In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence
2010.

Latent Class Models for Algorithm Portfolio Methods

Bryan Silverthorn and **Risto Miikkulainen**
Department of Computer Science
The University of Texas at Austin
{bsilvert,risto}@cs.utexas.edu

Abstract

Different solvers for computationally difficult problems such as satisfiability (SAT) perform best on different instances. *Algorithm portfolios* exploit this phenomenon by predicting solvers' performance on specific problem instances, then shifting computational resources to the solvers that appear best suited. This paper develops a new approach to the problem of making such performance predictions: natural generative models of solver behavior. Two are proposed, both following from an assumption that problem instances cluster into *latent classes*: a mixture of multinomial distributions, and a mixture of Dirichlet compound multinomial distributions. The latter model extends the former to capture *burstiness*, the tendency of solver outcomes to recur. These models are integrated into an algorithm portfolio architecture and used to run standard SAT solvers on competition benchmarks. This approach is found competitive with the most prominent existing portfolio, SATzilla, which relies on domain-specific, hand-selected problem features; the latent class models, in contrast, use minimal domain knowledge. Their success suggests that these models can lead to more powerful and more general algorithm portfolio methods.

Introduction

Heuristic solvers have become an essential tool for tackling problems that appear otherwise intractable. Solvers for the satisfiability problem (SAT), for example, verify hardware designs whose encodings involve millions of variables (Kautz and Selman 2007). Success has led to a profusion of heuristics. None, however, dominates empirical comparisons; each has strengths and weaknesses.

Algorithm portfolio methods (Huberman, Lukose, and Hogg 1997) use information about solvers and problem instances to allocate computational resources among multiple solvers, attempting to maximize the time spent on those well suited to each instance. Portfolios have proved effective in SAT (Gomes and Selman 2001; Xu et al. 2008) as well as directly in application areas such as planning (Roberts and Howe 2006) and scheduling (Cicirello and Smith 2005).

An algorithm portfolio must decide which solvers to run and for how long to run them. These decisions rely on expectations about solver behavior. Consider the situation when a solver fails to find a solution. Should the same solver be run again on the same instance? How likely is that solver to succeed in that future run, given its earlier failure? It may be nondeterministic, or require a longer run, or be unable to solve the instance at all. How can a portfolio method make these judgments?

This paper will argue that knowledge of solver behavior can be captured by a well-chosen set of statistical assumptions regarding solver-problem interaction. Those assumptions can be used to learn predictable aspects of solver behavior—such as how likely a solver is to succeed if it has previously failed—given data such as the successes and failures of solvers on many other problem instances.

Existing portfolio approaches include those that apply bandit methods (Gagliolo and Schmidhuber 2006), fixed execution schedules (Streeter, Golovin, and Smith 2007), and performance predictors built on domain-specific knowledge of problem instances (Nudelman et al. 2004). The latter are used heavily by SATzilla, the most prominent modern portfolio method for SAT (Xu et al. 2008; 2009).

Latent class models of solver behavior, developed in this paper, instead use solver performance data to identify groups of similar problem instances. Two models are proposed: a multinomial mixture that captures the basic correlations between solvers, runs, and problem instances, and a mixture of Dirichlet compound multinomial distributions that also captures the tendency of solver outcomes to recur.

These models are integrated into a general algorithm portfolio architecture outlined in the next section. Using greedy action selection, this portfolio method is evaluated with the solvers and benchmark collections published by the most recent comprehensive SAT competition. Despite minimal domain knowledge, this strategy beats baseline and non-portfolio methods and is competitive with SATzilla. Latent class models of solver behavior may thus lead to a new, more general, and more effective family of portfolio methods.

A Model-Based Portfolio Architecture

An algorithm portfolio schedules the execution of its constituent algorithms on previously-unseen instances of some difficult problem. Horvitz et al. (2001) suggested that problem solving should be centered on a probabilistic model of solver behavior. This paper proposes a portfolio architecture in this paradigm.

