Bayesian Ranking for Strategy Scheduling in Automated Theorem Provers

Chaitanya Mangla[0000−0001−7600−1681], Sean B. Holden[0000−0001−7979−1148], and Lawrence C. Paulson[0000−0003−0288−4279]

Computer Laboratory, University of Cambridge, England
{cm772,sbh11,lp15}@cl.cam.ac.uk

Abstract. A *strategy schedule* allocates time to proof strategies that are used in sequence in a theorem prover. We employ Bayesian statistics to propose alternative sequences for the strategy schedule in each proof attempt. Tested on the TPTP problem library, our method yields a time saving of more than 50%. By extending this method to optimize the fixed time allocations to each strategy, we obtain a notable increase in the number of theorems proved.

Keywords: Bayesian Machine Learning · Strategy Scheduling · Automated Theorem Proving.

1 Introduction

Theorem provers have wide-ranging applications, including formal verification of large mathematical proofs [9] and reasoning in knowledge-bases [37]. Thus, improvements in provers that lead to more successful proofs, and savings in the time taken to discover proofs, are desirable.

Automated theorem provers generate proofs by utilizing inference procedures in combination with heuristic search. A specific configuration of a prover, which may be specialized for a certain class of problems, is termed a *strategy*. Provers such as E [27] can select from a portfolio of strategies to solve the goal theorem. Furthermore, certain provers hedge their allocated proof time across a number of proof strategies by use of a *strategy schedule*, which specifies a time allocation for each strategy and the sequence in which they are used until one proves the goal theorem. This method was pioneered in the Gandalf prover [33].

Prediction of the effectiveness of a strategy prior to a proof attempt is usually intractable or undecidable [12]. A practical implementation must infer such a prediction by tractable approximations. Therefore, machine learning methods for strategy invention, selection and scheduling are actively researched. Machine learning methods for strategy selection conditioned on the proof goal have shown promising results [3]. Good results have also been reported for strategy synthesis using machine learning [1]. Work on machine learning for algorithm portfolios — which allocate resources to multiple solvers simultaneously — is also relevant to strategy scheduling because of its similar goals. For this purpose, Silverthorn and Miikkulainen propose latent class models [31] .

In this work, we present a method for generating strategy schedules using Bayesian learning with two primary goals: to reduce proving time or to prove more theorems. We have evaluated this method for both purposes using iLeanCoP, an intuitionistic first-order logic prover with a compact implementation and good performance [18]. Intuitionistic logic is a non-standard form of first-order logic, of which relatively little is known with regard to automation. It is of interest in theoretical computer science and philosophy of mathematics [7]. Among intuitionistic provers, iLeanCoP is seen as impressive and is able to prove a sufficient number of theorems in our benchmarks for significance testing. Its core is implemented in around thirty lines of Prolog; such simplicity adds clarity to interpretations of our results. Our method was benchmarked on the Thousands of Problems for Theorem Provers (TPTP) problem library [32], in which we are able to save more than 50% on proof time when aiming for the former goal. Towards the latter goal, we are able to prove notably more theorems.

Our two primary, complementary, contributions presented here are: first, a Bayesian machine learning model for strategy scheduling; and second, engineered features for use in that model. The text below is organized as follows. In Section 2, we introduce preliminary material used subsequently to construct a machine learning model for strategy scheduling, described in Sections 3–7. The data used to train and evaluate this model are described in Section 8, followed by experiments, results and conclusions in Sections 9–12.

2 Distribution of Permutations

We model a strategy schedule using a vector of strategies, and thus all schedules are *permutations* of the same.

Definition 1 (Permutation). *Let $M \in \mathbb{N}$. A permutation $\boldsymbol{\pi} \in \mathbb{N}^M$ is a vector of indices, with $\pi_i \in \{1, \ldots, M\}$ and $\forall i \neq j : \pi_i \neq \pi_j$, representing a reordering of the components of an M-dimensional vector $\boldsymbol{s}$ to $[s_{\pi_1}, s_{\pi_2}, \ldots, s_{\pi_M}]^\intercal$.*

In this text, vector-valued variables, such as $\boldsymbol{\pi}$ above, are in boldface, which must change when they are indexed, like π_1 for example. For probabilistic modelling of schedules represented using permutations, we use the Plakett-Luce model [14,21] to define a parametric probability distribution over permutations.

Definition 2 (Plakett-Luce distribution). *The Plakett-Luce distribution* $\mathrm{Perm}(\boldsymbol{\lambda})$ *with parameter $\boldsymbol{\lambda} \in \mathbb{R}^M_{>0}$, has support over permutations of indices $\{1, \ldots, M\}$. For permutation $\boldsymbol{\Pi}$ distributed as* $\mathrm{Perm}(\boldsymbol{\lambda})$*,*

$$\Pr(\boldsymbol{\Pi} = \boldsymbol{\pi}; \boldsymbol{\lambda}) = \prod_{j=1}^{M} \frac{\lambda_{\pi_j}}{\sum_{u=j}^{M} \lambda_{\pi_u}}.$$

In latter sections, we use the parameter $\boldsymbol{\lambda}$ to assign an abstract 'score' to strategies when modelling distributions over schedules. This score is particularly useful due to the following theorem.

Theorem 1. *Let* $\boldsymbol{\pi}^*$ *be a mode of the distribution* $\mathrm{Perm}(\boldsymbol{\lambda})$*, that is*

$$\boldsymbol{\pi}^* = \operatorname*{argmax}_{\boldsymbol{\pi}} \Pr(\boldsymbol{\pi}; \boldsymbol{\lambda}).$$

Then, $\lambda_{\pi_1^*} \geqslant \lambda_{\pi_2^*} \geqslant \lambda_{\pi_3^*} \geqslant \ldots \geqslant \lambda_{\pi_M^*}$.

Thus, assuming $\boldsymbol{\lambda}$ is a vector of the score of each strategy, the highest probability permutation indexes the strategies in decreasing order of scores. Conversely, the highest probability permutation can be obtained efficiently by sorting the indices of $\boldsymbol{\lambda}$ with respect to their corresponding values in decreasing order. Cao et al. [4] have presented a proof of Theorem 1, and Cheng et al. [5] have discussed some further interesting details.

