

11. Automated algorithm selection

AEA 2026

Motivation

Meta-learning

Algorithm runtime prediction (for SAT)

Generalizations to other domains

No Free Lunch theorems

Always picking the best algorithm?

- ▶ "No Free Lunch" theorem: no algorithm can be the best across all possible problems and, on average, all algorithms perform the same.

Motivation for **Algorithm Selection**: if, on average, **different** algorithms are the best for different parts of the problem space, selecting them based on the features of the problem can improve performance.

Task: identify a portfolio of complementary algorithms and a strategy for choosing between them.

Debate: if the NFL theorems would apply to Algorithm Selection systems, the ramifications in practice may not be significant.

Algorithm selection

- ▶ *The algorithm selection problem*: select the most suitable algorithm for a particular problem.

Which algorithm is likely to perform best for my problem? - a learning task.

- ▶ *Meta-learning*: learning about **ML algorithm**'s performance on **classification problems**
- ▶ Methods based on algorithm selection began to outperform standalone solvers in **SAT** competitions¹.
Other problems addressed: CSP, planning, Max-SAT, ASP.

¹SATzilla: portfolio-based algorithm selection for SAT, 2008

Algorithm selection

The **winner-take-all** approach: measure algorithms' performance on a set of instances, and use **the algorithm with the lowest average runtime**.

- ▶ neglect algorithms that, while uncompetitive on average, have good performance on particular instances

The algorithm selection problem

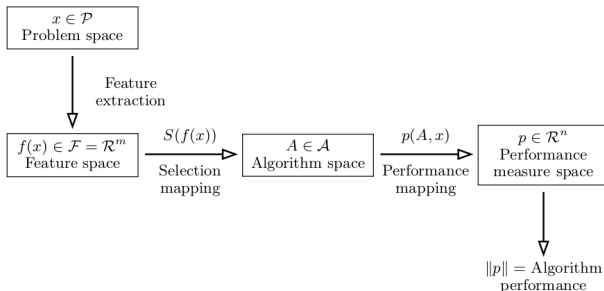
A formal abstract model:

- ▶ the **problem** space \mathcal{P} : the set of instances of a problem class
- ▶ the **feature** space \mathcal{F} : measurable characteristics of the instances (generated by a computational **feature extraction** process applied to \mathcal{P})
- ▶ the **algorithm** space \mathcal{A} : the set of considered algorithms for tackling the problem
- ▶ the **performance** space: the mapping of each algorithm to a set of performance metrics

The problem: for a problem instance $x \in \mathcal{P}$, with features $f(x) \in \mathcal{F}$, find the selection mapping $S(f(x))$ into algorithm space \mathcal{A} s.t. the selected algorithm $A \in \mathcal{A}$ maximizes the performance mapping $p(A, x)$.

The algorithm selection problem

Figure: Schematic diagram of Rice's algorithm selection problem model. The objective is to determine the selection mapping S so as to have high algorithm performance.



Algorithm Selection

ML techniques **learn the performance mapping** (from problems to algorithms) using features extracted from the problems.

- ▶ a *training phase*: algorithms are **run on a sample of the problem space** to experimentally evaluate their performance
- ▶ the training data are used to create a *performance model* used to predict the performance on new problems

Key components of an Algorithm Selection system:

- ▶ feature extraction
- ▶ performance model
- ▶ prediction-based selector

Content

Motivation

Meta-learning

Algorithm runtime prediction (for SAT)

Generalizations to other domains

Meta-learning

In ML, Algorithm Selection is referred to as *meta-learning*².

- ▶ Select the best learning algorithm to solve a **classification problem** using different performance measures (accuracy).

- ▶ Characterize a classification problem and use **rule-based learning algorithms** to develop rules for selection:

If the dataset has characteristics C_1, C_2, \dots, C_n

Then use algorithm A1 and don't use algorithm A2.