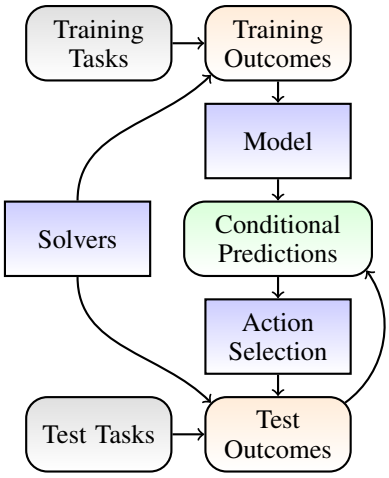

Figure 1: Solving instances of some difficult computational problem, such as SAT, using the algorithm portfolio architecture described in this paper: a model is fit to training data, its predictions are used to select solvers, and observed outcomes affect later predictions.

Architecture Components

This architecture employs three core components: a *portfolio* of algorithms; a *generative model*, which is fit to data on those algorithms' past performance, then used to predict their future performance; and a policy for *action selection*, which repeatedly chooses algorithms based on those predictions. These components interact with a common set of entities: *tasks* in the problem domain, *solvers* that take tasks as input, *runs* of solvers on tasks, *outcomes* of runs, and the *duration* of each run in processor time.

A model is a joint probability distribution over the random variables of interest. Run outcomes are treated as observed random variables. The model assumes the form of some underlying structure in those outcomes; this structure is encoded by unobserved, or *latent*, variables. As new outcomes are observed, the posterior distributions over the latent variables become increasingly informative.

Experiments in this paper are performed in the context of SAT, in which tasks are Boolean formulae, solvers are heuristic search algorithms, and a run's outcome is either "proved satisfiable or unsatisfiable" or "failed".

Architecture Assumptions

This paper focuses on a specific problem-solving setting whose five key assumptions are detailed below.

First, solvers are assumed to have discrete outcomes: they must terminate in one of only a small and fixed number of ways. While this assumption excludes problem domains such as the optimization of objective functions mapping to the real numbers, it includes many important domains as well, such as all decision problems.

Second, while no ranking of outcomes is assumed when modeling solver behavior, outcomes must be compared when action selection weighs its options. Assigning each outcome a numeric *utility* makes comparison easy, so let us assume a utility function $u : 1 \dots O \to [0, 1]$ mapping outcome indices to ordered real values in a fixed range, where O is the number of possible outcomes. This paper's utility function for SAT maps proofs of satisfiability or unsatisfiability to 1 and all other outcomes to 0.

Third, the possible durations of solver runs are limited to a small and fixed set of values. This restriction makes the portfolio less efficient, since a run of some duration N might be required when a duration $M < N$ would have sufficed, but it simplifies modeling.

Fourth, the model is fit to data offline, but actions are chosen online. Fitting involves two stages: solvers are executed multiple times on training instances, and model parameters are inferred from the resulting outcome data. Solvers and run durations are then selected as tasks are presented to the portfolio and outcomes are observed. Time spent in solver selection is therefore time not spent in solver execution.

Fifth, the outcomes of solver runs are the only data available to the model. In practice, other sources of information may exist, but these require additional domain knowledge to identify and exploit. This paper examines whether general patterns in solver behavior are sufficiently informative to drive a competitive portfolio.

Portfolio Operation

Figure 1 places the components of this architecture in the context of their operation under the assumptions above. Portfolio operation begins with offline training, in which (1) training tasks are drawn from the task distribution, (2) each solver is run many times on each training task, and (3) a model is fit to the outcomes observed in training.

In the test phase that follows, repeatedly, (1) a test task is drawn from the same task distribution, (2) the model predicts the likely outcomes of each solver, (3) the portfolio selects and runs a solver for some duration, (4) the run's outcome conditions later predictions, and (5) the process continues from (2) until a time limit expires. Task draws are assumed independent, an assumption often violated but nonetheless useful and common (Bishop 2006). Between different runs of some solver of some maximum duration, the only parameter that changes is the seed provided to the solver's pseudo-random number generator.

The remaining sections of this paper construct the key components of this architecture—a model of solver behavior, a policy for action selection, and a portfolio of complementary solvers—and evaluate their combination on the SAT domain.

Models of Solver Behavior

The portfolio architecture above relies on an accurate model of solver behavior. The following sections propose two related, natural models of solver behavior, and discuss how they can be fit efficiently to data.

Basic Structure in Solver Behavior

A model of solver behavior should capture predictable aspects of that behavior. The most basic include relation-

ships between tasks, between solvers, and between run durations. Tasks are related because solvers are likely to perform (dis)similarly on (dis)similar tasks; solvers are related because (dis)similar solvers are likely to perform (dis)similarly on many tasks; runs are related because a solver's runs of some duration are likely to have outcomes (dis)similar to runs of (dis)similar duration on the same task. The model introduced in the following section captures these patterns where they are present.

