Evaluating Distributional Predictions of Search Time: Put Up or Shut Up Games

Sean Mariasin[1], Andrew Coles[2], Erez Karpas[3], Wheeler Ruml[4], Solomon Eyal Shimony[1], Shahaf Shperberg[1]

[1]Ben-Gurion University,
[2]King's College London,
[3]Technion,
[4]University of New Hampshire
seanmar@post.bgu.ac.il, andrew.coles@kcl.ac.uk, karpase@technion.ac.il
ruml@cs.unh.edu, shimony@cs.bgu.ac.il, shperbsh@bgu.ac.il

Abstract

Metareasoning can be a helpful technique for controlling search in situations where computation time is an important resource, such as real-time planning and search, algorithm portfolios, and concurrent planning and execution. Metareasoning often involves an estimate of the remaining search time of a running algorithm, and several ways to compute such estimates have been presented in the literature. Many applications actually require a full estimated probability distribution over the remaining time, rather than just a point estimate of expected search time. We study several methods for estimating such distributions, including some novel adaptations of existing schemes. To properly evaluate the estimates, we introduce 'put-up or shut-up games', which probe the distributional estimates without requiring infeasible computation. Our experimental evaluation reveals that estimates that are more accurate in expected value do not necessarily deliver better distributions, yielding worse scores in the game.

Introduction

Problem-solving agents operating in the real world must be able to handle the fact that, as the agent plans, time passes in the real world, potentially affecting agent actions (Boddy and Dean 1989; Russell and Wefald 1991; Shperberg et al. 2019). Such runtime and other computational resource constraints occur in numerous settings, such as robotics, algorithm portfolios, and control of anytime algorithms.

What is common to all these cases is that some search for a solution is performed and the time remaining until the agent must act may or may not be sufficient to complete the search, whether optimally or suboptimally. Knowing whether search failure is imminent is important, as this allows any number of fallbacks to be employed: opting for a suboptimal solution instead of an optimal solution, switching to another search algorithm (e.g. in an algorithm portfolio), beginning to execute a partially developed plan while continuing the search (a risky move, yet justifiable if failure is otherwise near-certain), or even declaring failure early, hoping to 'cut your losses'.

In order to manage such situations rationally, a metalevel controller, or even a human-in-the-loop, can greatly benefit from a reliable prediction of whether such a search effort is about to fail. While there has been quite a body of work on attempting to predict the remaining amount of search effort (Thayer, Stern, and Lelis 2012; Sudry and Karpas 2022), typically these methods deliver a single number corresponding to the expected value, or other best estimate, of this quantity. At best, some schemes provide a measure of variance of the estimate. We argue that for most metareasoning applications this is insufficient, and that a distribution over the search effort is needed. For example, situated planners (that plan 'online' while time passes) must assess the probability that a candidate partial plan will be executable at the time the search will finish, thus requiring a cumulative distribution function (CDF) over remaining search time. Currently, such planners use only rudimentary distribution estimates, such as one based on one-step-error (Shperberg et al. 2021).

The project of developing distributional estimators immediately raises the question of how such estimators should be evaluated. Testing in the context of metalevel control is not attractive, as most metareasoning control schemes are quite complicated and hard to analyze, often requiring solving an MDP or even POMDP based on the remaining-runtime distributions (Shperberg et al. 2019). Slightly easier to examine is the notion of a stopping criterion in metareasoning for search (Hay et al. 2012; Russell and Wefald 1991), but even this simpler notion requires, in addition to the distributions, utilities of complicated scenarios and/or making unrealistic assumptions (such as the 'meta-greedy' and 'single step' assumptions). It is much easier to examine the accuracy of the distribution estimate in isolation. Here we encounter a difficulty, as a ground truth against which to measure such a distribution is not available. That is because, with a deterministic algorithm solving a given problem instance, the length of the remaining search is determined but unknown. The distribution of the remaining search time that we wish to model is, given the observations so far, over all possible problem instances consistent with the observations, which is not realistically obtainable.

Instead, the fundamental insight of this paper is to exploit the subjectivist Bayesian interpretation of probability. That is, a rational agent that believes with probability p that it will complete the search within a given time t must be willing to bet that it will do so just when it is presented with a bet requiring any payment less than p to gain a reward of 1 if search indeed completes on or before t. Accordingly, our first contribution is to introduce a simple 'put up or shut up game' that embodies this notion at varying odds and examine existing schemes with respect to how well they perform in this game. For concreteness, we examine in this paper search effort predictors for the standard A^* search algorithm (Hart, Nilsson, and Raphael 1968).

