Latent Class Models for Algorithm Portfolio Methods

Bryan Silverthorn and Risto Miikkulainen

Department of Computer Science
The University of Texas at Austin

13 July 2010

Our setting: hard computational problems

Many **important** computational questions, such as satisfiability (**SAT**), are **intractable** in the worst case.

Definition (SAT)

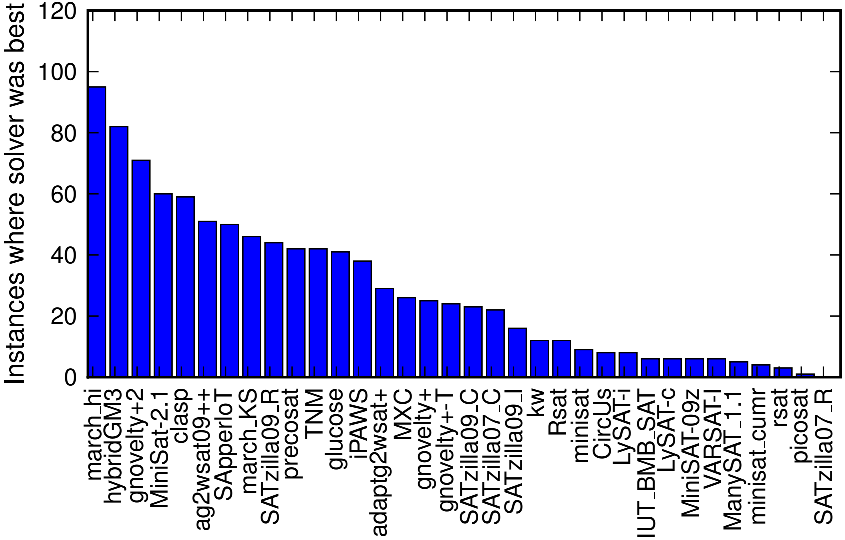
Does a truth assignment exist that makes a given Boolean expression true?

But **heuristics** often work: enormous problem instances can be solved.

> *SAT solvers are now in routine use for applications such as hardware verification... with up to a million variables.*

(Kautz and Selman, 2007)

Which solver do you choose? (2009 SAT competition)



Algorithm portfolios: what and why

Definition

An algorithm portfolio is

- a **pool** of algorithms ("solvers") and
- a **method** for scheduling their execution.

Portfolios can

- **reduce effort** by choosing solvers automatically, and
- **improve performance** by allocating resources more effectively.

Existing portfolios, such as SATzilla (Xu et al. 2008), often use **classifiers** trained on **feature information** to predict solver performance.

How should we predict solver performance?

Research Questions

- **What predictions** can we make with minimal information?
- **What assumptions** are needed to make useful predictions?
- **Do they hold** sufficiently well in practice?

To explore these questions, we will **build** unifying **generative models** of solver behavior and **evaluate** them in the SAT domain.

Assumptions that make modeling easier

Outcomes of runs are discrete, few, and fixed.

Utilities of outcomes are known.

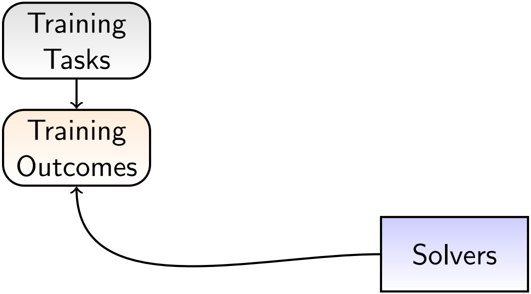
Durations of runs are discrete, few, and fixed.

Learning is offline, but **action** is online.

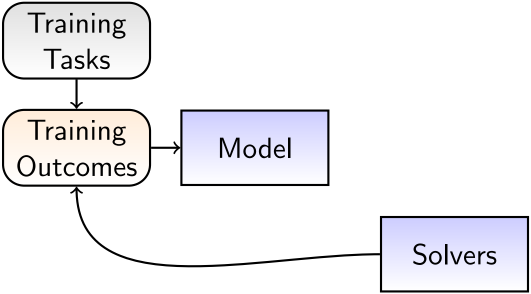
Tasks are drawn IID from some distribution.