The Multinomial Latent Class Model

The mechanism of solver behavior suggests the model's outline. Each run's outcome is a function only of the input task, the duration of the run, and any randomness internal to the solver. Run outcomes are thus independent when conditioned on a task and solver, since they depend only on the solver's uncorrelated internal randomness. If similar tasks can be grouped into classes, runs' outcomes would remain nearly independent when conditioned on a class, and each solver's behavior on a class could be modeled separately.

This class-conditional independence can be expressed in a hierarchical mixture model in which the outcomes of a solver on all tasks in some class are sampled from a common multinomial distribution. The model is easily described as a generative process. For now, let the duration of solver runs be fixed. Assume that the number of classes K, the number of tasks T, the number of solvers S, and the number of action outcomes O are known. Let an indicator k_t of the class of each task t be drawn from a multinomial distribution Mult($\boldsymbol{\beta}$) over classes in the task set, and draw the outcome $o_{t,s,r}$ of run r of $R_{s,t}$ of solver s from a distribution Mult($\boldsymbol{\theta}_{s,k_t}$) over outcomes of that solver on tasks in class k_t. More formally:

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

Dirichlet priors, denoted by Dir and parameterized by $\boldsymbol{\xi} \in \mathbb{R}^{+K}$ and each $\boldsymbol{\alpha}_s \in \mathbb{R}^{+O}$, were placed over $\boldsymbol{\beta}$ and each $\boldsymbol{\theta}_s$.

The model above emerges naturally from the perspective on algorithm selection laid out across earlier sections, and it has exactly the form of models such as multivariate latent class analysis (McLachlan and Peel 2004; Bishop 2006).

The description above assumes a single, fixed solver run duration. This assumption is relaxed by treating the combinations of solvers and run durations as entirely separate actions—*solver-duration pairs*. To keep the number of actions manageable under this approach, possible run durations must be fixed and few, as assumed.

This multinomial mixture model is not complex, but it captures the three essential aspects of solver behavior. Classes link related tasks. Classes also link the outcomes of actions, since information about the class of a task is information about the distribution of every action's outcomes. Information from one action can thus affect the posterior probabilities of another action's outcomes. Since solver-duration pairs are treated as distinct actions, run durations are linked in the same way.

The DCM Latent Class Model

Solvers exhibit varied levels of determinism. Some are fully deterministic and should be run at most once; others are heavily randomized and should be run multiple times. The level of determinism exhibited by a solver also depends on the task distribution: any solver appears deterministic on a set of impossible or trivial tasks.

The multinomial distribution does not directly capture this aspect of solver behavior. Lessons from the area of text analysis can help to explain why. Consider, for example, a class of documents related to college sports. Individual documents focus on one or two teams, so a document that mentions a particular team is likely to mention that team again, even if that team name is rare overall. This phenomenon is described as *burstiness* in text (Katz 1996). The multinomial distribution models this class of documents poorly because the frequency of the name terms are not roughly the same in every document. Viewing run outcomes as terms and tasks as documents, solvers, too, exhibit this notion of burstiness.

The Dirichlet compound multinomial (DCM) distribution, also known as the multivariate Pólya distribution, has been used to model burstiness in text (Madsen, Kauchak, and Elkan 2005; Doyle and Elkan 2009), so it is a good candidate for modeling it in solver outcomes. The distribution is hierarchical, rooted in a Dirichlet distribution parameterized by $\boldsymbol{\alpha} \in \mathbb{R}^{+D}$, and can be interpreted as a draw of $\boldsymbol{\theta}$ from Dir($\boldsymbol{\alpha}$) followed by a draw of $\boldsymbol{x}$ from Mult($\boldsymbol{\theta}$).

Elkan (2006) noted that the equivalent Pólya-Eggenberger urn scheme better conveys the burstiness of the DCM. Imagine an urn from which colored balls are drawn repeatedly with replacement. After returning each ball, another ball of the same color is added to the urn, increasing the number of balls inside. If the urn initially contains many balls, the additional balls have little effect, but if the urn begins nearly empty, they make colors already drawn more likely be drawn again. The starting configuration of the urn corresponds to the parameter vector $\boldsymbol{\alpha}$.