Our second contribution is to show, by example and empirically, the importance of modeling the entire remaining-runtime distribution. In particular, we can see that in some cases, runtime predictors that are more accurate in predicting the expected value can perform worse than predictors that are less accurate.

Our third contribution is to develop methods for creating new distribution estimates using ideas from existing estimators, and also adapting some standard learning schemes to predict the distributions. The new predictors are evaluated empirically on how they score in this game versus existing predicting schemes and in most cases show a marked score improvement compared to these baselines.

Put Up or Shut Up Games

Consider the following fundamental game, called 'put up or shut up', designed to measure the correctness of a subjectivist probability estimate delivered by a metareasoner. A given search algorithm (such as A^*, with a known heuristic h) runs on a given problem instance from a given domain. A metareasoner observes the search algorithm run up to a certain point and then, given a deadline, has to bet whether to 1) quit (shut up) or 2) pay a sum (put up) and collect a reward if the search ends successfully before the deadline. Note that this will *not* necessarily be an even bet, i.e. the gain is not necessarily equal to the initial payment. We will demonstrate that success hinges on having a reliable error model; merely achieving a better prediction (e.g. lower root-mean-square error (RMSE) on a numerical estimate) is insufficient.

Formally, we define the simplest game variant:

Definition 1 (BPSG). *Basic put-up or shut-up game: given a search algorithm A running on a problem instance I, some observations O, a remaining time target t. Should we (shut up) stop the computation, avoiding any cost or gain, or (put up), by paying an ante of θ, get a known reward of R (thus net gain $R - \theta$) just when A solves instance I before t time passes?*

To play the game rationally, we must estimate the subjective probability $\hat{p} = P(t(A, I) \leq t|O)$, the probability that algorithm A will solve instance I in remaining time $t(A, I)$ that is less than t, given our observations O. The rational agent should not put up unless $\hat{p} \geq \frac{\theta}{R}$. The latter ratio is also called *betting odds* of θ to $R - \theta$.

Even BPSG can have several variants. We could be in a situation where we have not started to run A at all, but have been given some features of I. Or we could have already run A for some time and made additional observations about the search. We can also assume certain types of algorithms. Another dimension is the amount of information the estimator can use: is it purely online, or is it allowed to learn algorithm behavior from other instances in the same (or a different) domain?

We assume in this paper that we have an expansion-step-based search algorithm and that time is in units of a single expansion step. We examine scenarios in which A has already executed X expansion steps and we are asked for the probability that it will conclude the search on or before some given Y additional expansion steps.

As explained in the introduction, for many applications it is important to have a realistic distribution estimate, as it is even possible for a better predictor (lower error) to result in worse performance than a higher-error predictor with a realistic error model:

Theorem 1. *There exist cases where a predictor more accurate in expected error will perform on average worse in BPSG than a predictor with a higher expected error.*

Proof: It is sufficient to show an example, so consider the following scenario: A has run on I for some number of expansions and we are now asked whether it will finish within 100 expansions. Suppose that the search algorithm will run for another 100 or 101 expansions, each with probability 0.5. This information is unknown to the metareasoning agent, which thus relies on predictors to gauge the anticipated remaining search duration, thereby informing its decision on whether to prolong or terminate the search. Further assume that we have two predictors: predictor **a** estimates only the mean number of expansions, but does so rather accurately, with an unbiased error of at most 1 (uniformly distributed). Importantly, the agent is unaware of the true underlying error model, it receives only the estimation of the mean number of expansions and assumes that predictor **a** delivers exactly the correct number.[1] That is, when the true remaining number of expansions is 100, **a** will predict either 99, 100, or 101, each of these predictions having a probability of $\frac{1}{3}$ of being made, and the agent will believe that the predicted value is exact. Predictor **b** has an unbiased error of at most 2 expansions, uniformly distributed. Yet, unlike **a**, predictor **b** outputs a distribution. The returned distribution results from randomly drawing a value uniformly from within the range of ± 2 expansions from the true value, then constructing a uniform distribution within ± 2 around the predicted value. For instance, if the true expansion value is 100, Predictor **b** would base its prediction around a value $v \in [98, 99, 100, 101, 102]$, each having a probability of being the base of $\frac{1}{5}$, and then **b** returns a uniform distribution over the 5 values $[v-2, v-1, v, v+1, v+2]$. Below, we use the term *centered around* v to refer to the latter distribution.

