used in a very loose sense

- algorithms
- heuristics
- machine learning models
- software systems
- machines
- ...
Parallel Portfolios

Why not simply run all algorithms in parallel?
▷ not enough resources may be available/waste of resources
▷ algorithms may be parallelized themselves
▷ memory/cache contention
Building an Algorithm Selection System

▷ requires algorithms with complementary performance
▷ most approaches rely on machine learning
▷ train with representative data, i.e. performance of all algorithms in portfolio on a number of instances
▷ evaluate performance on separate set of instances
▷ potentially large amount of prep work
Key Components of an Algorithm Selection System

- feature extraction
- performance model
- prediction-based selector/scheduler

Optional:
- presolver
- secondary/hierarchical models and predictors (e.g. for feature extraction time)
Types of Performance Models

**Types of Models**

- **Regression Models**
  - A1: 1.2
  - A2: 4.5
  - A3: 3.9

- **Classification Model**
  - A1
  - A2
  - A3

- **Pairwise Classification Models**
  - A1 vs. A2: A1: 1 vote, A2: 0 votes

- **Pairwise Regression Models**
  - A1 - A2: 0
  - A1 - A3: 0
  - A3: 1.7
Tuning Algorithms
Algorithm Configuration

Given a (set of) problem(s), find the best parameter configuration.
Parameters?

▷ anything you can change that makes sense to change
▷ e.g. search heuristic, optimization level, computational resolution
▷ not random seed, whether to enable debugging, etc.
▷ some will affect performance, others will have no effect at all
Automated Algorithm Configuration

▷ no background knowledge on parameters or algorithm – black-box process
▷ as little manual intervention as possible
  ▷ failures are handled appropriately
  ▷ resources are not wasted
  ▷ can run unattended on large-scale compute infrastructure
Algorithm Configuration

Parameter domains & starting values

Configurator

Calls with different parameter settings

Configuration scenario

Target algorithm

Solves

Problem instances

Returns solution cost

General Approach

▷ evaluate algorithm as black-box function
▷ observe effect of parameters without knowing the inner workings, build surrogate model based on this data
▷ decide where to evaluate next, based on surrogate model
▷ repeat
When are we done?

▷ most approaches incomplete, i.e. do not exhaustively explore parameter space
▷ cannot prove optimality, not guaranteed to find optimal solution (with finite time)
▷ performance highly dependent on configuration space

→ How do we know when to stop?
Time Budget

How much time/how many function evaluations?

▷ too much $\rightarrow$ wasted resources
▷ too little $\rightarrow$ suboptimal result
▷ use statistical tests
▷ evaluate on parts of the instance set
▷ for runtime: adaptive capping
▷ in general: whatever resources you can reasonably invest
Grid and Random Search

▷ evaluate certain points in parameter space

Model-Based Search

▷ evaluate small number of configurations
▷ build model of parameter-performance surface based on the results
▷ use model to predict where to evaluate next
▷ repeat
▷ allows targeted exploration of new configurations
▷ can take instance features into account like algorithm selection

Model-Based Search Example

Iter = 1, Gap = 1.9909e−01

- $y$
- $e_i$
- $x$
- $\text{type}$
  - $\text{init}$
  - $\text{prop}$

- $\text{y}$
- $\text{yhat}$
- $\text{ei}$
Model-Based Search Example

Iter = 2, Gap = 1.9909e−01

- \( y \)
- \( x \)
- \( \text{type} \)
- \( \text{init} \)
- \( \text{prop} \)
- \( \text{seq} \)
- \( \text{yhat} \)
- \( \text{ei} \)

Graph showing iterations and gap for model-based search.
Model-Based Search Example

Iter = 3, Gap = 1.9909e−01
Model-Based Search Example

Iter = 4, Gap = 1.9992e−01
Model-Based Search Example

Iter = 5, Gap = 1.9992e−01
Model-Based Search Example

Iter = 6, Gap = 1.9996e−01

Diagram shows a graph with x and y axes, labeled with points and lines indicating different types such as init, prop, seq, y, yhat, and ei.
Model-Based Search Example

Iter = 8, Gap = 2.0000e−01
Model-Based Search Example

Iter = 9, Gap = 2.0000e−01

Type
- init
- prop
- seq

Y

Ei

X

Y

Ei

Iter = 9, Gap = 2.0000e−01
Model-Based Search Example

Iter = 10, Gap = 2.0000e−01

![Graph showing model-based search example](image-url)
Selected Applications
Compiler Parameter Tuning

- pre-defined optimization levels offer not much flexibility
- improvements possible by tuning full compiler parameter space
- tuned compute-intensive AI algorithms
- up to 40% runtime improvement over gcc -O2/-O3