Meta-learning for classification problems

The study of Aha '92 ²: an empirical method for generalizing results from case studies

- ▶ A nearest-neighbor classifier (IB1), a set covering rule learner (CN2), a decision tree (C4)
- ▶ Instances: training datasets from the letter recognition database
- ▶ Features: #instances, #classes, #relevant and irrelevant attributes, #prototypes per class, the distribution range of the instances and prototypes

```
IF (#training instances < 737) AND  
(#prototypes per class > 5.5) AND  
(#relevants > 8.5) AND  
(#irrelevants < 5.5)  
THEN IB1 will be better than CN2.
```

StatLog³ (*Comparative Testing of Statistical and Logical Learning*) project: a broader range of datasets, algorithms and features.

Figure: Features used to characterize classification datasets

Measures	Definitions
Simple	
N	Number of examples
p	Number of attributes
q	Number of classes
Bin.att	Number of binary attributes
Cost	Cost matrix indicator
Statistical	
SD	Standard deviation ratio (geometric mean)
corr.abs	Mean absolute correlation of attributes
cancor1	First canonical correlation
fract1	Fraction separability due to cancor1
skewness	Skewness—mean of $ E(X - \mu)^3 /\sigma^3$
kurtosis	Kurtosis—mean of $ E(X - \mu)^4 /\sigma^4$
Information theory	
$H(C)$	Entropy (complexity) of class
$\bar{H}(X)$	Mean entropy (complexity) of attributes
$\bar{M}(C, X)$	Mean mutual information of class and attributes
EN.attr	Equivalent number of attributes $H(C)/\bar{M}(C, X)$
NS.ratio	Noise-signal ratio $(\bar{H}(X) - \bar{M}(C, X))/\bar{M}(C, X)$

Information theoretical measures can capture the randomness of an instance, which correlates with complexity.

StatLog project

In the notation of Rice's model:

- ▶ \mathcal{P} : 22 classification datasets
- ▶ \mathcal{A} : 23 machine learning, neural, statistical learning algorithms
- ▶ \mathcal{F} : 16 features of the datasets
- ▶ performance: classification accuracy


No single algorithm performed best for all problems (supported by the NFL theorem).

To learn rules for each algorithm (the mapping S): use a **decision tree** algorithm (C4.5).

Example: rules generated for the *CART* (*Classification And Regression Tree*) algorithm

CART recommended IF ($N \leq 6435$ AND $skew > 0.57$)
CART **not** recommended IF ($N > 6435$ OR ($N < 6435$ AND $skew < 0.57$))

Valuable meta-data widely used by researchers (extended studies⁴)

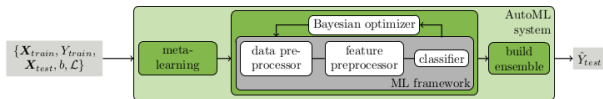
⁴Ali, S., Smith, K. *On learning algorithm selection for classification*, 2006  13/36

Automated Machine Learning

- ▶ Automated machine learning (AutoML)⁵
<http://www.ml4aad.org/automl/>: **automatically** choose a good **algorithm** and **feature preprocessing** steps for a new dataset, and set their **hyperparameters**.
- ▶ AUTO-SKLEARN⁶ uses **Bayesian optimization**, a SOTA optimization framework for the **global optimization of expensive blackbox functions**.

AUTO-SKLEARN improves the AutoML methods by:

- ▶ a meta-learning step for initializing the Bayesian optimizer
- ▶ construct ensembles from the models evaluated during the optimization



⁵F. Hutter et al. *Automated Machine Learning: Methods, Systems, Challenges*, '19

⁶M. Feurer et al. *Efficient and Robust Automated Machine Learning*, '15

Bayesian optimization

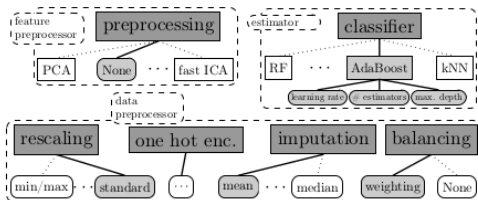
No analytical expression of $f(x)$ and the cost of finding $f(x)$ is very high.