First, consider the case of even odds, i.e. the agent can pay an ante $\theta = 1$ to obtain a reward of $R = 2$ if the search terminates before the target remaining time. The metareasoner, equipped with predictor **a**, will behave as follows. If the true value is 100 (probability 0.5), **a** will predict 99 or 100 with probability total $\frac{2}{3}$ in this case, and because it is sure its prediction is correct, the metareasoner will decide to ante up and collect the reward, gaining 1. If the true value is 101 (probability 0.5), **a** will predict 100 with probability

[1]For simplicity, our exposition refers to predictor error models (that may be unrealistic) rather than a model over priors plus a predictor sensor model.

$\frac{1}{3}$, and the metareasoner, believing that the prediction is correct, will ante up and lose. So at even odds, using predictor **a**, the metareasoner will gain $0.5(\frac{2}{3} - \frac{1}{3}) = \frac{1}{6}$.

With predictor **b**, the agent will behave as follows. With a true value of 100, **b** would return a uniform distribution over $[v-2, v-1, v, v+1, v+2]$ (with $v \in [98, 99, 100, 101, 102]$ each with probability $\frac{1}{5}$). For $v = 98$, the probability of timely completion (100 additional expansions or less) is 1. For $v = 99$, that probability is 0.8, etc. so all in all for $v \in [98, 99, 100, 101, 102]$ the corresponding probabilities of timely completion are $[1, 0.8, 0.6, 0.4, 0.2]$. With the true number of expansions being 101, we have $v \in [99, 100, 101, 102, 103]$ and the corresponding probabilities that the true number of expansions is 100 or less will be $[0.8, 0.6, 0.4, 0.2, 0]$. The decision now depends on the betting odds. At even odds, the metareasoner using **b** will ante up and collect in 3 cases (total probability $\frac{3}{5}$) and will ante up and lose in 2 cases, for a total expected gain of $\frac{1}{10}$.

As expected, at even odds using **a** the agent scores better in expectation than using **b**. With different odds, say an ante of $\theta = 5$ and a reward $R = 6$ (again total gain 1), using **a** the agent makes the same decisions but here gains 1 in 2 cases and loses 5 in one case, for a total expected **loss** of $\frac{1}{2}$. But using **b** the agent only antes up when its subjective probability of winning is better than $\frac{5}{6}$, so refuses to ante unless **b**'s prediction is 98, where it is certain that the true value is no greater than 100; thus its total expected **gain** is $\frac{1}{10}$. Conversely, with an ante of $\theta = 1$ and reward $R = 6$ (potential gain of 5), using **a** again the agent makes the same decisions, so gains 5 in 2 cases (out of 3) and loses 1 in 1 case (out of 3), a total expected gain of $\frac{9}{6}$. With these odds, the metareasoner using **b** always antes up unless the probability of reward is less than $\frac{1}{6}$, which occurs only when **b** predicts 103. So with **b** the gain is 5 in 5 cases (out of 5), and the loss is 1 in 4 cases (out of 5), for a total expected gain of $\frac{21}{10}$ and again better than one using **a**. $\square$

Note that at uneven odds predictor **b** is better than **a**, despite **a** being on the average closer than **b** to the true value. That is because **a**'s error model was overly optimistic in the quality of its estimate. Providing a reliable error model (a distribution) allows an agent using predictor **b** to achieve a higher expected gain at varied odds.

Background: Existing Estimation Methods

Several methods for predicting remaining search time have been proposed. One approach involves the concept of *expansion delay* (Dionne, Thayer, and Ruml 2011), denoted as t_{exp}, which represents the time (measured in expansion or real-time units) between when a node is expanded and when its parent was expanded. Let N denote the nodes that have been expanded during a particular search process, and consider t_{exp} as the anticipated average expansion delay of the nodes on the path leading to the goal discovered by the search algorithm. Let d^* be the distance to the goal from the most recently expanded node on that path. If that node has just been expanded, then it is evident that the anticipated number of future expansions in the search equals $t_{\text{exp}} \cdot d^*$. Obviously, neither t_{exp} nor d^* are known until the conclusion of the search, thus necessitating estimation during the search process. Specifically, d^* is estimated using the provided heuristic function, resulting in $\hat{d^*}$, while t_{exp} is estimated based on the average time between the generation and expansion of nodes in N, yielding $\hat{t_{\text{exp}}}$. Consequently, an estimate of the remaining search effort is derived by multiplying these two estimated quantities:

$$\hat{t} = \hat{d^*} \cdot \hat{t}_{\text{exp}} \tag{1}$$

Another scheme measures *velocity*, which is the expected decrease of the minimum h value on the search's open list per expansion (Hiraishi, Ohwada, and Mizoguchi 1998). Given the average future velocity of the search V, the overall number of remaining expansions can be computed as $\frac{h_{\text{min}}}{V}$, where h_{min} is the minimal h-value in the open list. Similar to the expansion delay, V is unknown during the search, but it can be estimated. For example, one can count the number of expansions n required to decrease the minimal heuristic in the open list from the initial heuristic value h_0 to its current value h_{min}. Using this estimate, denoted as $\hat{V}$, the estimate of the remaining search effort can also be estimated:

$$\hat{t} = \frac{h_{\text{min}}}{\hat{V}} = \frac{h_{\text{min}}}{\frac{h_{\text{min}} - h_0}{n}} = \frac{n \cdot h_{\text{min}}}{h_{\text{min}} - h_0} \tag{2}$$