Example 1. Let $\boldsymbol{\lambda} = [1, 9]^\intercal$, $\boldsymbol{\pi}^{(1)} = [1, 2]^\intercal$ and $\boldsymbol{\pi}^{(2)} = [2, 1]^\intercal$. Then,

$$\Pr(\boldsymbol{\Pi} = \boldsymbol{\pi}^{(1)}; \boldsymbol{\lambda}) = \frac{\lambda_{\boldsymbol{\pi}_1^{(1)}}}{\lambda_{\boldsymbol{\pi}_1^{(1)}} + \lambda_{\boldsymbol{\pi}_2^{(1)}}} \cdot \frac{\lambda_{\boldsymbol{\pi}_2^{(1)}}}{\lambda_{\boldsymbol{\pi}_2^{(1)}}} = \frac{1}{1+9} \cdot \frac{9}{9} = \frac{1}{10}.$$

Similarly, $\Pr(\boldsymbol{\Pi} = \boldsymbol{\pi}^{(2)}; \boldsymbol{\lambda}) = 9/10$. □

Theorem 2. $\mathrm{Perm}(c\boldsymbol{\lambda}) = \mathrm{Perm}(\boldsymbol{\lambda})$*, for any scalar constant* $c > 0$.

In other words, the Plakett-Luce distribution is invariant to the scale of the parameter vector.

Lemma 1. $\mathrm{Perm}(\exp(\boldsymbol{\lambda} + c)) = \mathrm{Perm}(\exp(\boldsymbol{\lambda}))$*, for any scalar constant* $c \in \mathbb{R}$.

Lemma 1 follows from Theorem 2, and shows the same distribution is translation invariant if the parameter is exponentiated. Cao et al. [4] give proofs of both.

3 A Maximum Likelihood Model

We model a strategy schedule as a ranking of known strategies, where each strategy is constructed by a parameter setting and time allocation. A ranking therein is a permutation of strategies, with each strategy retaining its time allocation irrespective of the ordering. We construct, in this section, a model for inference of such permutations that is linear in the parameters.

Suppose we have a repository of N theorems which we test against each of our M known strategies to build a data-set $\mathcal{D} = \{(\boldsymbol{\pi}^{(i)}, \boldsymbol{x}^{(i)})\}_{i=1}^{N}$, where $\boldsymbol{\pi}^{(i)}$ is a desirable ordering of strategies for theorem i and $\boldsymbol{x}^{(i)}$ is a feature vector representation of the theorem. In Section 9, we detail how we instantiated $\mathcal{D}$ for our experiments, which serves as an example for any other implementation. We assume that $\boldsymbol{\pi}^{(i)}$ has Plakett-Luce distribution conditioned on $\boldsymbol{x}^{(i)}$ such that

$$\Pr(\boldsymbol{\pi}; \boldsymbol{x}, \boldsymbol{\omega}) = \mathrm{Perm}(\boldsymbol{\Lambda}(\boldsymbol{x}, \boldsymbol{\omega})), \tag{1}$$

where $\boldsymbol{\omega}$ is a parameter the model must learn and $\boldsymbol{\Lambda}(\cdot)$ is a vector-valued function of range $\mathbb{R}_{>0}^{M}$. We use the notation $\Lambda(\cdot)_i$ to index into the value of $\boldsymbol{\Lambda}(\cdot)$. We

represent our prover strategies with feature vectors $\{\boldsymbol{d}^{(j)}\}_{j=1}^{M}$. To calculate the score of strategy j using $\Lambda(\cdot)_j$, we specify

$$\Lambda(\boldsymbol{x}^{(i)}, \boldsymbol{\omega})_j = \exp\left(\boldsymbol{\phi}(\boldsymbol{x}^{(i)}, \boldsymbol{d}^{(j)})^\intercal \boldsymbol{\omega}\right) \tag{2}$$

to ensure that the scores are positive valued, where $\boldsymbol{\phi}$ is a suitable basis expansion function. Assuming the data is i.i.d, the likelihood of the parameter vector is given by

$$\mathcal{L}(\boldsymbol{\omega}) = p(\mathcal{D}; \boldsymbol{\omega}) = \prod_{i=1}^{N} \Pr(\boldsymbol{\pi}^{(i)}; \boldsymbol{\Lambda}(\boldsymbol{x}^{(i)}, \boldsymbol{\omega})). \tag{3}$$

An $\hat{\boldsymbol{\omega}}$ that maximizes this likelihood can then be used to forecast the distribution over permutations for a new theorem $\boldsymbol{x}^*$ by evaluating $\mathrm{Perm}(\boldsymbol{\Lambda}(\boldsymbol{x}^*, \hat{\boldsymbol{\omega}}))$ for all permutations. This would incur factorial complexity; however, we are often only interested in the most likely permutation, which can be retrieved in polynomial time. Specifically for strategy scheduling the permutation with the highest predicted probability should reflect the orderings in the data. For this purpose, we use Theorem 1 to find the highest probability permutation $\boldsymbol{\pi}^*$ by sorting the values of $\{\boldsymbol{\Lambda}(\boldsymbol{x}^*, \hat{\boldsymbol{\omega}})_j\}_{j=1}^{M}$ in descending order.

Remark 1. A method named ListNet designed to rank documents for search queries using the Plakett-Luce distribution is evaluated by Cao et al [4]. Their evaluation uses a linear basis expansion. We can derive a similar construction in our model by setting

$$\boldsymbol{\phi}(\boldsymbol{x}^{(i)}, \boldsymbol{d}^{(j)}) = [\boldsymbol{x}^{(i)\intercal}, \boldsymbol{d}^{(j)\intercal}]^\intercal. \tag{4}$$

Remark 2. The likelihood in Equation (3) can be maximized by minimizing the negative log likelihood $\ell(\boldsymbol{\omega}) = -\log \mathcal{L}(\boldsymbol{\omega})$, which (as shown by Schäfer and Hüllermeier [26]) is convex and therefore can be minimized using gradient-based methods. The minima may, however, be unidentifiable due to translation invariance, as demonstrated by Lemma 1. This problem is eliminated in our Bayesian model by the use of a Gaussian prior, as explained in Section 4.

Example 2. Let there be $N = 2$ theorems and $M = 2$ strategies. Let the theorems and strategies be characterized by univariate values such that $x^{(1)} = 1$, $x^{(2)} = 2$, $d^{(1)} = 1$ and $d^{(2)} = 2$.

Suppose strategy $d^{(1)}$ is ideal for theorem $x^{(1)}$ and strategy $d^{(2)}$ for $x^{(2)}$, as shown on the right, where a + indicates the preferred strategy.