Find global optimal value of $f(x)$, $\max f(x)$

- ▶ Surrogate function $g()$ guess $f(x)$, given the current information
- ▶ Acquisition function guides our choice of the next sample of x

AUTO-SKLEARN's components

- ▶ **Classification algs:** linear models, SVMs, nearest neighbors, naive Bayes, decision trees, ensemble, etc.
- ▶ **Feature preprocessing:** feature selection, kernel approximation, matrix decomposition, embeddings, feature clustering, polynomial feature expansion, etc.
- ▶ **Data preprocessing:** rescaling of the inputs, imputation of missing values, one-hot encoding, balancing of the target classes



Content

Motivation


Meta-learning

Algorithm runtime prediction (for SAT)

Generalizations to other domains

Algorithm runtime prediction

- ▶ Predict the runtime of an algorithm on an unseen input: use ML techniques to build a **model of the algorithm's runtime** as a function of problem-specific instance features
- ▶ *Empirical performance models (EPM)*⁷, used for:
 - ▶ **algorithm selection**
 - ▶ parameter tuning and algorithm configuration
 - ▶ generate hard benchmarks
 - ▶ gain insights into instance hardness and algorithm performance

⁷F. Hutter et al. *Algorithm runtime prediction: Methods & evaluation*, '14  18/36

Empirical performance models

A problem instance: a list of **features** $z = [z_1, \dots, z_m]^T$, drawn from a *feature space* \mathcal{F} ; the features are computable (a problem-specific code that extracts characteristics for an instance).

The *configuration space* Θ of a parameterized algorithm with parameters $\theta_1, \dots, \theta_k$ with domains $\Theta_1, \dots, \Theta_k$:

- ▶ a subset of the cross-product of parameter domains
 $\Theta \subseteq \Theta_1 \times \dots \times \Theta_k$; the elements of Θ : **configurations** (instantiations of the algorithm's parameters)

The *input space*: $\mathcal{I} = \Theta \times \mathcal{F}$.

Given an algorithm A with configuration space Θ and a distribution of instances with feature space \mathcal{F} ,
an **empirical performance model (EPM)** is a stochastic process $f : \mathcal{I} \rightarrow \Delta(\mathbb{R})$ that defines a probability distribution over performance measures, for each combination of a **configuration** $\theta \in \Theta$ of A and an instance with **features** $z \in \mathcal{F}$.

Empirical performance models

To construct an EPM for algorithm A with configuration space Θ on an instance set Π

- ▶ run A on various combinations of configurations $\theta_i \in \Theta$ and instances $\pi_i \in \Pi$, and record the performance values y_i
- ▶ record a $p = k + m$ -dimensional vector of *predictor variables* $x_i = [\theta_i^T, z_i^T]^T$, θ_i the k -dimensional parameter **configuration**, z_i the m -dimensional **feature vector** of the instance used in the i^{th} run
- ▶ the training data for the regression models: $\{(x_1, y_1), \dots, (x_n, y_n)\}$; X the $n \times p$ matrix containing $[x_1, \dots, x_n]^T$ (*design matrix*), y the vector of performance values $[y_1, \dots, y_n]^T$.

Predict log runtime: a log-transformation is useful for large variation in runtimes for hard combinatorial problems. Discard the input dimensions constant across all training data points and normalize the remaining ones (+ disregard missing values).

Problem-specific instance features for SAT ⁷

Problem Size Features:

- 1-2. **Number of variables and clauses in original formula (trivial):** denoted v and c , respectively
- 3-4. **Number of variables and clauses after simplification with SATLite (cheap):** denoted v' and c' , respectively
- 5-6. **Reduction of variables and clauses by simplification (cheap):** $(v-v')/v'$ and $(c-c')/c'$
- 7. **Ratio of variables to clauses (cheap):** v/c'

Variable-Clause Graph Features:

- 8-12. **Variable node degree statistics (expensive):** mean, variation coefficient, min, max, and entropy
- 13-17. **Clause node degree statistics (cheap):** mean, variation coefficient, min, max, and entropy