The DCM distribution can be used to extend the multinomial latent class model by representing the per-solver components of each class as Dirichlet distributions, drawing each outcome from a multinomial distribution parameterized by a per-task Dirichlet draw. The model becomes:

$$\begin{array}{rcll} \boldsymbol{\beta} & \sim & \text{Dir}(\boldsymbol{\xi}), & \\ k_t & \sim & \text{Mult}(\boldsymbol{\beta}), & t \in 1 \dots T, \\ \boldsymbol{\theta}_{t,s} & \sim & \text{Dir}(\boldsymbol{\alpha}_{s,k_t}), & s \in 1 \dots S, \\ & & & t \in 1 \dots T, \\ o_{t,s,r} & \sim & \text{Mult}(\boldsymbol{\theta}_{t,s}), & t \in 1 \dots T, \\ & & & s \in 1 \dots S, \\ & & & r \in 1 \dots R_{s,t}. \end{array} \tag{2}$$

The graph representation of this model is presented in Figure 2. Fitting it to data means inferring the Dirichlet parameter vector $\boldsymbol{\alpha}_{s,k}$ for each of the S distributions in each of the K mixture components.

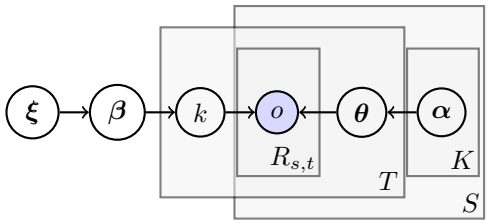

Figure 2: The DCM latent class model of solver behavior, in its graph representation, using standard plate notation (Bishop 2006). The model is structurally similar to the multinomial latent class model, but it additionally captures the bursty nature of some solvers' outcomes.

Fitting Solver Models to Data

Expectation maximization (EM) can be used to fit these models to data. Its details are covered extensively elsewhere (McLachlan and Peel 2004). For the DCM latent class model, the responsibility $\gamma_{t,k}$ of each class k for solver outcomes on task t is

$$\gamma_{t,k} = \frac{\beta_k \prod_s^S \text{DCM}(\boldsymbol{x}_{t,s}|\boldsymbol{\alpha}_{s,k})}{\sum_j^K \beta_j \prod_s^S \text{DCM}(\boldsymbol{x}_{t,s}|\boldsymbol{\alpha}_{s,j})}, \quad (3)$$

where $\boldsymbol{x}_{t,s} \in \mathbb{Z}^{+O}$ is the vector of counts defined by $x_{t,s,i} = \sum_r^{R_{s,t}} \delta(o_{t,s,r} = i)$. The corresponding formula for the multinomial model differs only in the inner distribution term.

During the maximization step, an estimate of the DCM parameter vector $\boldsymbol{\alpha}$ can be calculated from count vectors, such as each $\boldsymbol{x}$ above, using one of several procedures. The digamma-recurrence fixed-point iteration of Wallach (2008) is particularly efficient, and used here.

Conditional Prediction

In the test phase, the portfolio receives information about each new task from the outcomes of the solvers it executes on that task. This information narrows the likely class of the new task, leading to probability

$$p(o_{t,s,r} = i|\boldsymbol{o}_t) = \sum_{j=1}^{K} \gamma_{t,k} p(o_{t,s,r} = i|k_t = j, \boldsymbol{o}_t) \quad (4)$$

that the outcome of solver s on task t will be i, conditioned on the outcomes $\boldsymbol{o}_t$ of actions previously taken on task t. The term $\gamma_{t,k}$ is the responsibility (as in Equation 3). For the DCM model, the inner conditional probability is

$$p(o_{t,s,r} = i|k_t = j, \boldsymbol{o}_t) = \text{DCM}(\boldsymbol{o}_{t,s} + \boldsymbol{i}|\boldsymbol{\alpha}_{s,j} + \boldsymbol{o}_{t,s}), \quad (5)$$

where $\boldsymbol{i}$ is the one-of-O vector for i. For the multinomial model, this expression is simply $\theta_{s,j,i}$.

Model-Based Algorithm Selection

Predictions of solver performance are useful only if they can be used to run more appropriate solvers more often in realistic settings. This paper employs a simple, greedy scheme to select actions using the predictions of latent class models.