---

Compiler Parameter Tuning

▷ not only for C/C++
▷ JavaScript (JavaScriptCode, V8) optimized for standard benchmarks
▷ up to 35% runtime improvement

“Deep” Parameter Tuning

▷ automatically identify non-exposed parameters and allow them to be tuned (e.g. magic constants)
▷ tuned dlmalloc library, specialized for e.g. awk, flex, sed
▷ runtime improvements of up to 12%, decrease in memory consumption of up to 21%

---

Beyond Software

\[ I_G/I_D = 1.2 \]

\[ I_G/I_D = 6.8 \]
Outlook
Quo Vadis, Software Engineering?

Quo Vadis, Software Engineering?

## Algorithm Selection Literature Summary

**Last update 21 November 2018**

<table>
<thead>
<tr>
<th>citation</th>
<th>domain</th>
<th>features</th>
<th>predict what</th>
<th>predict how</th>
<th>predict when</th>
<th>portfolio</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Langley 1993a, Langley 1993b</td>
<td>search</td>
<td>past performance</td>
<td>algorithm</td>
<td>hand-crafted and learned rules</td>
<td>offline and online</td>
<td>dynamic</td>
<td>1993</td>
</tr>
<tr>
<td>Carlson et al. 1991</td>
<td>planning</td>
<td>problem domain features, search statistics</td>
<td>control rules</td>
<td>exploration-based rule construction</td>
<td>online</td>
<td>dynamic</td>
<td>1991</td>
</tr>
<tr>
<td>Gratch and Daujong 1992</td>
<td>planning</td>
<td>problem domain features, search statistics</td>
<td>control rules</td>
<td>probabilistic rule construction</td>
<td>online</td>
<td>dynamic</td>
<td>1992</td>
</tr>
<tr>
<td>Smith and Selß 1992</td>
<td>software design</td>
<td>features of abstract representation</td>
<td>algorithms and data structures</td>
<td>simulated annealing</td>
<td>offline</td>
<td>static</td>
<td>1992</td>
</tr>
<tr>
<td>Aha 1992</td>
<td>machine learning</td>
<td>instance features</td>
<td>algorithm</td>
<td>learned rules</td>
<td>offline</td>
<td>static</td>
<td>1992</td>
</tr>
<tr>
<td>Brody 1993</td>
<td>machine learning</td>
<td>instance and algorithm features</td>
<td>algorithm</td>
<td>hand-crafted rules</td>
<td>offline</td>
<td>static</td>
<td>1993</td>
</tr>
<tr>
<td>Kanet et al. 1993</td>
<td>differential equations</td>
<td>past performance, instance features</td>
<td>algorithm</td>
<td>hand-crafted rules</td>
<td>offline</td>
<td>static</td>
<td>1993</td>
</tr>
<tr>
<td>Carli 1994</td>
<td>software design</td>
<td>instance features</td>
<td>algorithms and data structures</td>
<td>frame-based knowledge base</td>
<td>online</td>
<td>static</td>
<td>1994</td>
</tr>
<tr>
<td>Tsang et al. 1995</td>
<td>CSP</td>
<td>instance features</td>
<td>algorithm</td>
<td>learning rules</td>
<td>offline</td>
<td>static</td>
<td>1995</td>
</tr>
<tr>
<td>Breuer 1995</td>
<td>software design</td>
<td>runtime performance, algorithms and data structures, instance parameters</td>
<td>Bayesian belief propagation, neural nets</td>
<td>static</td>
<td>static</td>
<td>1996</td>
<td></td>
</tr>
<tr>
<td>Borrett et al. 1996</td>
<td>CSP</td>
<td>search statistics, search algorithm</td>
<td>algorithm</td>
<td>decision trees, Bayesian classifier, nearest neighbour, neural net</td>
<td>static, static order</td>
<td>1996</td>
<td></td>
</tr>
<tr>
<td>Huberman et al. 1997</td>
<td>graph colouring</td>
<td>past performance</td>
<td>resource allocation</td>
<td>statistical-model</td>
<td>offline</td>
<td>static</td>
<td>1997</td>
</tr>
<tr>
<td>Gomes and Selman 1997a, Gomes and Selman 1997b</td>
<td>CSP</td>
<td>algorithm</td>
<td>dynamic</td>
<td>statistical-model</td>
<td>offline</td>
<td>static</td>
<td>1997</td>
</tr>
<tr>
<td>Cook and Varel 1997</td>
<td>parallel search</td>
<td>runtime performance</td>
<td>decision trees, Bayesian classifier, nearest neighbour, neural net</td>
<td>online</td>
<td>static</td>
<td>1997</td>
<td></td>
</tr>
<tr>
<td>Cirstea et al. 1999</td>
<td>vehicle routing problem</td>
<td>runtime performance</td>
<td>algorithm</td>
<td>genetic algorithms</td>
<td>offline</td>
<td>static</td>
<td>1999</td>
</tr>
<tr>
<td>Howe et al. 1999</td>