Variable Graph Features (expensive):

- 18-21. **Node degree statistics:** mean, variation coefficient, min, and max
- 22-26. **Diameter*:** mean, variation coefficient, min, max, and entropy

Clause Graph Features (expensive):

- 27-31. **Node degree statistics:** mean, variation coefficient, min, max, and entropy
- 32-36. **Clustering Coefficient*:** mean, variation coefficient, min, max, and entropy

Balance Features:

- 37-41. **Ratio of positive to negative literals in each clause (cheap):** mean, variation coefficient, min, max, and entropy
- 42-46. **Ratio of positive to negative occurrences of each variable (expensive):** mean, variation coefficient, min, max, and entropy
- 47-49. **Fraction of unary, binary, and ternary clauses (cheap)**

Proximity to Horn Formula (expensive):

- 50. **Fraction of Horn clauses**
- 51-55. **Number of occurrences in a Horn clause for each variable:** mean, variation coefficient, min, max, and entropy

DPLL Probing Features:

- 56-60. **Number of unit propagations (expensive):** computed at depths 1, 4, 16, 64 and 256

- 61-62. **Search space size estimate (cheap):** mean depth to contradiction, estimate of the log of number of nodes

LP-Based Features (moderate):

- 63-66. **Integer slack vector :** mean, variation coefficient, min, and max
- 67. **Ratio of integer vars in LP solution**
- 68. **Objective value of LP solution**

Local Search Probing Features, based on 2 seconds of running each of SAPS and GSAT (cheap):

- 69-78. **Number of steps to the best local minimum in a run:** mean, median, variation coefficient, 10th and 90th percentiles
- 79-82. **Average improvement to best in a run:** mean and coefficient of variation of improvement per step to best solution
- 83-86. **Fraction of improvement due to first local minimum:** mean and variation coefficient
- 87-90. **Best solution:** mean and variation coefficient

Clause Learning Features* (based on 2 seconds of running ZCHAFF_RANDOM; cheap):

- 91-99. **Number of learned clauses:** mean, variation coefficient, min, max, 10%, 25%, 50%, 75%, and 90th quantiles
- 100-108. **Length of learned clause:** mean, variation coefficient, min, max, 10%, 25%, 50%, 75%, and 90th quantiles

Survey Propagation Features* (moderate)

- 109-117. **Confidence of survey propagation:** For each variable, compute the higher of $P(true)/P(false)$ or $P(false)/P(true)$. Then compute statistics across variables: mean, variation coefficient, min, max, 10%, 25%, 50%, 75%, and 90th quantiles
- 118-126. **Unconstrained variables:** For each variable, compute $P(unconstrained)$. Then compute statistics across variables: mean, variation coefficient, min, max, 10%, 25%, 50%, 75%, and 90th quantiles

Timing Features*

- 127-138. **CPU time required for feature computation:** one feature for each of 12 subsets of features (see text for details)

Problem-specific instance features

- ▶ *Probing features*: execute a cheap algorithm for the instance and keep track of several statistics (e.g. #clauses a DPLL SAT solver learns in 2 sec)

Probing features based on one type of algorithm (e.g, LS) are often useful for predicting the performance of another type of algorithm (e.g., tree search).