	$d^{(1)}$	$d^{(2)}$
$x^{(1)}$	+	−
$x^{(2)}$	−	+

This is evidently an example of a parity problem [34], and hence cannot be modelled by a simple linear expansion using the basis function mentioned in Remark 1. A solution in this instance is to use

$$\phi(x^{(i)}, d^{(j)}) = x^{(i)} \cdot d^{(j)}.$$

The parameter ω is then one-dimensional, and the required training data takes the form $\mathcal{D} = \{([1, 2]^\intercal, 1), ([2, 1]^\intercal, 2)\}$. We find that $\mathcal{L}(w)$ is convex, with maxima at $\hat{\omega} = 0.42$ as shown in figure 1. □

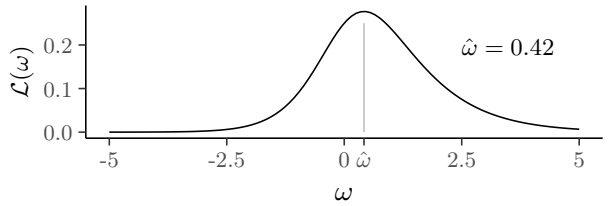


Fig. 1: The likelihood function in Example 2.

4 Bayesian Inference

We place a Gaussian prior distribution on the parameter $\boldsymbol{\omega}$ of the model described in Section 3. This has two advantages: first, the posterior mode is identifiable, as noted by Johnson et al. [11] and demonstrated in Example 3 on page 7; second, the parameter is regularized. With this prior specified as the normal distribution

$$\boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{m_0}, \boldsymbol{S_0}), \tag{5}$$

and assuming $\boldsymbol{\pi}$ is independent of $\mathcal{D}$ given $(\boldsymbol{x}, \boldsymbol{\omega})$, the posterior predictive distribution is

$$p(\boldsymbol{\pi}|\boldsymbol{x}^*, \mathcal{D}) = \int p(\boldsymbol{\pi}|\boldsymbol{x}^*, \boldsymbol{\omega}) p(\boldsymbol{\omega}|\mathcal{D}) \mathrm{d}\boldsymbol{\omega},$$

which may be approximated by sampling from the posterior,

$$\boldsymbol{\omega}^s \sim p(\boldsymbol{\omega}|\mathcal{D}), \tag{6}$$

to obtain

$$p(\boldsymbol{\pi}|\boldsymbol{x}^*, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^{S} p(\boldsymbol{\pi}|\boldsymbol{x}^*, \boldsymbol{\omega}^s). \tag{7}$$

Given a new theorem $\boldsymbol{x}^*$, to find the permutation of strategies with the highest probability of success, using the approximation above would require its evaluation for every permutation of $\boldsymbol{\pi}$. This process incurs factorial complexity. We instead make a Bayes point approximation [16] using the mean values of the samples such that,

$$\begin{aligned} p(\boldsymbol{\pi}|\boldsymbol{x}^*, \mathcal{D}) &\approx p(\boldsymbol{\pi}|\boldsymbol{x}^*, \langle \boldsymbol{\omega}^s \rangle) && \text{using eq. (7)} \\ &= \Pr(\boldsymbol{\pi}|\boldsymbol{\Lambda}(\boldsymbol{x}^*, \langle \boldsymbol{\omega}^s \rangle)) && \text{using eq. (1)}, \end{aligned}$$

where $\langle \cdot \rangle$ denotes mean value. The mean of the Plakett-Luce parameter for Bayesian inference has been used in prior work [8] to obtain good results. Furthermore, using that, the highest probability permutation can be obtained by using Theorem 1, thereby incurring only the cost of sorting the items. This saving is substantial when generating a strategy schedule, because it saves on prediction time, which is important for the following reason.

Algorithm 1 Metropolis-Hastings Algorithm

Suppose we have generated samples $\{\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(i)}\}$ from the *target distribution* p. Generate $\boldsymbol{\omega}^{(i+1)}$ as follows.

1: Generate candidate value $\dot{\boldsymbol{\omega}} \sim q(\boldsymbol{\omega}^{(i)})$, where q is the *proposal distribution*.
2: Evaluate $r \equiv r(\boldsymbol{\omega}^{(i)}, \dot{\boldsymbol{\omega}})$ where

$$r(x, y) = \min\left\{\frac{p(y)}{p(x)}\frac{q(x|y)}{q(y|x)}, 1\right\}.$$

3: Set

$$\boldsymbol{\omega}^{(i+1)} = \begin{cases} \dot{\boldsymbol{\omega}} & \text{with probability } r \\ \boldsymbol{\omega}^{(i)} & \text{with probability } 1 - r. \end{cases}$$

Remark 3. While benchmarking and in typical use, a prover is allocated a fixed amount of time for a proof attempt, and any time taken to predict a strategy schedule must be accounted for within this allocation. Time taken for this prediction is time taken away from the prover itself which could have been invested in the proof search. Therefore, it is essential to minimize schedule prediction time. It is particularly wise to favour a saving in prediction time at the cost of model optimization and training time.

Remark 4. In our implementation we set $\boldsymbol{m}_0 = 0$. This has the effect of prioritizing smaller weights $\boldsymbol{\omega}$ in the posterior. Furthermore, we set $\boldsymbol{S}_0 = \eta\boldsymbol{I}, \eta \in \mathbb{R}$, where $\boldsymbol{I}$ is the identity matrix. Consequently, the hyperparameter η controls the strength of the prior, since the entropy of the Gaussian prior scales linearly by $\log|\boldsymbol{S}_0|$.

Remark 5. A specialization of the Plakett-Luce distribution using the Thurstonian interpretation admits a Gamma distribution conjugate prior [8]. That, however, is unavailable to our model when parametrized as shown in Equation (1).

5 Sampling

We use the Markov chain Monte Carlo (MCMC) Metropolis-Hastings algorithm [38] to generate samples from the posterior distribution. In MCMC sampling, one constructs a Markov chain whose stationary distribution matches the target distribution p. For the Metropolis-Hastings algorithm, stated in Algorithm 1, this chain is constructed using a proposal distribution $y|x \sim q$, where q is set to a distribution that can be conveniently sampled from.

Note that while calculating r in Algorithm 1, the normalization constant of the target density p cancels out. This is to our advantage; to generate samples $\boldsymbol{\omega}^s$ from the posterior, which is, by Equation (3) and Equation (5),

$$\begin{aligned} p(\boldsymbol{\omega}|\mathcal{D}) &\propto p(\mathcal{D}|\boldsymbol{\omega})p(\boldsymbol{\omega}) \\ &= \mathcal{L}(\boldsymbol{\omega})\mathcal{N}(\boldsymbol{m_0}, \boldsymbol{S_0}), \end{aligned} \tag{8}$$

the posterior only needs to be computed in this unnormalized form.