A third method attempts to estimate the current fraction of the way through the total run time the search currently is (measured in number of expansions again) using deep NN learning (Sudry and Karpas 2022).

However, the above methods deliver only a single number, rather than a remaining runtime distribution. The only scheme of which we are aware that does output a distribution is based on the *one-step error* (Dionne, Thayer, and Ruml 2011; Shperberg et al. 2021). The one-step error is defined as the difference between the heuristic value of a node and the minimal heuristic value of its children:

$$\epsilon(n) = \min_{n' \in \text{Children}(n)} (h(n') + c(n, n')) - h(n), \tag{3}$$

where $c(n, n')$ is the cost of the edge between n and n'. Shperberg et al. (2021) collected statistics on the one-step error during the search, creating an empirically obtained distribution denoted as the (iid) random variable(s) X_i. The empirical distribution was initialized with 1000 "ghost" samples of value 0. Then, for a search node v, (assuming unit costs) the estimated number of steps to the goal is the random variable:

$$Y = h(v) + \sum_{i=1}^{h(v)} X_i$$

Under the assumption that all the X_i are jointly independent, so to get the PMF of Y requires convolution operations on the PMF of the X_i. The distribution of the remaining number of expansions is the random variable $t_{\text{exp}}Y$. The intuition behind the one-step-error scheme is that greater one-step-error entails greater overall heuristic error, and also that the variance of the errors increases with $h(v)$, which has been observed in multiple domains. The one-step-error scheme achieves these properties, so is intuitively appealing, although not fully backed by theory.

Note that some offline methods for estimating the number of search steps needed to solve a search problem have been proposed (Everitt and Hutter 2015a,b). Other techniques try to estimate the optimal solution cost (Lelis et al. 2016). However, these techniques do not use information obtained during search, and are thus less relevant to our work.

Proposed Distribution Estimators

In all cases, the semantics of the distribution we wish to model is: what is the distribution of the target quantity among problem instances randomly drawn from a set consistent with the observations made before attempting the prediction? This could be achieved by learning schemes, but note that this is not necessarily the type of distribution we see, e.g., by measuring the variance of these quantities as the search progresses. For example, although a high variance between successive measurements of the expansion delay might suggest a high variance in the future average expansion delay across instances, this correlation is not always guaranteed. Likewise, an estimate of past heuristic error (such as a one-step error) does not inherently translate into an accurate predictor of the variance of d^* from h across instances, even under the assumption of uniform cost. Still, these variances could serve as additional parameters when estimating or learning such distributions.

In order to estimate distributions on the remaining search time, we begin by collecting data by performing search, using some search algorithm A, on n problems uniformly drawn from a domain D. During the search process on some problem instance I_i, we store multiple examples, where each example corresponds to a node n expanded during the search. The information (features) stored for each node n is as follows:

- $g(n)$, the cost of the best-known path to node n.
- $h(n)$, the heuristic estimate of the cost from n to the goal.
- $f(n)$, the priority value for node n (which varies with the search algorithm).
- $b(n)$, the branching factor of n, that is, the number of successors n has.
- $N(n)$, the serial number of n, that is, how many nodes were expanded before n.
- h_0, the heuristic value of the initial state of the problem being solved.
- h_{min}, the minimal h-value we have seen so far among the expanded nodes.
- f_{max}, the maximum priority value seen so far.
- $V(n)$, the search's velocity estimation, that is, $\frac{h_0 - h_{\text{min}}}{N(n)}$.
- $AverageV(n)$, the search's average velocity while expanding n, i.e., $\frac{\sum_{n'=1}^{N(n)} V(n')}{N(n)}$.
- $t_{\text{exp}}(n)$, The expansion delay of node n, meaning, the number of expansions between n's expansion and its parent's expansion.
- Average t_{exp}, the average expansion delay over all expansions, $\frac{\sum_{n'=1}^{N(n)} t_{\text{exp}}(n')}{N(n)}$.
- The one-step error, $\epsilon(n)$ (as per Eq. 3).
- Average ϵ, the average one-step error over all expansions, $\frac{\sum_{n'=1}^{N(n)} \epsilon(n')}{N(n)}$.