- ▶ *Timing features*: the time for computing various features

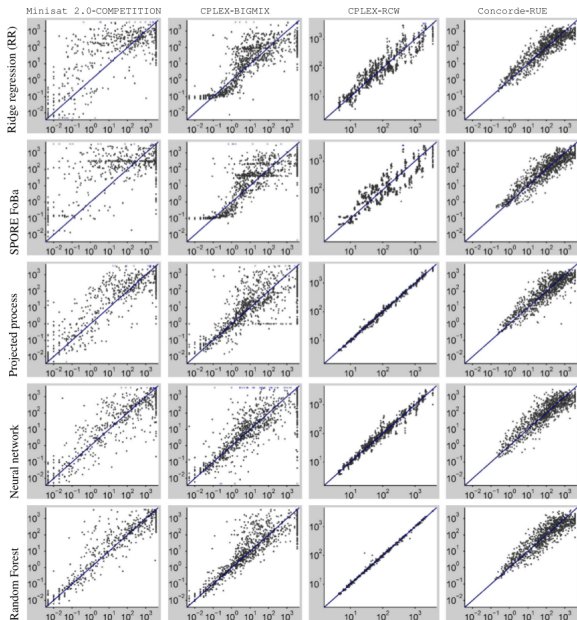
Performance predictions

- ▶ Complementary quantitative metrics to evaluate mean predictions μ_1, \dots, μ_n and predictive variances $\sigma_1^2, \dots, \sigma_n^2$ given true performance values y_1, \dots, y_n .
 - ▶ *Root mean squared error (RMSE)*: $\sqrt{1/n \sum_{i=1}^n (y_i - \mu_i)^2}$
 - ▶ *Pearson's correlation coefficient (CC)*: $\frac{\sum_{i=1}^n (\mu_i y_i) - n \cdot \bar{\mu} \cdot \bar{y}}{(n-1) \cdot s_\mu \cdot s_y}$, \bar{x} sample mean, s_x standard deviation of x
- ▶ 10-fold cross-validation, report means of the measures across the 10 folds; statistical significance using a Wilcoxon signed-rank test (paired - cross-validation folds are correlated)
- ▶ data and software:
<http://www.cs.ubc.ca/labs/beta/Projects/EPMS/>

Quantitative comparison of models for runtime predictions on previously unseen instances ⁷