In this work, we choose a random walk proposal of the form

$$q(\boldsymbol{\omega}'|\boldsymbol{\omega}) = \mathcal{N}(\boldsymbol{\omega}'|\boldsymbol{\omega}, \Sigma_q), \tag{9}$$

and tune Σ_q for efficient sampling simulation. We start the simulation at a local mode $\hat{\boldsymbol{\omega}}$, and set $\mathcal{N}(\hat{\boldsymbol{\omega}}, \Sigma_q)$ to approximate the local curvature of the posterior at that point using methods by Rossi [25]. Specifically, our procedure for computing Σ_q is as follows.

1. First, writing the posterior from Equation (8) as
$$p(\boldsymbol{\omega}|\mathcal{D}) = \frac{1}{Z} e^{-E(\boldsymbol{\omega})},$$
where Z is the normalization constant, we have
$$E(\boldsymbol{\omega}) = -\log \mathcal{L}(\boldsymbol{\omega}) - \log \mathcal{N}(\boldsymbol{m_0}, \boldsymbol{S_0}). \tag{10}$$
We find a local mode $\hat{\boldsymbol{\omega}}$ by optimizing $E(\boldsymbol{\omega})$ using a gradient-based method.
2. Then, using a Laplace approximation [2], we approximate the posterior in the locality of this mode to
$$\mathcal{N}(\hat{\boldsymbol{\omega}}, H^{-1}), \text{ where } H = \nabla\nabla E(\boldsymbol{\omega})|_{\hat{\boldsymbol{\omega}}}$$
is the Hessian matrix of $E(\boldsymbol{\omega})$ evaluated at that local mode.
3. Finally, we set
$$\Sigma_q = s^2 H^{-1}$$
in Equation (9), where s is used to tune all the length scales. We set this value to $s^2 = 2.38$ based on the results by Roberts and Rosenthal [24].

Remark 6. When calculating r in Algorithm 1 during sampling, to evaluate the unnormalized posterior at any point $\boldsymbol{\omega}^s$ we compute it from Equation (10) as $\exp(-E(\boldsymbol{\omega}^s))$ — it is therefore the only form in which the posterior needs to be coded in the implementation.

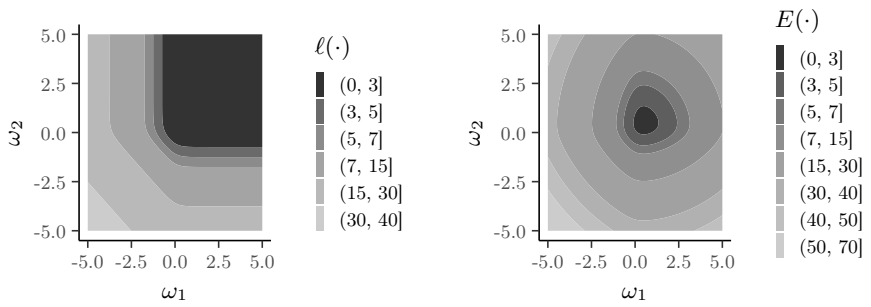
Example 3 (Gaussian Prior). To demonstrate the effect of using a Gaussian prior, we build upon Example 2, with the data taking the form

$$\mathcal{D} = \{([1,2]^\intercal, 1), ([2,1]^\intercal, 2)\}.$$

We perform basis expansion as explained in Section 6 with prior parameter $\eta = 1.0$, kernel $\sigma = 0.1$ and $\varsigma = 2$ centres. Thus, the model parameter is

$$\boldsymbol{\omega} = [\omega_1, \omega_2]^\intercal, \quad \boldsymbol{\omega} \in \mathbb{R}^2.$$

The unnormalized negative log posterior $E(\omega_1, \omega_2)$, as defined in Equation (10), is shown in Figure 2b; and the negative log likelihood $\ell(\omega_1, \omega_2) = -\log \mathcal{L}(\omega_1, \omega_2)$ as mentioned in Remark 2, is shown in Figure 2a. Note the contrast in the shape of the two surfaces. The minimum is along the top-right portion in Figure 2a, which is flat and leads to an unidentifiable point estimate, whereas in Figure 2b, the minimum is in a narrow region near the centre. The Gaussian prior, in informal terms, has lifted the surface up, with an effect that increases in proportion to the distance from the origin. □



(a) Likelihood function $\ell(\omega_1, \omega_2)$ (b) Posterior function $E(\omega_1, \omega_2)$

Fig. 2: Comparison of the shape of the likelihood and the posterior functions.

6 Basis Expansion

Example 2 shows how the linear expansion in Remark 1 is ineffective even in very simple problem instances. The maximum likelihood bilinear model presented by Schäfer and Hüllermeier [26] is related to our model defined in Section 2 with the basis performing the Kronecker (tensor) product $\boldsymbol{\phi}(x, d) = x \otimes d$. Their results show such an expansion produces a competitive model, but falls behind in comparison to their non-linear model.

To model non-linear interactions between theorems and strategies, we use a *Gaussian kernel* for the basis expansion.

Definition 3 (Gaussian kernel). *A Gaussian kernel κ is defined by*

$$\kappa(\boldsymbol{y}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{y} - \boldsymbol{z}\|^2}{2\sigma^2}\right), \quad \textit{for } \sigma > 0.$$

The Gaussian kernel $\kappa(\boldsymbol{y}, \boldsymbol{z})$ effectively represents the inner product of $\boldsymbol{y}$ and $\boldsymbol{z}$ in a Hilbert space whose bandwidth is controlled by σ. Smaller values of σ correspond to a higher bandwidth, more flexible, inner product space. Larger values of σ will reduce the kernel to a constant function, as detailed in [30]. For our ranking model, we must tune σ to balance between over-fitting and under-performance.

We use the Gaussian kernel for basis expansion by setting

$$\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{d}) = \left[\kappa\big([\boldsymbol{x}^\intercal, \boldsymbol{d}^\intercal]^\intercal, \boldsymbol{c}^{(1)}\big), \ldots, \kappa\big([\boldsymbol{x}^\intercal, \boldsymbol{d}^\intercal]^\intercal, \boldsymbol{c}^{(C)}\big)\right]^\intercal,$$

where $\{\boldsymbol{c}^{(i)}\}_{i=1}^{C}$ is a collection of *centres*. By choosing centres to be themselves composed of theorems $\boldsymbol{x}^{(\cdot)}$ and strategies $\boldsymbol{d}^{(\cdot)}$, such that $\boldsymbol{c}^{(\cdot)} = [\boldsymbol{x}^{(\cdot)\intercal}, \boldsymbol{d}^{(\cdot)\intercal}]^\intercal$, the basis expansion above represents each data item with a non-linear inner product against other known items.