The first eight features were used in the work of Sudry and Karpas (2022), whereas the last six features were used in the expansion delay and the velocity-based predictors.

In addition to the features, we also store the "ground-truth" remaining search time for each node. Let $A(I_i)$ represent the number of nodes expanded by algorithm A when solving problem instance I_i. For each node n expanded during the execution of A on I_i, we record the number of nodes expanded after n during the search ("remaining time"), computed as $y^i_{\text{rt}} = A(I_i) - N(n)$. Similarly, we also maintain the ground-truth remaining average velocity, $y^i_v = \frac{h_0 - h_{\text{min}}}{y_{\text{rt}}}$ and expansion delay, $y^i_{ed} = \frac{\sum_{j=N(n)}^{A(I_i)} t_{\text{exp}}(j)}{A(I_i) - N(n)}$. It is important to note that y^i_v and y^i_{ed} are not used for prediction, since their value is unknown until the search terminates; rather, they serve as labels during training. We denote the dataset of examples by $D = (X^i, Y^i = (y^i_{\text{rt}}, y^i_v, y^i_{ed}))^m_{i=1}$, where X^i is the set of feature values in example i, and y^i_{rt}, y^i_v, and y^i_{ed} are the ground-truth quantities for this example. Using this dataset of examples D, we introduce several approaches to obtain distributions over remaining search time.

Leveraging Existing Estimators

The approach involves considering t_{exp}, d^*, and V in equations (Eq.1 and Eq.2) as random variables, rather than as directly estimated values. Then it suffices to have a distribution of these random variables (which is obtained by collecting statistics over these quantities) in order to get a distribution of the number of expansions remaining in the search. For instance, when aiming to estimate the distribution over T_r, the remaining search time, using the velocity estimator (Eq.2), one can observe the current search status, denoted by a set of features X. From this, a distribution estimate $\mathcal{V}$ is constructed to describe the probabilistic behavior of the average velocity over future expansions, with the random variable V following this distribution. Then T_r can be seen as a random variable distributed as $\frac{h_{\text{min}}}{V}$.

Considering the current state of the search denoted by X, we propose to estimate the probability $\mathcal{Y}$ concerning the desired quantity (e.g., average velocity or expansion delay), represented by the random variable Y, as follows. Initially, we extract from the dataset all training examples corresponding to the same number of expansions as the ongoing search, i.e. $D_N = \{(X^i, Y^i) | (X^i, Y^i) \in D \text{ and } X^i_{N(n)} = X_{N(n)}\}$. Let X_j represent the value of the feature corresponding to the quantity we aim to measure (e.g., average expansion delay observed thus far), and y_j denote the quantity we aim to estimate (e.g., average velocity or expansion delay across future expansions). We compute the *relative error* of y^i_j to x^i_j, denoted as $z^i_j = \frac{y^i_j - x^i_j}{x^i_j}$, for each example. Subsequently, we construct a distribution $\mathcal{Z}$, corresponding to a random variable Z, by forming a histogram using the values z_j^i for all $(X^i, y^i) \in D$. Consequently, given X_j, we define $Y = Z \cdot (X_j + 1)$, as our distribution estimate.

In practice, the distribution $\mathcal{Z}$ was further preprocessed by binning and smoothing. Specifically, we used 100 bins. Smoothing was done by using a discrete linear convolution of the $\mathcal{Z}$ distribution with an approximate one-dimensional Gaussian kernel filter: a sequence of 15 elements with a standard deviation of 2 and an expected value of 0.06.

We obtained distributions as mentioned above for average future expansion delay and for distance to get the *expansion-delay based distribution estimator*, and a distribution over future average velocity to get the *velocity-based distribution estimator*. In addition, we have a *direct estimator* which computes the distribution of y_{rf} from D_n and uses it directly.

Learning a Distribution Estimator

Another approach to estimating the probability of a sample being solved within t remaining time is by treating the problem as a supervised learning task. In this context, we propose to train a probabilistic binary classifier, denoted as $f(X, t)$, based on the dataset D. Here, $f(X, t)$ represents a function that takes as input a set of features corresponding to the current state of the search and the remaining time as a query, and outputs a probability estimate p that indicates whether the search is expected to terminate within the next t expansions. Unlike the estimation methods discussed earlier, here we do not restrict our dataset to examples with the same number of expansions. However, including all examples in D can introduce a bias toward search processes that involve more node expansions since each expansion corresponds to an example in D. To mitigate this bias, we construct a new training dataset D' comprising 1000 random samples from each search process in the original dataset. We utilize D' to train our classifiers.

