Automated Algorithm Selection and Configuration

Lars Kotthoff

University of British Columbia
larsko@cs.ubc.ca

UCL, 4 May 2016
Outline

▷ Motivation
▷ Algorithm Selection Problem
▷ Algorithm Configuration
Big Picture

- Advance the state of the art through meta-algorithmic techniques
- rather than inventing new things, use existing things better
Performance Differences

Leveraging the Differences

Performance Improvements

Algorithm Selection
Algorithm Selection

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

$x \in \mathcal{P}$
Problem space

$A \in \mathcal{A}$
Algorithm space

$S$
Selection model

$S(x) = A$
Prediction

$p \in \mathcal{R}^n$
Performance measure

$\forall x \in \mathcal{P}' \subset \mathcal{P},$
$A \in \mathcal{A}:$
$p(A, x)$
Training data

Feature extraction
Feedback
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
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)
Features

- relate properties of problem instances to performance
- relatively cheap to compute
- specified by domain expert
- syntactic – analyse instance description
- probing – run algorithm for short time
- dynamic – instance changes while algorithm is running
Types of Performance Models

▷ models for entire portfolios
▷ models for individual algorithms
▷ models that are somewhere in between (e.g. pairs of algorithms)
Models for Entire Portfolios

▷ predict the best algorithm in the portfolio
▷ alternatively: cluster and assign best algorithms to clusters

optional (but important):
▷ attach a “weight” during learning (e.g. the difference between best and worst solver) to bias model towards the “important” instances
▷ special loss metric


Models for Individual Algorithms

▷ predict the performance for each algorithm separately
▷ combine the predictions to choose the best one
▷ for example: predict the runtime for each algorithm, choose the one with the lowest runtime

---

Hybrid Models

▷ get the best of both worlds
▷ for example: consider pairs of algorithms to take relations into account
▷ for each pair of algorithms, learn model that predicts which one is faster


Types of Predictions/Algorithm Selectors

▷ best algorithm
▷ $n$ best algorithms ranked
▷ allocation of resources to $n$ algorithms
▷ change the currently running algorithm?


Time of Prediction

before problem is being solved
▷ select algorithm(s) once
▷ no recourse if predictions are bad

while problem is being solved
▷ continuously monitor problem features and/or performance
▷ can remedy bad initial choice or react to changing problem
Example System – SATzilla

- 7 SAT solvers, 4811 problem instances
- syntactic (33) and probing features (15)
- ridge regression to predict log runtime for each solver, choose the solver with the best predicted performance
- later version uses random forests to predict better algorithm for each pair, aggregation through simple voting scheme
- pre-solving, feature computation time prediction, hierarchical model
- won several competitions

---

Tools

claspfolio  http://www.cs.uni-potsdam.de/claspfolio/
LLAMA      https://bitbucket.org/lkotthoff/llama
SATzilla   http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/
## Algorithm Selection literature summary