To find the relevant subset of $\mathcal{D}$ from which centres should be formed, we follow the method described in the steps below.

1. Initially, we set the collection of centres to every possible centre. That is, for N theorems and M strategies, we produce a centre for every combination of the two, thereby producing $C = N \cdot M$ centres.
2. Next, we use $\boldsymbol{\phi}$ to expand every centre to produce the $C \times C$ matrix $\boldsymbol{\Gamma}$ such that
$$\boldsymbol{\Gamma}_{i,j} = \boldsymbol{\phi}(\boldsymbol{c}^{(i)})_j = \kappa(\boldsymbol{c}^{(i)}, \boldsymbol{c}^{(j)}).$$
3. Then, we generate a vector $\boldsymbol{\gamma}$ such that $\boldsymbol{\gamma}_i$ represents a *score* for centre $\boldsymbol{c}^{(i)}$. Since each centre is a combination of a theorem and a strategy, we set the score to signify how well the strategy performs for that theorem, as detailed in Remark 7 below.
4. Finally, we use Automatic Relevance Determination (ARD) [17] with $\boldsymbol{\Gamma}$ as input and $\boldsymbol{\gamma}$ as the response variable. The result is a weight assignment to each centre to signify its relevance. The highest absolute-weighted ς centres are chosen, where ς is a parameter which decides the total number of centres.

This method is inspired by the procedure used in Relevance Vector Machines [35] for a similar purpose.

Remark 7 (score). For a strategy that succeeds in proving a theorem, the score for the pair is the fraction of the time allocation left unconsumed by the prover. For an unsuccessful strategy-theorem combination, we set the score to a value close to zero.

Remark 8 (ς). The parameter ς is another tunable parameter which, in similar fashion to the parameter σ earlier in this section, controls the model complexity introduced by the basis expansion. Both variables must be tuned together.

7 Model Selection and Time Allocations

From Remark 8, ς and σ are hyperparameters that control the complexity introduced into our model through the Gaussian basis expansion; and Remark 4 introduces η, the hyperparameter that controls the strength of the prior. The final model is selected by tuning them. Tuning must aim to avoid overfitting to the training data; and to maximize, during testing, either the savings in proof-search time or the number of theorems proved. However, we do not have a closed-form expression relating these parameters to this aim, thus any combination of the parameters can be judged only by testing them.

In this work we have used *Bayesian optimization* [29] to optimize these hyperparameters. Bayesian optimization is a black-box parameter optimization method that attempts to search for a global optimum within the scope of a set resource budget. It models the optimization target as a user-specified *objective function*, which maps from the parameter space to a loss metric. This model of the objective function is constructed using *Gaussian Process* (GP) regression [22], using data generated by repeatedly testing the objective function.

Our specified objective function maps from the hyperparameters $(\varsigma, \sigma, \eta)$ to a loss metric ξ. We use cross-validation within the training data while calculating ξ

to penalize hyperparameters that over-fit. Hyperparameters are tuned at training time only, after which they are fixed for subsequent testing. The final test set is never used for any hyperparameter optimization.

In the method presented thus far we are only permuting strategies with fixed time allocations to build a sequence for a strategy schedule. In this setting, the number of theorems proved cannot change, but the time taken to prove theorems can be reduced. Therefore, with this aim, a useful metric for ξ is the total time taken by the theorem prover to prove the theorems in the cross-validation test set.

However, we can take further advantage of the hyperparameter tuning phase to additionally tune the times allocated to each strategy, by treating these times as hyperparameters. Therefore, for each strategy $\boldsymbol{d}^{(i)}$ we create a hyperparameter $\nu^{(i)} \in (0, 1)$ which sets the proportion of the proof time allocated to that strategy. We can then optimize our model to maximize the number of theorems proved; a count of the remaining theorems is then a viable metric for ξ. Note that once the $\nu^{(\cdot)}$ are set, time allocation for $\boldsymbol{d}^{(i)}$ is fixed to $\nu^{(i)}$, irrespective of its order in the strategy schedule.

Remark 9. Our results include two types of experiment:

- one where the time allocations for each strategy are set to the defaults shipped with our reference theorem prover, and so we optimize for saving proof time; and
- another wherein we allocate time to each strategy during the hyperparameter tuning phase, and so we optimize for proving the maximum number of theorems.

8 Training Data and Feature Extraction

Our chosen theorem prover, iLeanCoP, is shipped with a fixed strategy schedule consisting of 5 strategies. It splits the allocated proof time across the first four strategies by 2%, 60%, 20% and 10%. However, only the first strategy is complete and therefore usually expected to take up its entire time allocation. The remaining strategies are incomplete, and may exit early on failure. Therefore, the fifth and final strategy, which we refer to as the fallback strategy, is allocated all the remaining time.

Emulating iLeanCop. We have constructed a dataset by attempting to prove every theorem in our problem library using each of these strategies individually. With this information, the result of any proof attempt can be calculated by emulating the behaviour of iLeanCoP. This is how we evaluate the predicted schedules — we emulate a proof attempt by iLeanCoP using that schedule for each theorem in the test set. For a faithful emulation of the fallback strategy, it is always attempted last, and therefore any new schedule is only a permutation of the first four strategies. Our experiments allocate a time of 600 seconds per

theorem. The dataset is built to ensure that, within this proof time, any such strategy permutation can be emulated. We kept a timeout of 1200s per strategy per theorem when building the dataset, which is more than sufficient for current experiments and gives us headroom for future experiments with longer proof times.

Strategy Features. Each strategy in iLeanCoP consists of a time allocation and parameter settings; the parameters are described by Otten [19]. We use a one-hot encoding feature representation for strategies based on the parameter setting as shown in Table 1. Another feature noting the completeness of each strategy is also shown. Another feature (not shown in the table) contains the time allocated to each strategy. Note the fallback strategy is used in prover emulation but not in the schedule prediction.

Table 1: Features of the four main strategies.

Strategy	Parameter					Completeness
	`def`	`scut`	`cut`	`comp(7)`	`conj`	
`def,scut,cut,comp(7)`	1	1	1	1	0	1
`def,scut,cut`	1	1	1	0	0	0
`conj,scut,cut`	0	1	1	0	1	0
`def,conj,cut`	1	0	1	0	1	0

Theorem Features. The TPTP problem library contains a large, comprehensive collection of theorems and is designed for testing automated theorem provers. The problems are taken from a range of domains such as Logic Calculi, Algebra, Software Verification, Biology and Philosophy, and presented in multiple logical forms. For iLeanCoP, we select the subset in first-order form, denoted there as `FOF`. In version 7.1.0, there are 8157 such problems covering 43 domains. Each problem consists of a set of formulae and a goal theorem. The problems are of varying sizes. For example, the problem named `HWV134+1` from the Hardware Verification domain contains 128975 formulae, whilst `SET703+4` from the Set Theory domain contains only 12.