Information is obtained from outcomes alone.

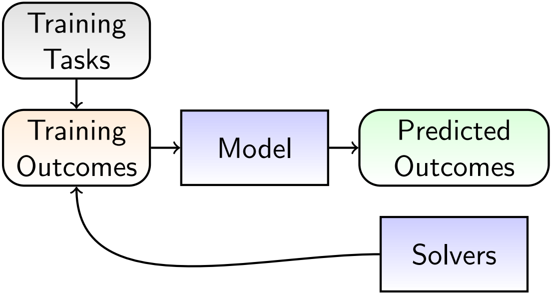
Architecture of a model-based portfolio



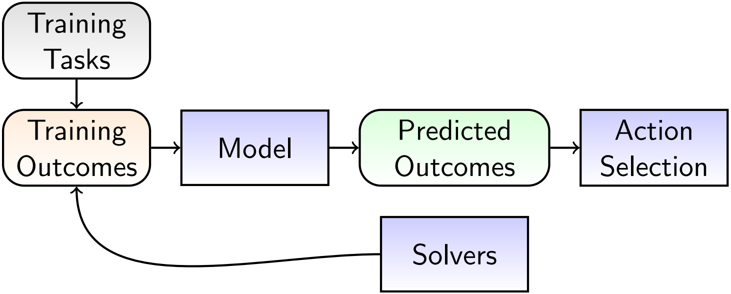
Architecture of a model-based portfolio



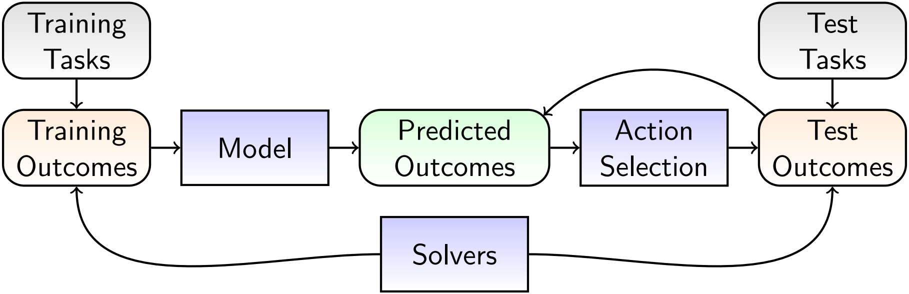
Architecture of a model-based portfolio



Architecture of a model-based portfolio



Architecture of a model-based portfolio



Basic structure in solver behavior

Inter-**algorithm correlations**: solvers can be (dis)similar.

Example

"If solver X yielded outcome A on this task, solver Y likely will as well."

Inter-**task correlations**: tasks can be (dis)similar.

Example

"If solver X yielded outcome A on task 1, it likely will on task 2 as well."

Inter-**duration correlations**: runs can have (dis)similar outcomes.

Example

"If solver X did not quickly yield outcome A on this task, it never will."

Conditional independence in solver behavior

The outcome of a solver run is a function of only three inputs:

- the **task** on which it is executed,
- the **duration** of the run, and
- the **seed** of any internal pseudorandom sequence.

This strong **local independence** suggests a possible model:

- take actions to be solver-duration **pairs**, and
- assume that **tasks cluster** into classes.

Classes then capture the basic **three aspects** of solver behavior.

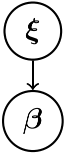
Multinomial latent class model of search

Model

Key

$\boldsymbol{\xi}$ class prior

Multinomial latent class model of search



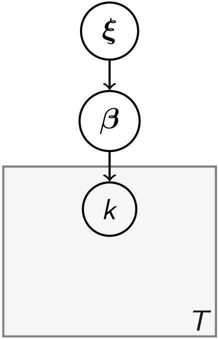
Model

$$\boldsymbol{\beta} \sim \mathrm{Dir}(\boldsymbol{\xi})$$

Key

$\boldsymbol{\xi}$	class prior
$\boldsymbol{\beta}$	class distr.