**Last update 08 December 2015**

<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>Carbonell et al. 1991</td>
<td>planning</td>
<td>problem domain features, search statistics</td>
<td>control rules</td>
<td>explanation-based rule construction</td>
<td>online</td>
<td>dynamic</td>
<td>1991</td>
</tr>
<tr>
<td>Gratch and Dejiing 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 Settiti 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>Brodtkre 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>Kernell 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>Caihli 1994</td>
<td>software design</td>
<td>instance features</td>
<td>algorithms and data structures</td>
<td>frame-based knowledge base</td>
<td>offline</td>
<td>static</td>
<td>1994</td>
</tr>
<tr>
<td>Tsing et al. 1995</td>
<td>constraints</td>
<td>instance features</td>
<td>-</td>
<td>-</td>
<td>static</td>
<td>1995</td>
<td></td>
</tr>
<tr>
<td>Bremer 1995</td>
<td>software design</td>
<td>runtime performance</td>
<td>algorithms, data structures and their parameters</td>
<td>statistical model</td>
<td>-</td>
<td>static</td>
<td>1995</td>
</tr>
<tr>
<td>Beavett al. 1996</td>
<td>constraints</td>
<td>search statistics</td>
<td>switch algorithm</td>
<td>hand-crafted rules</td>
<td>online</td>
<td>static</td>
<td>1996</td>
</tr>
<tr>
<td>Sakkett et al. 1994</td>
<td>constraints</td>
<td>search statistics</td>
<td>switch algorithm</td>
<td>hand-crafted rules</td>
<td>online</td>
<td>static</td>
<td>1996</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 1997b, Gomes and Selman 1997a</td>
<td>constraints</td>
<td>problem size and past performance</td>
<td>algorithm</td>
<td>statistical model</td>
<td>offline</td>
<td>static</td>
<td>1997</td>
</tr>
<tr>
<td>Cook and Varnell 1997</td>
<td>parallel search</td>
<td>proving</td>
<td>set of search strategies</td>
<td>decision trees, Bayesian classifier, nearest neighbour, neural net</td>
<td>online</td>
<td>static</td>
<td>1997</td>
</tr>
<tr>
<td>Lobayot and Lemoil 1999</td>
<td>branch and bound</td>
<td>proving</td>
<td>runtime performance</td>
<td>algorithm</td>
<td>genetic algorithms</td>
<td>offline</td>
<td>static</td>
</tr>
<tr>
<td>Caoeis 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>Hoew 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>Terashima-Minent al. 1999</td>
<td>scheduling</td>
<td>instance and search features</td>
<td>algorithm</td>
<td>genetic algorithms</td>
<td>offline</td>
<td>static</td>
<td>1999</td>
</tr>
<tr>
<td>Wilson et al. 2000</td>
<td>software design</td>
<td>instance features</td>
<td>data structures</td>
<td>nearest neighbour</td>
<td>offline</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Beck and Fox 2000</td>
<td>job shop scheduling</td>
<td>instance feature changes</td>
<td>algorithm</td>
<td>hand-crafted rules</td>
<td>online</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Brazil and Soarh 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>Lagnodakis and Litten 2000</td>
<td>order selection, sorting</td>
<td>instance features</td>
<td>remaining cost for each sub-problem</td>
<td>MDP</td>
<td>online</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Silito 2000</td>
<td>constraints</td>
<td>proving</td>
<td>cost of solving problem</td>
<td>statistical model</td>
<td>offline</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Pfahnerger et al. 2000</td>
<td>classification</td>
<td>instance features, proving</td>
<td>algorithm</td>
<td>9 different classifiers</td>
<td>offline</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Fukushima 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>Soarh and Brazil 2000</td>
<td>machine learning</td>
<td>instance features</td>
<td>ranking</td>
<td>nearest neighbour</td>
<td>offline</td>
<td>static</td>
<td>2000</td>
</tr>
<tr>
<td>Gomes and Selman 2001</td>
<td>constraints, mixed integer</td>
<td>past performance</td>
<td>algorithm</td>
<td>statistical model</td>
<td>offline</td>
<td>dynamic</td>
<td>2001</td>
</tr>
</tbody>
</table>

http://larskotthoff.github.io/assurvey/

<table>
<thead>
<tr>
<th>Study</th>
<th>Methodology</th>
<th>Input Type</th>
<th>Input Features</th>
<th>Model Type</th>
<th>Learning Type</th>
<th>Type</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brewer 1995</td>
<td>software design</td>
<td>runtime performance</td>
<td>algorithms, data structures and their parameters</td>
<td>statistical model</td>
<td>offline</td>
<td>static</td>
<td>1995</td>
</tr>
<tr>
<td>Cahill 1994</td>
<td>software design</td>
<td>instance features</td>
<td>algorithms and data structures</td>
<td>frame-based knowledge base</td>
<td>offline</td>
<td>static</td>
<td>1994</td>
</tr>
<tr>
<td>Cavazos and O'Boyle 2006</td>
<td>software design</td>
<td>instance features</td>
<td>algorithm</td>
<td>logistic regression</td>
<td>offline</td>
<td>static</td>
<td>2006</td>
</tr>
<tr>
<td>Gent et al. 2010b</td>
<td>software design</td>
<td>instance features</td>
<td>implementation</td>
<td>19 different classifiers</td>
<td>offline</td>
<td>static</td>
<td>2010</td>
</tr>
<tr>
<td>Park et al. 2012</td>
<td>software design</td>
<td>instance features</td>
<td>runtime performance</td>
<td>SVM</td>
<td>offline</td>
<td>static</td>
<td>2012</td>
</tr>
<tr>
<td>Simon et al. 2013</td>
<td>software design</td>
<td>instance features</td>
<td>algorithm</td>
<td>neural networks, decision trees</td>
<td>offline</td>
<td>dynamic</td>
<td>2013</td>
</tr>
<tr>
<td>Smith and Setliff 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>Wilson et al. 2000</td>
<td>software design</td>
<td>instance features</td>
<td>data structures</td>
<td>nearest neighbour</td>
<td>offline</td>
<td>static</td>
<td>2000</td>
</tr>
</tbody>
</table>
Algorithm Configuration
(with material from Frank Hutter, Marius Lindauer and Holger Hoos)
Algorithm Configuration