We have constructed a dataset containing features extracted from the first-order logic problems in TPTP (see Appendix A). Here, we describe how those features were developed.

In deployment, a prover using our method to generate strategy schedules would have to extract features from the goal theorem at the beginning of a proof attempt. To minimize the computational overhead of feature extraction, in keeping with our goal noted in Remark 3, we use features that can be collected when the theorem is parsed by the prover. The collection of features developed

in this work is based on the authors' prior experience, and later we will briefly examine the quality of each feature to discard the uninformative ones. We extract the following features, which are all considered candidates for the subsequent feature selection process.

Symbol Counts: A count of the logical connectives and quantifiers. We extract one feature per symbol by tracking lexical symbols encountered while parsing.
Quantifier Rank: The maximum depth of nesting of quantifiers.
Quantifier Count: A count of the number of quantifiers.
Mean and Maximum Function Arity: Obtained by keeping track of functions during parsing.
Number of Functions: A count of the number of functions.
Quantifier Alternations: A count of the number of times the quantifiers flip between the existential and universal. When calculated by examining only the sequence of lexical symbols, the count may be inaccurate. An accurate count is obtained by tracking negations during parsing while collecting quantifiers. We extract both as candidates.

Feature Selection and Pre-processing. We examine the degree of association between the individual theorem features described above and the speed with which the strategies solve each theorem; for this we use the Maximal Information Coefficient (MIC) measure [23]. For every theorem we calculate the score, as defined in Remark 7, averaged over all strategies. This score is paired with each feature to calculate its MIC. Most lexical symbols achieve an MIC close to zero. We selected the features with relatively high MIC for the presented work, and these are shown in Figure 3.

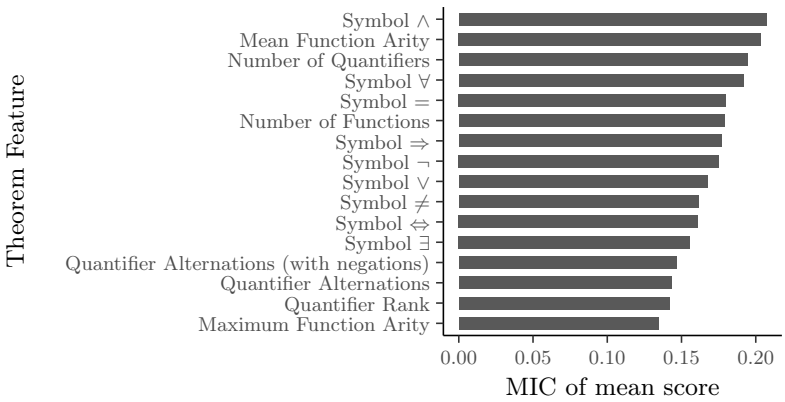


Fig. 3: MIC between selected features and scores.

The two features based on quantifier alternations are clearly correlated, but both meet the above criterion for selection. Correlations can also be expected

between the other features. Furthermore, our features range over different scales. For example, the maximal function arity in TPTP averages 2, whereas the number of predicate symbols averages 2097. It is desirable to remove these correlations to alleviate any burden on the subsequent modelling phase, and to standardize the features to zero mean and unit variance to create a feature space with similar length-scales in all dimensions. The former is achieved by *decorrelation*, the latter by *standardization*, and both together by a *sphering transformation*. We transform our extracted features as such using Zero-phase Component Analysis (ZCA), which ensures the transformed data is as close as possible to the original [6].

Coverage. As mentioned above, we run iLeanCoP on every first-order theorem in TPTP with each strategy allocated 1200 seconds. Although every theorem in intuitionistic logic also holds for classical logic, the converse does not hold. For that reason and because of the limitations of iLeanCoP, many theorems remain unproved by any strategy. We exclude these theorems from our experiments, leaving us with a data-set of 2240 theorems.

9 Experiments

We present two experiments in this work, as noted in Remark 9. In this section, we describe our experimental apparatus in detail.

As noted in Section 8, our data contains:

- $N = 2240$ theorems that are usable in our experiments;
- five strategies, of which $M = 4$ are used to build strategy schedules since one is a fallback strategy; and
- features $\boldsymbol{x}^{(i)}$ of theorems where $i \in [1, N]$ and features $\boldsymbol{d}^{(j)}$ of strategies where $j \in [1, M]$.

This data needs to be presented to our model for training in the form of $\mathcal{D} = \{(\boldsymbol{\pi}^{(i)}, \boldsymbol{x}^{(i)})\}_{i=1}^{N}$, as described in Section 3. Since the two experiments have slightly different goals, we specialize $\mathcal{D}$ according to each.

When aiming to predict schedules that minimize the time taken to prove theorems, a natural value for $\boldsymbol{\pi}^{(i)}$ is the index order that sorts strategies in increasing amounts of time taken to prove theorem i. However, some strategies may fail to prove theorem i within their time allocation. In that case, we consider the failed strategies equally bad and place them last in the ordering in $\boldsymbol{\pi}^{(i)}$. Furthermore, we create additional items $(\boldsymbol{\pi}'^{(i)}, \boldsymbol{x}^{(i)})$ in $\mathcal{D}$, by permuting the positions of the failed strategies in $\boldsymbol{\pi}^{(i)}$ to create multiple $\boldsymbol{\pi}'^{(i)}$.

When the goal is only to prove more theorems, the strategies that succeed are all considered equally ranked above the failed strategies. In this mode, the successful strategies are similarly permuted in the data, in addition to those that failed.

In each experiment, a random one-third of the N theorems are separated into a *holdout* test set $\dot{\boldsymbol{N}}$, leaving behind a training set $\ddot{\boldsymbol{N}}$. This training set

is first used for hyperparameter tuning using BO. As explained in Section 7, each hyperparameter combination is tested with five-fold cross-validation within $\ddot{\boldsymbol{N}}$, to penalize instances that overfit to $\ddot{\boldsymbol{N}}$. This results in estimated optimum values for the hyperparameters. These are used to set the model, which is then trained on $\ddot{\boldsymbol{N}}$ and then finally evaluated on $\dot{\boldsymbol{N}}$. The whole process is repeated ten times with new random splits $\dot{\boldsymbol{N}}$ and $\ddot{\boldsymbol{N}}$ to create one set of ten results for that experiment.