Greedy, Discounted Selection Policies

Among the simplest selection policies are those that use expected utility but ignore the potential utility of later actions. Such policies are efficient, and are easy to understand, implement, and test.

Runs of different durations have different computational costs, and a model-based selection policy should incorporate these costs into its decisions. A greedy approach that ignored cost would prefer long solver runs to short, even if multiple short runs would lead to higher overall utility on average. As in other settings (DeGroot 1970), later rewards are thus discounted according to a factor γ: utility w accrued after c seconds has present value $\gamma^c w$.

Two selection policies are evaluated. Both assume a particular fixed discount rate. The first policy, "hard" selection, deterministically selects the solver-duration pair with maximum expected discounted utility. The action that appears best, in other words, is always chosen. This method places the most trust in its model. The second policy, "soft" selection, nondeterministically selects a solver-duration pair with probability proportional to its expected discounted utility u'. The probability that action l is the selected action s is

$$p(s = l) = \frac{\sum_{i=1}^{O} p(o_{t,s,r} = i|\boldsymbol{o}_t) u'(i)}{\sum_{i=1}^{S} \sum_{j=1}^{O} p(o_{t,i,r} = j|\boldsymbol{o}_t) u'(j)}. \quad (6)$$

This method places less trust in its model. Evaluating two policies in this manner tests how sensitive the portfolio is to the details of selection.

Experiments

The core components of a model-based portfolio architecture have been developed in the sections above. In this section, the assembled architecture will be evaluated by applying it to SAT, an attractive test domain common in prior portfolio work (Gomes and Selman 2001; Streeter, Golovin, and Smith 2007).

The SAT Setting

SAT asks whether a given Boolean formula, typically expressed in conjunctive normal form (CNF), can be made true. These formulae often encode instances of concrete problem domains, such as planning (Kautz and Selman 1992), verification (Biere et al. 1999), and electronic design automation (Wood and Rutenbar 1997), among others (Gomes et al. 2007). Application areas therefore benefit from improved SAT methods.

SAT is an appealing testbed for portfolio research. Its solvers are diverse and complementary. Competitions held by the SAT research community lead to the open distribution of solvers, solver source, and task collections. Using such material from a competition mitigates the risk of cherry-picking an easy environment: competitions include representative modern solvers, as well as qualitatively different applied and synthetic tasks.

The latent class portfolios tested in this section use every solver from the main round of the 2009 competition—except for the SATzilla portfolios, which are presented separately—and are tested on all three of its benchmark collections:

`random` (570 generator-sampled instances), `crafted` (281 manually-selected instances), and `industrial` (292 real-world instances).

SATzilla is the most prominent SAT portfolio method and an impressive solver: in the 2009 competition, it won three of the nine categories and placed second in another two. It uses per-solver regression-based predictors of performance trained on $\approx$ 90 task features, such as statistics of the formula clause graph and of local search probes. Three variants of SATzilla competed in 2009, one trained for each of the three benchmark collections.

SATzilla performance numbers are included in the results below, with each variant tested on its associated collection. The comparison between methods, however, is inherently imperfect. SATzilla trains on thousands of training instances drawn from multiple benchmark collections, while the latent class portfolios learn from fewer instances drawn from the single test distribution. Furthermore, SATzilla employs a set of solvers chosen by its designers; this set does not include every solver in the 2009 competition, and it includes some that did not compete, such as recent automatically-discovered local search solvers (KhudaBukhsh et al. 2009). The SATzilla performance scores therefore provide only rough targets.

Since SATzilla applies the large body of research on properties of CNF formulae in SAT, its prediction scheme requires substantial domain knowledge. The latent class model approach, in contrast, makes more complex assumptions but uses minimal domain knowledge. Even a rough comparison between these different approaches is useful.

Methodology

The performance of several problem-solving strategies are compared: hard and soft selection policies coupled with the DCM and multinomial models across a range of K; random action selection, which provides a baseline; the best individual solvers, selected by an oracle with knowledge of each solver's true performance across each benchmark collection, which provide non-portfolio points of reference; and SATzilla, which represents the most successful alternative portfolio approach.

In each evaluation, the portfolio was built and operated as in Figure 1, and was allowed 5,000 seconds to solve each test task. This cutoff was chosen to match that used by the competition on the majority of its instances. Training and test tasks were drawn without replacement. As in the competition scoring system, SATSPEC, strategies are ranked according to the number of test tasks they are able to solve.