Multinomial latent class model of search



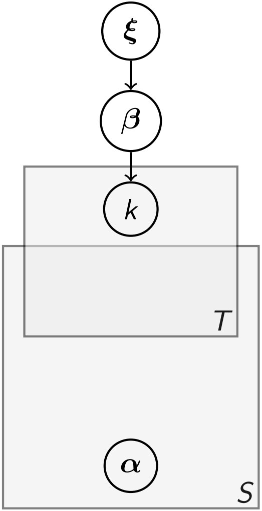
Model

$$\boldsymbol{\beta} \sim \mathrm{Dir}(\boldsymbol{\xi})$$
$$k_t \sim \mathrm{Mult}(\boldsymbol{\beta}) \qquad t \in 1 \ldots T$$

Key

$\boldsymbol{\xi}$	class prior
$\boldsymbol{\beta}$	class distr.
k	class
T	tasks

Multinomial latent class model of search



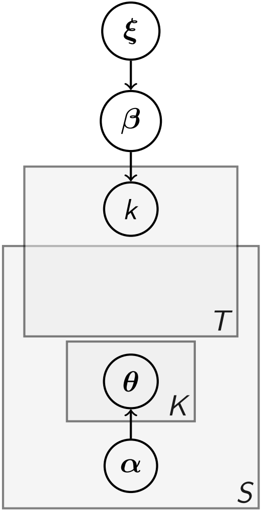
Model

$$
\begin{aligned}
\boldsymbol{\beta} &\sim \mathrm{Dir}(\boldsymbol{\xi}) \\
k_t &\sim \mathrm{Mult}(\boldsymbol{\beta}) \qquad\qquad t \in 1 \ldots T
\end{aligned}
$$

Key

$\boldsymbol{\xi}$	class prior	$\boldsymbol{\alpha}$	outcome prior
$\boldsymbol{\beta}$	class distr.		
k	class		
T	tasks	S	actions

Multinomial latent class model of search



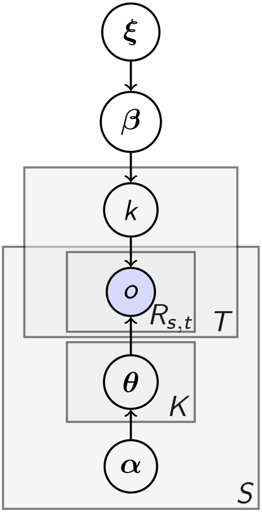
Model

$$
\begin{aligned}
\beta &\sim \mathrm{Dir}(\xi) & & \\
k_t &\sim \mathrm{Mult}(\beta) & t &\in 1 \dots T \\
\theta_{s,k} &\sim \mathrm{Dir}(\alpha_s) & s &\in 1 \dots S \\
& & k &\in 1 \dots K
\end{aligned}
$$

Key

ξ	class prior	α	outcome prior
β	class distr.	θ	outcome distr.
k	class		
T	tasks	S	actions
K	classes		

Multinomial latent class model of search



Model

$$
\begin{aligned}
\boldsymbol{\beta} &\sim \mathrm{Dir}(\boldsymbol{\xi}) & & \\
k_t &\sim \mathrm{Mult}(\boldsymbol{\beta}) & t &\in 1 \dots T \\
\boldsymbol{\theta}_{s,k} &\sim \mathrm{Dir}(\boldsymbol{\alpha}_s) & s &\in 1 \dots S \\
& & k &\in 1 \dots K \\
o_{t,s,r} &\sim \mathrm{Mult}(\boldsymbol{\theta}_{s,k_t}) & t &\in 1 \dots T \\
& & s &\in 1 \dots S \\
& & r &\in 1 \dots R_{s,t}
\end{aligned}
$$

Key

$\boldsymbol{\xi}$	class prior	$\boldsymbol{\alpha}$	outcome prior
$\boldsymbol{\beta}$	class distr.	$\boldsymbol{\theta}$	outcome distr.
k	class	o	outcome
T	tasks	S	actions
K	classes	R	runs

Burstiness: another important aspect of solver behavior

Definition

Burstiness is the tendency of some random events to recur.

Solver **outcomes recur**—for some solvers more than others.