Domain	RMSE						Time to learn model (s)					
	RR	SP	NN	PP	RT	RF	RR	SP	NN	PP	RT	RF
Minisat 2.0-COMPETITION	1.01	1.25	0.62	0.92	0.68	0.47	6.8	28.08	21.84	46.56	20.96	22.42
Minisat 2.0-HAND	1.05	1.34	0.63	0.85	0.75	0.51	3.7	7.92	6.2	44.14	6.15	5.98
Minisat 2.0-RAND	0.64	0.76	0.38	0.55	0.5	0.37	4.46	7.98	10.81	46.09	7.15	8.36
Minisat 2.0-INDU	0.94	1.01	0.78	0.86	0.71	0.52	3.68	7.82	5.57	48.12	6.36	4.42
Minisat 2.0-SWV-IBM	0.53	0.76	0.32	0.52	0.25	0.17	3.51	6.35	4.68	51.67	4.8	2.78
Minisat 2.0-IBM	0.51	0.71	0.29	0.34	0.3	0.19	3.2	5.17	2.6	46.16	2.47	1.5
Minisat 2.0-SWV	0.35	0.31	0.16	0.1	0.1	0.08	3.06	4.9	2.05	53.11	2.37	1.07
CryptoMinisat-INDU	0.94	0.99	0.94	0.9	0.91	0.72	3.65	7.9	5.37	45.82	5.03	4.14
CryptoMinisat-SWV-IBM	0.77	0.85	0.66	0.83	0.62	0.48	3.5	10.83	4.49	48.99	4.75	2.78
CryptoMinisat-IBM	0.65	0.96	0.55	0.56	0.53	0.41	3.19	4.86	2.59	44.9	2.41	1.49
CryptoMinisat-SWV	0.76	0.78	0.71	0.66	0.63	0.51	3.09	4.62	2.09	53.85	2.32	1.03
SPEAR-INDU	0.95	0.97	0.85	0.87	0.8	0.58	3.55	9.53	5.4	45.47	5.52	4.25
SPEAR-SWV-IBM	0.67	0.85	0.53	0.78	0.49	0.38	3.49	6.98	4.32	48.48	4.9	2.82
SPEAR-IBM	0.6	0.86	0.48	0.66	0.5	0.38	3.18	5.77	2.58	45.72	2.5	1.56
SPEAR-SWV	0.49	0.58	0.48	0.44	0.47	0.34	3.09	6.24	2.09	56.09	2.38	1.13
tnm-RANDSAT	1.01	1.05	0.94	0.93	1.22	0.88	3.79	8.63	6.57	46.21	7.64	5.42
SAPS-RANDSAT	0.94	1.09	0.73	0.78	0.86	0.66	3.81	8.54	6.62	49.33	6.59	5.04
CPLEX-BIGMIX	2.7E8	0.93	1.02	1	0.85	0.64	3.39	8.27	4.75	41.25	5.33	3.54
Gurobi-BIGMIX	1.51	1.23	1.41	1.26	1.43	1.17	3.35	5.12	4.55	40.72	5.45	3.69
SCIP-BIGMIX	4.5E6	0.88	0.86	0.91	0.72	0.57	3.43	5.35	4.48	39.51	5.08	3.75
lp_solve-BIGMIX	1.1	0.9	0.68	1.07	0.63	0.5	3.35	4.68	4.62	43.27	2.76	4.92
CPLEX-CORLAT	0.49	0.52	0.53	0.46	0.62	0.47	3.19	7.64	5.5	27.54	4.77	3.4
Gurobi-CORLAT	0.38	0.44	0.41	0.37	0.51	0.38	3.21	5.23	5.52	28.58	4.71	3.31
SCIP-CORLAT	0.39	0.41	0.42	0.37	0.5	0.38	3.2	7.96	5.52	26.89	5.12	3.52
lp_solve-CORLAT	0.44	0.48	0.44	0.45	0.54	0.41	3.25	5.06	5.49	31.5	2.63	4.42
CPLEX-RCW	0.25	0.29	0.1	0.03	0.05	0.02	3.11	7.53	5.25	25.84	4.81	2.66
CPLEX-REG	0.38	0.39	0.44	0.38	0.54	0.42	3.1	6.48	5.28	24.95	4.56	3.65
CPLEX-CR	0.46	0.58	0.46	0.43	0.58	0.45	4.25	11.86	11.19	29.92	11.44	8.35
CPLEX-CRR	0.44	0.54	0.42	0.37	0.47	0.36	5.4	18.43	17.34	35.3	20.36	13.19
LK-H-RUE	0.61	0.63	0.64	0.61	0.89	0.67	4.14	1.14	12.78	22.95	11.49	11.14
LK-H-RCE	0.71	0.72	0.75	0.71	1.02	0.76	4.19	2.7	12.93	24.78	11.54	10.79
LK-H-TSPLIB	9.55	1.11	1.77	1.3	1.21	1.06	1.61	3.02	0.51	4.3	0.17	0.11
Concorde-RUE	0.41	0.43	0.43	0.42	0.59	0.45	4.18	3.6	12.7	22.28	10.79	9.9
Concorde-RCE	0.33	0.34	0.34	0.34	0.46	0.35	4.17	2.32	12.68	24.8	11.16	10.18
Concorde-TSPLIB	120.6	0.69	0.99	0.87	0.64	0.52	1.54	2.66	0.47	4.26	0.22	0.12

Visual comparison of models for runtime predictions ⁷



Content

Motivation

Meta-learning

Algorithm runtime prediction (for SAT)

Generalizations to other domains

Sorting

- ▶ Use **dynamic programming** to estimate the mapping S via a Markov decision process ⁸
 - ▶ feature: the length of the sequence
 - ▶ generate rules/optimal policies showing which of the algorithms (*InsertionSort*, *MergeSort*, *Quick-Sort*) should be used (depending on the length of sequence)

⁸Lagoudakis, M., Litman, M., Parr, R. *Selecting the right algorithm*, 2001

Sorting

- ▶ A **learning approach** for the mapping S , more sorting algorithms, additional features to characterize a sequence (the degree of presortedness) ⁹
 - ▶ \mathcal{P} : 4 classes of random sequences of varying size and degree of presortedness (43195 instances)
 - ▶ \mathcal{A} : *InsertionSort*, *ShellSort*, *HeapSort*, *MergeSort*, *QuickSort*
 - ▶ performance: algorithm rank based on CPU time
 - ▶ \mathcal{F} : length of sequence, measures of presortedness (#inversions, the length of the longest ascending subsequence, etc.)
 - ▶ C4.5 to produce rules, and two classifiers; a dynamic approach was also adopted for the recursive sorting algorithms.