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

“Unlike our human subjects, [the system] experimented with a wide variety of combinations of heuristics. Our human subjects rarely had the inclination or patience to try many alternatives, and on at least one occasion incorrectly evaluated alternatives that they did try [...]”

In Context

Automated Procedures to model & optimize empirical performance

Problem instances -> Domain expert(s) -> Solution strategies

Algorithm configuration
Find best for

Performance prediction
Fit model \( f(\cdot, \cdot) \approx \cdot \)
Quantify importance of algorithm components

Algorithm portfolios
Select best for new
Parameter Spaces

▷ numeric – 1, 2, 3…
▷ ordinal – a, b, c…
▷ categoric – UCL, UBC, UCC…

→ not every tool suitable for every type of parameter
General Approach

▷ evaluate algorithm as black-box function
▷ observe effect of parameters without knowing the inner workings
▷ balance diversification/exploration and intensification/exploitation
When are we done?

- most approaches incomplete
- 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?
How much time/how many function evaluations?
▷ too much → wasted resources
▷ too little → suboptimal result
▷ use statistical tests
▷ evaluate on parts of the data
▷ for runtime: adaptive capping
Grid and Random Search

- evaluate certain points in parameter space

Grid Layout

Random Layout

---
Population-Based Methods

▷ e.g. Racing and Genetic Algorithms
▷ start with population of random configurations
▷ eliminate “weak” individuals
▷ generate new population from “strong” individuals
▷ iterate


Local Search

- start with random configuration
- change a single parameter (local search step)
- if better, keep the change, else revert
- repeat
- optional (but important): restart with new random configurations

---

Local Search Example
Local Search Example

Initialisation
Local Search Example

Local Search
Local Search Example
Local Search Example

Perturbation
Local Search Example
Local Search Example

Local Search
Local Search Example
Local Search Example

Selection (using Acceptance Criterion)

Animation credit: Holger Hoos

Hutter & Lindauer
AC-Tutorial
AAAI 2016, Phoenix, USA 33
Model-Based Search

▷ build model of parameter-response surface
▷ allows targeted exploration of new configurations
▷ can take instance features into account like algorithm selection

Model-Based Search Example

Iter = 1, Gap = 0.0000e+00

(type)
- init
- prop

(type)
- y
- yhat
- ei
Model-Based Search Example

Iter = 6, Gap = 0.0000e+00

Type
- init
- prop
- seq
- y
- yhat
- ei

Iter = 6, Gap = 0.0000e+00
Model-Based Search Example

Iter = 13, Gap = 0.0000e+00

- Type: init, prop, seq, y, yhat, ei

- Graph showing the model-based search process with iterations and gap values.
Model-Based Search Example

Iter = 20, Gap = 0.0000e+00
Tools and Resources

**HPOlib** [http://www.automl.org/hpolib.html](http://www.automl.org/hpolib.html)

**iRace** [http://iridia.ulb.ac.be/irace/](http://iridia.ulb.ac.be/irace/)

**mlrMBO** [https://github.com/mlr-org/mlrMBO](https://github.com/mlr-org/mlrMBO)


**Spearmint** [https://github.com/HIPS/Spearmint](https://github.com/HIPS/Spearmint)

**TPE** [https://jaberg.github.io/hyperopt/](https://jaberg.github.io/hyperopt/)
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

Thank you!