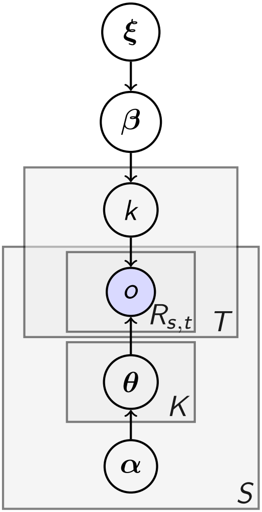
Example

"If solver X yields outcome A on this task, it will again; not so for Y."

Deterministic solvers are entirely bursty. **Randomized** solvers are less so.

Burstiness also appears in **text data**. The Dirichlet compound multinomial (DCM) distribution has modeled it well in that domain. (Madsen et al., 2005)

Multinomial latent class model of search



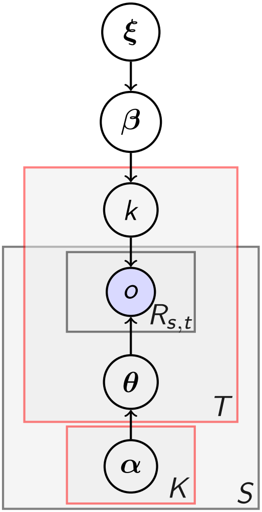
Model

$$
\begin{aligned}
\beta &\sim \mathrm{Dir}(\xi) & & \\
k_t &\sim \mathrm{Mult}(\beta) & t &\in 1 \ldots T \\
\theta_{s,k} &\sim \mathrm{Dir}(\alpha_s) & s &\in 1 \ldots S \\
& & k &\in 1 \ldots K \\
o_{t,s,r} &\sim \mathrm{Mult}(\theta_{s,k_t}) & t &\in 1 \ldots T \\
& & s &\in 1 \ldots S \\
& & r &\in 1 \ldots R_{s,t}
\end{aligned}
$$

Key

ξ	class prior	α	outcome prior
β	class distr.	θ	outcome distr.
k	class	o	outcome
T	tasks	S	actions
K	classes	R	runs

DCM (bursty) latent class model of search



Model

$$
\begin{aligned}
\beta &\sim \mathrm{Dir}(\xi) & & \\
k_t &\sim \mathrm{Mult}(\beta) & t &\in 1 \dots T \\
\theta_{t,s} &\sim \mathrm{Dir}(\alpha_{s,k_t}) & s &\in 1 \dots S \\
& & t &\in 1 \dots T \\
o_{t,s,r} &\sim \mathrm{Mult}(\theta_{t,s}) & t &\in 1 \dots T \\
& & s &\in 1 \dots S \\
& & r &\in 1 \dots R_{s,t}
\end{aligned}
$$

Key

ξ	class prior	α	outcome root
β	class distr.	θ	outcome distr.
k	class	o	outcome
T	tasks	S	actions
K	classes	R	runs

Greedy, discounted selection

One efficient approach is to choose the next action according to **immediate expected utility** without regard to later actions.

This approach gives us

- a **hard** policy that chooses the expected-best action, and
- a **soft** policy that draws actions proportional to expected utility.

Actions are solver-**duration** pairs: they have wildly different costs.

An obvious response is to reduce an action's expected utility by its cost, **discounting** by γ^c for a c-second run and factor γ.

Experimental procedure

In our experiments, we use

- **every individual solver** from the latest SAT competition, and
- **every problem instance** from its three benchmark collections;