Over 90% accuracy in selecting the fastest sorting algorithm.

⁹Guo, H. *Algorithm selection for sorting and probabilistic inference: A machine learning-based approach*, '03

Learning to select/switch between algorithms online

Learning to select a sorting algorithm at each node

- ▶ Keep track of a state (e.g., length of sequence left to be sorted recursively)
- ▶ Choose the algorithm to use for subtree - e.g., *QuickSort* for long sequences, *InsertionSort* for short ones

Constraint Satisfaction

- ▶ The *empirical hardness* of SAT instances: the existence of algorithm-independent *phase transitions* for random 3-SAT problems: when the ratio of #clauses to #variables is approximately 4.26, the instance becomes hard
- ▶ Identify features that correlate with the empirical hardness of problem instances

Constraint Satisfaction

The relationship between the features of an instance and the time of the algorithm ¹⁰

- ▶ \mathcal{P} : 8544 instances of the winner determination problem, generated randomly
- ▶ \mathcal{A} : CPLEX 7.1
- ▶ performance: computation time (log)
- ▶ \mathcal{F} : 25 features
 - ▶ statistical characteristics of the graph: degree statistics, edge density, average minimum path length, etc.
 - ▶ features derived from solving the LP relaxation of the problem (norms of the int slack variables)
 - ▶ statistics of the prices in the auction problem
- ▶ The mapping S was learned using a linear regression model, 2nd-degree polynomial regression and spline models.

¹⁰Leyton-Brown, K., et al. *Learning the empirical hardness of optimization problems: The case of combinatorial auctions*, '02

Algorithm portfolios

- ▶ Combine algorithms with high average running times to form an **algorithm portfolio**.
- ▶ To construct algorithm portfolios:
 - ▶ train a model for each algorithm
 - ▶ given an instance,
 - ▶ compute feature values
 - ▶ predict each algorithm's running time (using runtime models)
 - ▶ run the algorithm predicted to be fastest

Portfolio-based Algorithm Selection for SAT

- ▶ No single "dominant" SAT solver; different solvers perform best on different instances
- ▶ Make this decision online on a per-instance basis

SATzilla¹² an automated approach for constructing per-instance algorithm portfolios for SAT that use empirical hardness models to choose among their constituent solvers

- ▶ SAT algorithms: *tree-search algorithms, LS, resolution-based approaches*

¹²Xu et al. SATzilla: portfolio-based algorithm selection for SAT, 2008

Algorithm selection vs. algorithm configuration

- ▶ **Algorithm selection** operates on finite (small) sets of algorithms.

Algorithm configuration operates on the combinatorial space of an algorithm's parameter settings.

- ▶ **AClib** a benchmark library for instances of the Algorithm Configuration problem <http://aclib.net>
ASlib a benchmark library for Algorithm Selection <http://www.coseal.net/aslib/>
 - ▶ experiments on AClib - more costly than experiments on ASlib
- ▶ ASlib and AClib can be combined
 - ▶ generate actual performance data based on the resources in AClib, and
 - ▶ then create an ASlib scenario which selects between different solver configurations on a per-instance basis.

Bibliography

- ▶ F. Hutter et al. *Algorithm runtime prediction: Methods & evaluation*, Artificial Intelligence 206, 79-111, 2014
- ▶ M. Feurer et al. *Efficient and Robust Automated Machine Learning*, NIPS 28, 2962-2970, 2015
- ▶ K. Smith-Miles. *Cross-Disciplinary Perspectives on Meta-Learning for Algorithm Selection*, ACM Computing Surveys 41(1), 2009
- ▶ L. Kotthoff. *Algorithm Selection for Combinatorial Search Problems: A survey*, Data Mining and Constraint Programming, 149-190, 2016
- ▶ K. Leyton-Brown et al. *Empirical Hardness Models for Combinatorial Auctions*, 2005