10 Results

Each experiment, repeated ten times, is conducted in two phases: first, hyperparameter optimization; and second, model training and evaluation. The bounds on the search space in the first phase were always the same (see Appendix A). The holdout test set contained 747 theorems. A proof time of 600s was emulated.

10.1 Experiment 1: Optimizing Proof Attempt Time

The results are shown in Figure 4. The total prediction time for all 747 theorems, averaged across the trials, is 0.14s.

The times across proof attempts are not normally distributed, for both the unmodified iLeanCoP schedule and the predicted ones, as confirmed by a Jarque-Bera test. Therefore, we used the right-tailed Wilcoxon signed-rank test for a pair-wise comparison of the times taken for each theorem by the original schedule in iLeanCoP versus the predicted schedules, resulting in a p-value of less than 10^{-6} in each trial, confirming the alternate hypothesis that the reduction in time taken to prove each theorem comes from a distribution with median greater than zero. This confirms that the time savings are statistically significant. Furthermore, we note from Figure 4 a saving of more than 50% in the total proof-time in each trial.

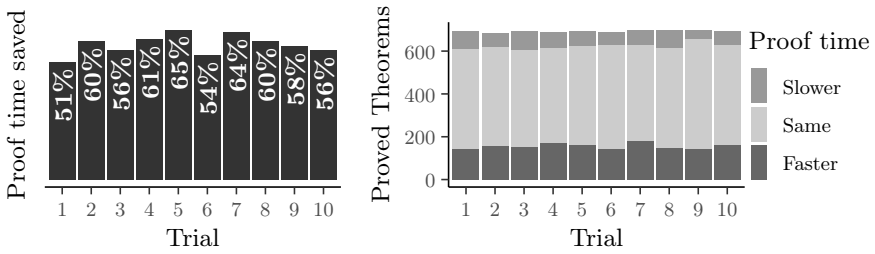


Fig. 4: Results of Experiment 1. Proof times are compared with precision 10^{-6}s.

10.2 Experiment 2: Proving More Theorems

We set our hyperparameter search to find time allocations for strategies. The resulting predicted schedules have gains and losses when compared to the original schedule, as shown in the four facets of Figure 5. However, there is a consistent gain in the number of theorems proved and a gain of five theorems on average, evident from the mean values in (†) and (‡).

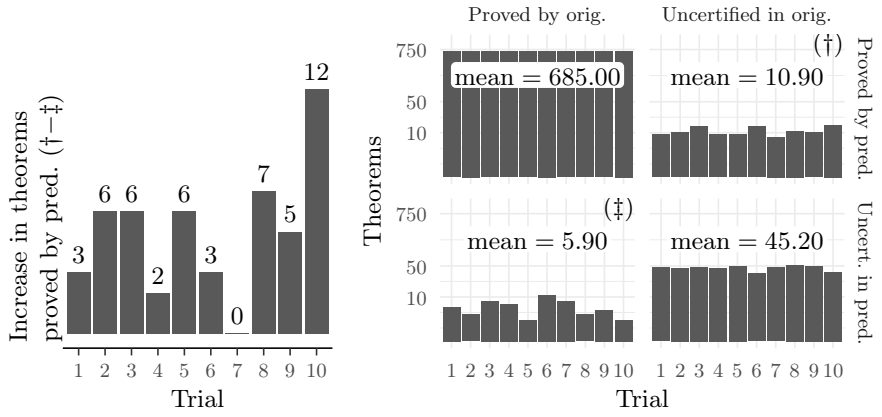


Fig. 5: Comparison of the proof attempts by the original (orig.) and predicted (pred.) schedules in Experiment 2. Theorems which are proved by pred. but could not be proved by orig. are counted in †, and the vice versa in ‡.

11 Related Work

Prior work on machine learning for *algorithm selection*, such as that introduced by Leyton-Brown et al. [13], is a precursor to our work. In that topic, the machine learning methods must perform the task of selecting a good algorithm from within a portfolio to solve the given problem instance. Typically, as was the case in the work by Leyton-Brown et al. [13], the learning methods predict the runtime of all algorithms, and then pick the fastest predicted one. This line of enquiry has been extended to select algorithms for SMT solvers — a recent example is MachSMT by Scott et al. [28]. The machine learning models in MachSMT are trained by considering all the portfolio members in pairs for each problem in the training set. This method is called *pairwise ranking*, which contrasts from our method, called *list-wise ranking*, in which we consider the full list of portfolio members all together.

In terms of the machine learning task, the work on scheduling solvers bears greater similarity to our presented work. In MedleySolver, for example, Pimpalkhare et al. [20] frame this task as a multi-armed bandit problem. They predict

a sequence of solvers as well as the time allocation for each to generate schedules for the goal problems. MedleySolver is able to solve more problems than any individual solver would on its own.

With an approach that contrasts with ours, Hůla et al. [10] have made use of Graph Neural Networks (GNNs) for solver scheduling. They produce a regression model to predict, for the given problem, the runtime of all the solvers; which is used as the key to sort the solvers in increasing order of predicted runtime to build a schedule. This is an example of *point-wise ranking*. The authors use GNNs to automatically discover features for machine learning. They combine this feature extraction with training of the regression model. They achieve an increase in the number of problems solved as well as a reduction in the total proof time. Meanwhile, our use of manual feature engineering combined with statistical methods for selection and normalization has certain advantages. For one, we can analyse our features and derive a subjective interpretation of their efficacy. Additionally, our features effectively impart our domain knowledge onto the model. Such domain knowledge may not be available in the data itself. Manual feature engineering such as ours can be combined with automatic feature extraction to reap the benefits of both.

12 Conclusions

We have presented a method to specialize, for the given goal theorem, the sequence of strategies in the schedule used in each proof attempt. A Bayesian machine learning model is trained in this method using data generated by testing the prover of interest. When evaluated with the iLeanCoP prover using the TPTP library as a benchmark, our results show a significant reduction in the time taken to prove theorems. For theorems that are successfully proved, the average time saving is above 50%. The prediction time is on average low enough to have a negligible impact on the resources subtracted from the proof search itself.

We also extend this method to optimize time allocations to each strategy. In this setting, our results show a notable increase in the number of theorems proved.

This work shows, by example, that Bayesian machine learning models designed specifically to augment heuristics in theorem provers, with detailed consideration of the computational compromises required in this setting, can deliver substantial improvements.

A Implementation, Code and Data

This work is implemented primarily in Matlab [36]. All experiments can be reproduced using the code, data and instructions available at [15]. The hyper-parameter search space in all experiments was restricted to $\varsigma \in [10, 300]$, $\sigma \in [0.01, 100.0]$ and $\eta \in [1, 100]$.