To learn an effective classifier, we explored various classification algorithms, including the Adaptive Boost Classifier (Freund and Schapire 1997) and Random Forest Classifier (Ho 1995).

To select the best algorithm and hyperparameters, we employed *Optuna* (Akiba et al. 2019), a hyperparameter optimization framework. Optuna's objective was to identify the learning algorithm and its configuration that maximized a defined score, as in the "Put or Shut Up" game. In order to maintain interpretability and keep complexity low, we restricted the number of decision trees and their maximum depths (regardless of the chosen learning algorithm) to 3 and 5, respectively.

Empirical Evaluation

As previously mentioned, while we want to estimate the distribution of remaining search time, that is infeasible to evaluate empirically. Therefore, we evaluate the performance of the different estimators listed above on the basis of the "put up or shut up" game, and compare it to their performance on predicting remaining search effort.

Evaluation Scheme

For each predictor, we compute two values: its score on the "put up or shut up game", and the root mean squared error (RMSE) of its "progress bar" prediction (Sudry and Karpas 2022). For both of these, we vary N_{cur} – the "current" number of expansions at which we attempt to predict the remaining search time. If a specific problem instance is finished before some given expansion number, we skip that value for that instance.

The progress-bar prediction works as follows: given N_{cur} and a predicted number of remaining future expansions x, the "progress-bar" prediction is $\frac{N_{cur}}{N_{cur}+x}$. This scheme, used by Sudry and Karpas (2022), has the advantage that all progress-bar predictions are in the range $[0, 1]$ and are thus naturally comparable. When using an estimator that returns a distribution, we use the expected value of the predicted distribution as our "progress-bar" prediction. To aggregate errors on different problems and states, we use RMSE.

To compute a score for each predictor F on the "put up or shut up" game, for each value of N_{cur} we also vary

t Remaining search time (measured in units of expansions). We vary t between 100 and 1000000 (increasing by a factor of 10 with each step), trying all possible values.

θ the probability threshold that indicates if the given predicted probability (in solving the sample in remaining time t) is sufficient to "put up" (and to verify if the sample could be solved in remaining time t). We vary θ between $\theta_{\text{min}} = 0.05$ and $\theta_{\text{max}} = 0.95$ in steps of $\delta_\theta = 0.01$.

That is, given a predictor F, we run a search algorithm for N_{cur} steps and predict whether the search will terminate in t more steps. If the predictor estimates the probability of this happening is greater than θ then we "put up", and otherwise we "shut up". Since the important parameter is the probability threshold θ, we set all rewards to 1.

This allows us to compute a $Score$ function as follows:

$$Score(\hat{p}, \theta, y) = \begin{cases} 1-\theta & \text{if } \theta \leq \hat{p}, y = 1 \\ -\theta & \text{if } \theta \leq \hat{p}, y = 0 \\ 0 & \text{if } \hat{p} < \theta \end{cases}$$

where $\hat{p}$ is the probability predicated by our estimator that the search will finish within t steps, and y is an indicator of whether search actually terminated within t steps or not. That is, our score is always 0 when deciding to "shut up", which occurs when $\hat{p} < \theta$. Otherwise, we "put up" and make a net gain of $1 - \theta$ when we win and a net loss of θ when we lose.

Of course, the score above is for a fixed θ and t, so to aggregate the results we average over the different values of θ and t, as well as over the different problems instances and different values of N_{cur} for each problem. The average score is denoted as $MeanScore$.

However, the range of possible average scores does not rely on the predictor's performance alone. The number of instances, each instance's total number of expansions and the given t values all have a significant impact on what would be the optimal score (when every decision is correct using hindsight) and the worst possible score (when every decision is wrong). In order to gain a consistent measure of the performance across different settings, we compute the $RelativeScore$ as follows:

$$RelativeScore = \frac{MeanScore - W}{Opt - W}$$

Where $MeanScore$ is defined as before, Opt is the optimal mean score, and W is the worst possible mean score.

Predictors Compared

We evaluate the performance of the proposed distribution estimators, based on the relative error distribution of 3 different remaining search time estimators: expansion delay, velocity, and direct, giving us 3 different predictors, which we label de_{ed}, de_v, and de_d respectively.