in repeated trials, we

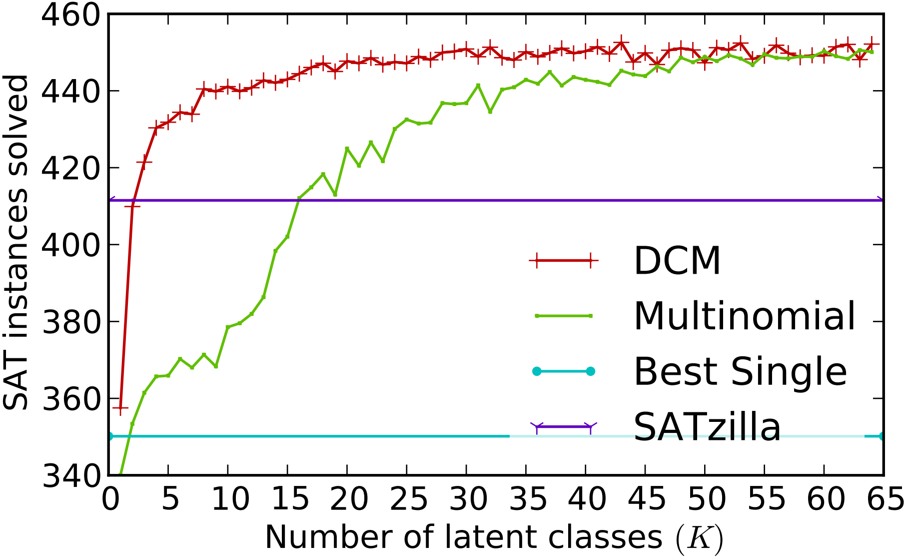
- **run** the solvers on a randomly-drawn **training set**,
- **fit** a model to that **training data**, and then
- **run** a portfolio using that model on the remaining **test set**.

Empirical Questions

For each combination of model and action selection policy,

- how does its **performance** compare to its **subsolvers**?
- how does its **performance** compare to that of **other portfolios**?

Portfolio performance (on the `random` collection)



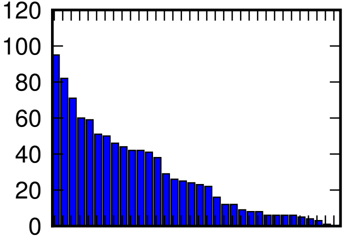
Recapitulation

These results suggest that

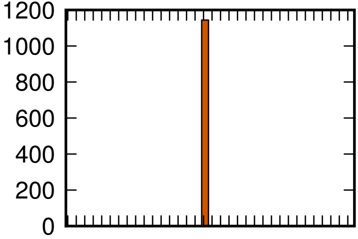
- models **can capture** useful patterns given little information, and
- these latent class models **can be applied** to a portfolio in practice.

Research in progress aims to

- **extend these models** to capture dynamic information, and
- **improve action planning** to better exploit their predictions.



→



Thanks!—Questions?

References I

Matteo Gagliolo and Jurgen Schmidhuber.
Learning Restart Strategies.
IJCAI 2007.

Carla Gomes and Bart Selman.
Algorithm Portfolio Design: Theory vs. Practice.
UAI 1997.

Eric Horvitz, Ruan Yongshao, Carla Gomes, Henry Kautz, Bart Selman, and David Chickering.
A Bayesian Approach to Tackling Hard Computational Problems.
UAI 2001.

Bernardo Huberman, Rajan Lukose, and Tad Hogg.
An Economics Approach to Hard Computational Problems.
Science, 275(5296), 1997.

References II

Frank Hutter, Domagoj Babić, Holger H. Hoos, and Alan J. Hu.
Boosting Verification by Automatic Tuning of Decision Procedures.
FMCAD 2007.

Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle.
ParamILS: An Automatic Algorithm Configuration Framework.
Journal of Artificial Intelligence Research, 2009.

Henry Kautz and Bart Selman.
The state of SAT.
Discrete Applied Mathematics, 12, 2007.

Ashiqur KhudaBukhsh, Lin Xu, Holger H. Hoos, Kevin Leyton-Brown.
SATenstein: Automatically Building Local Search SAT Solvers From Components.
IJCAI 2009.

References III

Rasmus Madsen, David Kauchak, and Charles Elkan.
Modeling Word Burstiness Using the Dirichlet Distribution.
ICML 2005.

David Mimno and Andrew McCallum.
Topic Models Conditioned on Arbitrary Features with Dirichlet-multinomial Regression.
UAI 2008.

Mladen Nikolić, Filip Marić, and Predrag Janičić.
Instance-Based Selection of Policies for SAT Solvers.
SAT 2009.

Eugene Nudelman, Kevin Leyton-Brown, Holger H. Hoos, Alex Devkar, and Yoav Shoham.
Understanding Random SAT: Beyond the Clauses-to-Variables Ratio.
CP 2004.

References IV

J. R. Rice.
The Algorithm Selection Problem.
Advances in Computers, 15, 1976.