The means (μ) and standard deviations (σ) of the scores of at least 32 randomly-reinitialized evaluations were obtained for each configuration. The discount factor was $1-10^{-4}$, 12 run durations were linearly spaced from 2 to 5,000 seconds, and 16 restarts were made on each of 64 training tasks. Each evaluation's test set contained all of its collection's instances not used for training. Outcomes were simulated during evaluation using precomputed per-task solver distributions.

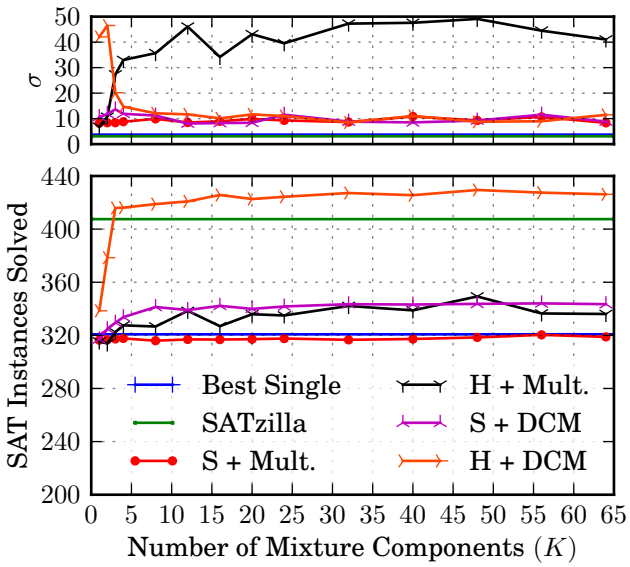

Figure 3: The number of tasks solved by selected strategies on `random`. Performance improves as the number of assumed latent classes, K, is increased, suggesting that class structure is an aspect of solver behavior useful to portfolio methods. All methods beat random selection, which is not shown. Portfolios using the DCM model beat those using the multinomial, and, at higher K, portfolios using hard selection ("H") beat those using soft selection ("S") and the same model.

Results

Figure 3 presents the connection between the assumed number of latent classes K and the performance of the portfolio employing each combination of model and selection policy. If the models capture useful patterns in solver behavior, we would expect performance to clearly improve with K, as it does. The behaviors of the different model-policy combinations are distinguishable. Model information is better exploited by the hard selection policy, but soft selection yields runs with lower variance at low K. In general, the DCM model performs best, and performs surprisingly well at $K = 1$; knowledge of solver burstiness is useful even when separate from that of latent class structure.

Table 1 presents the performance scores for every latent class portfolio combination at representative values of K across every benchmark collection. It also provides the performance of the random-selection baseline, the best single solvers, and, as a rough comparison, SATzilla. In general, the best-performing latent-class portfolio method is the combination of DCM mixture model and hard selection policy. Its performance substantially exceeds the random policy and all single solvers on `random`, noticeably beats the oracle-selected best single solver on `crafted`, and is only slightly worse than it on the challenging `industrial` set. The latent class portfolio architecture, in other words, is an effective system for solving SAT instances.

Overall, the DCM-based portfolio performs at least roughly comparably to SATzilla. These results suggest that

Method	SAT Instances Solved (σ)		
	random	crafted	indust.
Best Single	320.8 (3.7)	122.6 (3.8)	**153.5** (3.2)
Random	261.6 (4.8)	89.3 (3.5)	54.5 (3.4)
SATzilla	407.5 (3.1)	125.6 (3.3)	137.6 (3.3)
Soft + Mult.	319.3 (8.2)	116.9 (4.9)	136.2 (4.3)
Soft + DCM	342.2 (8.2)	118.2 (4.3)	138.1 (5.0)
Hard + Mult.	340.3 (45.7)	104.8 (9.7)	129.4 (8.0)
Hard + DCM	**424.2** (12.9)	**129.4** (6.5)	146.0 (5.6)

Table 1: Problem-solving performance under each selection policy and model on every benchmark collection, with K set to the number of training tasks. Hard selection combined with the DCM model is roughly competitive with SATzilla.

the latent class approach is a viable and promising alternative method for building algorithm portfolios.