Acknowledgments

Initial investigations for this work were co-supervised by Prof. Mateja Jamnik, Computer Laboratory, University of Cambridge, UK.

This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) through a Doctoral Training studentship, award reference 1788755. For the purpose of open access, the author has applied a Creative Commons Attribution (CC-BY-4.0) licence to any Author Accepted Manuscript version arising.

Computations for this work was performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) operated by the University of Cambridge Research Computing Service, provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/P020259/1), and DiRAC funding from the Science and Technology Facilities Council.

References

1. Balunovic, M., Bielik, P., Vechev, M.T.: Learning to Solve SMT Formulas. In: Annual Conference on Neural Information Processing Systems. pp. 10338–10349 (2018)
2. Barber, D.: Bayesian Reasoning and Machine Learning. Cambridge University Press (2012)
3. Bridge, J.P., Holden, S.B., Paulson, L.C.: Machine Learning for First-Order Theorem Proving. Journal of Automated Reasoning **53**(2), 141–172 (2014)
4. Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F., Li, H.: Learning to Rank: From Pairwise Approach to Listwise Approach. In: International Conference on Machine Learning. pp. 129–136. Association for Computing Machinery (2007)
5. Cheng, W., Dembczynski, K., Hüllermeier, E.: Label Ranking Methods based on the Plackett-Luce Model. In: International Conference on Machine Learning. pp. 215–222. Omnipress (2010)
6. Duboue, P.: The Art of Feature Engineering: Essentials for Machine Learning. Cambridge University Press (2020)
7. Dummett, M.: Elements of Intuitionism. Oxford: Clarendon, Second edn. (2000)
8. Guiver, J., Snelson, E.: Bayesian Inference for Plackett-Luce Ranking Models. In: International Conference on Machine Learning. pp. 377–384. Association for Computing Machinery (2009)
9. Hales, T., Adams, M., Bauer, G., Dang, T.D., Harrison, J., Le Truong, H., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T.T., et al.: A Formal Proof Of The Kepler Conjecture. Forum of Mathematics, Pi **5** (2017)
10. Hůla, J., Mojžíšek, D., Janota, M.: Graph Neural Networks for Scheduling of SMT Solvers. In: International Conference on Tools with Artificial Intelligence. pp. 447–451 (2021)
11. Johnson, S.R., Henderson, D.A., Boys, R.J.: On Bayesian inference for the Extended Plackett-Luce model (2020), arXiv:2002.05953
12. Kaliszyk, C., Urban, J., Vyskočil, J.: Machine Learner for Automated Reasoning 0.4 and 0.5 (2014), arXiv:1402.2359

13. Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., Shoham, Y.: A Portfolio Approach to Algorithm Selection. In: International Joint Conference on Artificial Intelligence. pp. 1542–1543. Morgan Kaufmann Publishers Inc. (2003)
14. Luce, R.D.: Individual Choice Behavior: A Theoretical Analysis. Wiley (1959)
15. Mangla, C.: BRASS (Feb 2022), doi: 10.5281/zenodo.6028568
16. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. MIT press (2012)
17. Neal, R.M.: Bayesian Learning for Neural Networks. Springer New York (1996)
18. Otten, J.: Clausal Connection-Based Theorem Proving in Intuitionistic First-Order Logic. In: Automated Reasoning with Analytic Tableaux and Related Methods. pp. 245–261. Springer Berlin Heidelberg (2005)
19. Otten, J.: leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic (System Descriptions). In: Automated Reasoning. pp. 283–291. Springer Berlin Heidelberg (2008)
20. Pimpalkhare, N., Mora, F., Polgreen, E., Seshia, S.A.: MedleySolver: Online SMT Algorithm Selection. In: Theory and Applications of Satisfiability Testing. pp. 453–470. Springer International Publishing (2021)
21. Plackett, R.L.: The Analysis of Permutations. Journal of the Royal Statistical Society: Series C (Applied Statistics) **24**(2), 193–202 (1975)
22. Rasmussen, C.E., Williams, C.K.I.: Gaussian processes for machine learning. Adaptive computation and machine learning, MIT Press (2006)
23. Reshef, D.N., Reshef, Y.A., Finucane, H.K., Grossman, S.R., McVean, G., Turnbaugh, P.J., Lander, E.S., Mitzenmacher, M., Sabeti, P.C.: Detecting novel associations in large data sets. Science **334**(6062), 1518–1524 (2011)
24. Roberts, G.O., Rosenthal, J.S.: Optimal Scaling for Various Metropolis-Hastings Algorithms. Statistical Science **16**(4), 351–367 (2001)
25. Rossi, P.E.: Bayesian statistics and marketing. J. Wiley (2006)
26. Schäfer, D., Hüllermeier, E.: Dyad ranking using Plackett–Luce models based on joint feature representations. Machine Learning **107**(5), 903–941 (2018)
27. Schulz, S.: E — a Brainiac Theorem Prover. AI Communications **15**(2,3), 111–126 (2002)
28. Scott, J., Niemetz, A., Preiner, M., Nejati, S., Ganesh, V.: MachSMT: A Machine Learning-based Algorithm Selector for SMT Solvers. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 303–325. Springer International Publishing (2021)
29. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., De Freitas, N.: Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE **104**(1), 148–175 (2015)
30. Shawe-Taylor, J.: Kernel methods for pattern analysis. Cambridge University Press (2004)
31. Silverthorn, B., Miikkulainen, R.: Latent Class Models for Algorithm Portfolio Methods. In: AAAI Conference on Artificial Intelligence. pp. 167–172. AAAI Press (2010)
32. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. Journal of Automated Reasoning **59**(4), 483–502 (2017)
33. Tammet, T.: Gandalf. Journal of Automated Reasoning **18**(2), 199–204 (1997)
34. Thornton, C.: Parity: The problem that won't go away. In: Advances in Artifical Intelligence. pp. 362–374. Springer Berlin Heidelberg (1996)
35. Tipping, M.E.: Sparse Bayesian learning and the relevance vector machine. Journal of machine learning research **1**, 211–244 (2001)
36. Trauth, M.H.: MATLAB® Recipes for Earth Sciences. Springer International Publishing (2021)

37. Tsarkov, D., Horrocks, I.: FaCT++ Description Logic Reasoner: System Description. In: Automated Reasoning. pp. 292–297. Springer Berlin Heidelberg (2006)
38. Wasserman, L.: All of Statistics: A Concise Course in Statistical Inference. Springer New York (2004)