<td>planning</td>
<td>instance features</td>
<td>resource allocation</td>
<td>linear regression</td>
<td>offline</td>
<td>static</td>
<td>1999</td>
</tr>
<tr>
<td>Tantinachirniet al. 1999</td>
<td>scheduling</td>
<td>instance and search features</td>
<td>genetic algorithms</td>
<td>offline</td>
<td>static</td>
<td>1999</td>
<td></td>
</tr>
<tr>
<td>Wilson et al. 2000</td>
<td>software design</td>
<td>instance features, instance features</td>
<td>nearest neighbour</td>
<td>static</td>
<td>static</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>Beck and Fox 2000</td>
<td>job shop scheduling</td>
<td>instance feature changes during search</td>
<td>algorithm scheduling policy</td>
<td>hand-crafted rules</td>
<td>offline</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Brelsford and Shaer 2000</td>
<td>classification</td>
<td>past performance</td>
<td>ranking</td>
<td>distribution model</td>
<td>offline</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Lagoudakis and Ullman 2000</td>
<td>order selection, sorting</td>
<td>instance features</td>
<td>remaining cost for each sub-problem</td>
<td>static model</td>
<td>offline</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Sibilo 2000</td>
<td>CSP</td>
<td>classification</td>
<td>instance features, probing</td>
<td>9 different classifiers</td>
<td>static</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Pfahringer et al. 2000</td>
<td>TSP</td>
<td>past performance</td>
<td>resource allocation</td>
<td>performance simulation for different allocations</td>
<td>offline</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Fukunaga 2000</td>
<td>CSP</td>
<td>classification</td>
<td>instance features, probing</td>
<td>algorithm</td>
<td>offline</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Soares and Brelsford 2000</td>
<td>machine learning</td>
<td>past performance</td>
<td>resource allocation</td>
<td>algorithm</td>
<td>offline</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Gomes and Selman 2001</td>
<td>CSP</td>
<td>instance features</td>
<td>ranking</td>
<td>nearest neighbour</td>
<td>offline</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Lagoudakis and Ullman 2001</td>
<td>DPLL branching rules</td>
<td>instance features</td>
<td>remaining cost for each sub-problem</td>
<td>MGP</td>
<td>online</td>
<td>static</td>
<td>2001</td>
</tr>
<tr>
<td>Neuwirth 2001</td>
<td>optimization</td>
<td>search statistics</td>
<td>expected utility of algorithm</td>
<td>reinforcement learning</td>
<td>offline and online</td>
<td>static</td>
<td>2001</td>
</tr>
<tr>
<td>Harvey et al. 2001</td>
<td>CSP</td>
<td>runtime performance, instance generator features, search statistics</td>
<td>runtime performance, restart parameters</td>
<td>Bayesian model</td>
<td>offline and online</td>
<td>static</td>
<td>2001</td>
</tr>
</tbody>
</table>

https://larskotthoff.github.io/asssurvey/

Tools and Resources

**LLAMA**  https://bitbucket.org/lkotthoff/llama
**SATzilla**  http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/
  - **iRace**  http://iridia.ulb.ac.be/irace/
  - **mlrMBO**  https://github.com/mlr-org/mlrMBO
  - **SMAC**  http://www.cs.ubc.ca/labs/beta/Projects/SMAC/
  - **Spearmint**  https://github.com/HIPS/Spearmint
  - **TPE**  https://jaberg.github.io/hyperopt/

  - **autofolio**  https://bitbucket.org/mlindauer/autofolio/
  - **Auto-WEKA**  http://www.cs.ubc.ca/labs/beta/Projects/autoweka/
  - **Auto-sklearn**  https://github.com/automl/auto-sklearn
Summary

**Algorithm Selection** choose the best *algorithm* for solving a problem

**Algorithm Configuration** choose the best *parameter configuration* for solving a problem with an algorithm

▷ mature research areas
▷ can combine configuration and selection
▷ effective tools are available
▷ COnfiguration and SElection of ALgorithms group COSEAL
  http://www.coseal.net

Don’t set parameters prematurely, embrace choice!
I’m hiring!

Several funded graduate positions available.