Future Work

Three open research avenues are worth noting in particular. First, offline training is reasonable in domains with collections of benchmark instances, such as SAT, but efficient online inference could make model-based portfolios more widely applicable. Second, more powerful planning techniques could better exploit model information, and could potentially balance exploration and exploitation concerns in online settings. Third, instance features, such as those leveraged by SATzilla, could be utilized in latent class models as well. They would make the model more complex and less general, but would likely further improve performance.

Conclusions

Models of solver behavior improve algorithm portfolio methods by allowing them to predict solver performance more accurately. This paper proposed two latent class models as particularly appropriate: a multinomial mixture model of the outcomes of solver-duration pairs, and a DCM mixture model that additionally captures the burstiness of those outcomes. Each model was embedded in a portfolio of diverse SAT solvers and evaluated on competition benchmarks. Both models support effective problem solving, and the DCM-based portfolio is competitive with the prominent SATzilla portfolio. Portfolio methods are already a powerful approach to problem solving, and the modeling work in this paper lays the foundation for making them both more powerful and more general. This work is thus a step toward better solution strategies for some of the most difficult computational problems.

Acknowledgments

This research was supported in part by the NSF under grants EIA-0303609, IIS-0757479, and IIS-0915038, and by the THECB under grant 003658-0036-2007.

References

Biere, A.; Cimatti, A.; Clarke, E. M.; Fujita, M.; and Zhu, Y. 1999. Symbolic Model Checking using SAT procedures instead of BDDs. In *ACM/IEEE Design Autom. Conference*.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*.

Cicirello, V., and Smith, S. 2005. The Max K-Armed Bandit: a New Model for Exploration Applied to Search Heuristic Selection. In *AAAI*.

DeGroot, M. H. 1970. *Optimal Statistical Decisions*.

Doyle, G., and Elkan, C. 2009. Accounting for Word Burstiness in Topic Models. In *ICML*.

Elkan, C. 2006. Clustering Documents with an Exponential-Family Approximation of the Dirichlet Compound Multinomial Distribution. In *ICML*.

Gagliolo, M., and Schmidhuber, J. 2006. Learning dynamic algorithm portfolios. *AMAI*.

Gomes, C., and Selman, B. 2001. Algorithm portfolios. *AI*.

Gomes, C. P.; Kautz, H.; Sabharwal, A.; and Selman, B. 2007. *Satisfiability Solvers*.

Horvitz, E.; Ruan, Y.; Gomes, C. P.; Kautz, H. A.; Selman, B.; and Chickering, D. M. 2001. A Bayesian Approach to Tackling Hard Computational Problems. In *UAI*.

Huberman, B.; Lukose, R.; and Hogg, T. 1997. Economics Approach to Hard Computational Problems. *Science*.

Katz, S. 1996. Distribution of content words and phrases in text and language modelling. *NLE*.

Kautz, H., and Selman, B. 1992. Planning as Satisfiability. In *ECAI*.

Kautz, H., and Selman, B. 2007. The state of SAT. *DAM*.

KhudaBukhsh, A.; Xu, L.; Hoos, H.; and Leyton-Brown, K. 2009. SATenstein: Automatically Building Local Search SAT Solvers From Components. In *IJCAI*.

Madsen, R. E.; Kauchak, D.; and Elkan, C. 2005. Modeling Word Burstiness Using the Dirichlet Distribution. In *ICML*.

McLachlan, G., and Peel, D. 2004. *Finite Mixture Models*.

Nudelman, E.; Leyton-Brown, K.; Hoos, H. H.; Devkar, A.; and Shoham, Y. 2004. Understanding Random SAT: Beyond the Clauses-to-Variables Ratio. In *CP*.

Roberts, M., and Howe, A. 2006. Directing a Portfolio with Learning. In *AAAI*.

Streeter, M.; Golovin, D.; and Smith, S. F. 2007. Combining Multiple Heuristics Online. In *AAAI*.

Wallach, H. 2008. *Structured Topic Models for Language*. Ph.D. Dissertation, University of Cambridge.

Wood, R. G., and Rutenbar, R. A. 1997. FPGA Routing and Routability Estimation Via Boolean Satisfiability. In *Int. Symposium on Field-Programmable Gate Arrays*.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: Portfolio-based Algorithm Selection for SAT. *JAIR*.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2009. SATzilla2009: an Automatic Algorithm Portfolio for SAT. In *SAT Competition 2009*.