As baselines, we take standard remaining search effort predictors, and convert them to distribution predictors by constructing a degenerate distribution which is certain the prediction is absolutely correct. That is, if M is the expected or predicted remaining number of steps, the predicted probability $\hat{p}$ of concluding the search within t expansions or less is $\hat{p} = 1$ if $M \leq t$, and 0 otherwise. With this scheme we used expansion delay (denoted d_{ed}), velocity (denoted d_v), and the expectation of the direct estimator (denoted d_d). We also used the neural network based predictor (Sudry and Karpas 2022) (denoted d_{NN}), and one learning random forests (denoted d_{RF}).

In addition, we compared to the distribution derived from the one-step error combined with expansion delay, as suggested by Shperberg et al. (2021) (denoted os_{ed}). For this predictor, we used a moving average over the last 500 nodes to estimate the distribution over the remaining search time.

For all of the predictors above, we compare their score on the “put up or shut up” game and the RMSE of their ”progress bar” prediction.

We also compare to some baseline predictors, which can only participate in the “put up or shut up” game, but do not provide any remaining search effort predictions. These *trivial classifiers* do not use any information about the search, but some do use training set distribution data.

Positive always predicts timely completion, (Denoted +)

Negative predicts search never ends on time, (Denoted -)

Toss a fair coin Predicts 0 or 1 with probability 0.5, (Denoted by .5)

Toss an odds based coin Given a classification threshold θ, predicts failure with probability θ, and success with probability of $1 - \theta$, (Denoted by θ)

Benchmarks

We now describe the search problem instances we used for the empirical evaluation. A search problem instance is defined by a search problem (state space), a heuristic, and a search algorithm.

For search algorithms, we used A^* . For the search problems, we used two different domains: the 15 puzzle and Pancake sorting.

For the 15 puzzle, we used two different heuristics: the Manhattan distance heuristic (MD) and Manhattan distance with Linear conflicts heuristic (MD+LC). With the MD heuristic, we have 104 solved instances, of which 33 are from Korf’s 100 instanced (Korf 1985), and the rest are randomly generated by sampling a random board configuration for the initial state. With the MD+LC heuristic, we have 326 instances, of which 66 are from Korf’s 100 instances and the rest are randomly generated.

For the Pancake sorting instances, the Gap heuristic function (Helmert 2010) was chosen. We have 498 instances of the Pancake sorting puzzle, randomly generated by sampling an initial state.

Experimental Setup

As learning methods, we used the scikit-learn library (Pedregosa et al. 2011) to fit a classifier/regressor to the data and for prediction. For the distribution estimators, we used the average expansion delay t_{exp} and current average velocity V.

In each of the three domains, some instances were randomly assigned to the training set, and the rest to the test set. The training set sizes were as follows. 15 Puzzle (with MD heuristic): 53, 15 Puzzle (with MD+LC heuristic): 136, Pancake: 248.

In addition, since search runs on problem instances of different domains produced search histories whose statistics are of different scales, different values for N_{cur} were used in each domain. For the 15 Puzzle domains, N_{cur} was chosen from the list $[30000, 75000]$. For the Sorting Pancake domain, N_{cur} was chosen from the list [100,250]. t was always chosen from the list $[100, 1000, 10000, 100000, 1000000]$.

In each setting, each method application provided predicted probabilities or labels that were used to compute the score on the basis of the evaluation scheme.

Empirical Results

We now describe the results of our empirical evaluation. First, Figure 1 shows, for each predictor, the RMSE of remaining search effort prediction on the x-axis and the average score in the “put up or shut up” game on the y-axis. Subfigure (a) shows this for the tile puzzle with MD heuristic, (b) for the tile puzzle with MD+LC heuristic, and (c) for the Pancake puzzle with gap heuristic. Note that the trivial estimators do not appear here, and neither do the machine-learning-based estimators. That is because we do not have a way to compute an estimate on remaining number of expansions for these, and thus there is no RMSE.

In all 3 domains of search problem instances, the overall picture is similar: the distribution-based schemes (other than the velocity-based one) are the top contenders w.r.t. score, especially the predictor based on the distribution from the expansion delay (de_{ed}). This supports the premise of this paper, which is that if we want to perform metareasoning, we must use more than standard, single-number, search effort predictors. In addition, we proved (Theorem 1) that there exist cases where better value-prediction does *not* entail a better score, for some betting odds θ to $1 - \theta$. In fact, this also seems to occur w.r.t. score averaged over many values

	Distribution				Direct			ML		Trivial			
Domain	de_{ed}	de_v	de_d	os_{ed}	d_{ed}	d_v	d_d	d_{NN}	d_{RF}	+	-	.5	θ
Pancake Gap	**0.91**	0.83	**0.91**	0.84	0.72	0.67	0.89	0.84	0.9	0.67	0.32	0.49	0.63
15 MD + LC	**0.91**	0.77	0.9	0.87	0.76	0.54	0.9	0.76	0.9	0.16	0.83	0.49	0.63
15 MD	**0.93**	0.77	**0.93**	0.89	0.71	0.5	0.87	0.72	0.92	0.12	0.87	0.52	0.63

Table 1: Mean Relative Score comparison for various benchmarks

of θ. This can be most markedly seen in the pancake domain, where the NN scheme had the lowest RMSE but was not a top contender w.r.t. score.

Second, we compare the performance of the different predictors on the “put up or shut up” game, including also the trivial predictors and the ML-based predictors. Table 1 shows the mean relative score of each predictor in each of our 3 domains, with the best (highest) score in each row highlighted in bold.

Unsurprisingly, the trivial predictors we included as baselines perform poorly. Also not surprising is the fact that the direct estimators did not perform very well either, with the exception of d_d which is close to the best. However, the direct estimators (d_d and de_d) are not normalized to the scale of the estimate, and thus are unlikely to generalize to search problems of different difficulties. This is in contrast to some other estimators, such as de_{ed} and de_{ev}.

Note that our distribution estimator based on expansion delay (de_{ed}) has the best score, outperforming even the ML-based estimators, despite the fact that it uses much less information about the state of the search in its prediction.

Finally, Figure 2 shows the relative score with different values of θ. As the results show, the best estimator out of the direct estimators changes for different thresholds. In particular, the NN-based predictor is among the best for some betting odds, thus further confirming the results of Sudry and Karpas (2022). However, as it does not deliver an error model (i.e. a distribution), the NN predictor seriously degrades at other values of θ. Our new distribution estimators (and especially the one based on expansion delay, which is the best overall) do well and, in addition, are robust to changes of θ, due to their rational modeling of the odds, as expected.

Conclusion

Remaining search effort prediction is important in numerous applications, such as real-time planning and search, algorithm portfolio control, and other metareasoning tasks. While there has been work on predicting this quantity, most existing work delivers an expected value, while a full distribution or a probability of timely termination is needed. Due to the lack of a ground truth of such distributions, in order to evaluate such predictors we defined the put-up or shut-up game, and showed that predictors providing better-expected value prediction do not necessarily perform better in this game.

We developed several new schemes for providing the needed distributions, some based on general learning methods and others based on simpler collections of statistics. In some cases, the learning-based schemes provide better BPSG scores. Nevertheless, we observe that the intended use of these remaining search-time distributions is metareasoning during search, where computing a value given a complicated learned model, such as a NN or random forest, may be too computationally intensive when decisions must be made in real-time. At present, it seems that the simple error-distribution-based schemes, especially those based on expansion delay, were the best overall when also factoring in their simplicity. Still, much work can be done to improve our proposed predictors, such as checking whether it is sufficient to obtain the error distributions based on much less data, thereby allowing a quick-start based on only a few problem instances in a new domain. It might also be desirable to condition the future velocity, the future expansion rate, and the future velocity variables on some features, without resorting to full-blown learning, thus achieving some of the advantages of the learning schemes without their complexity.

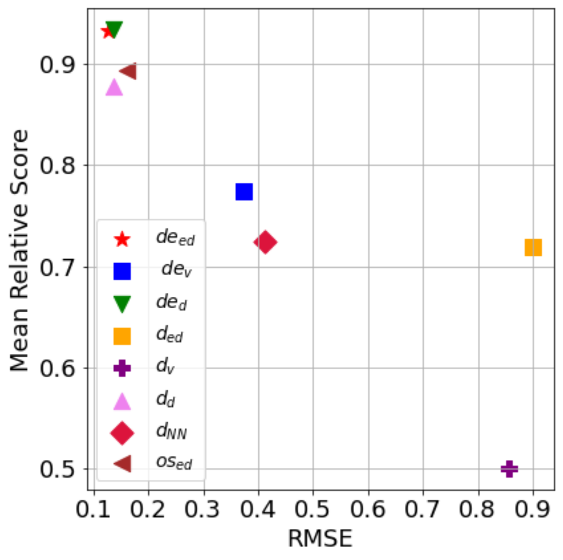
Examining additional settings of online information-gathering would make our work apply to additional metareasoning schemes. BPSG, as stated, is a one-shot game, where one must make only a single metareasoning decision. A more realistic setting is where the search is ongoing, and we must decide whether to continue for a few more search steps or stop. Deciding to continue, we might still decide to stop at some later time given additional observations. Note that here we must have a measurement model that expresses a distribution over future observations, and also how to update the beliefs about runtime given the observations.

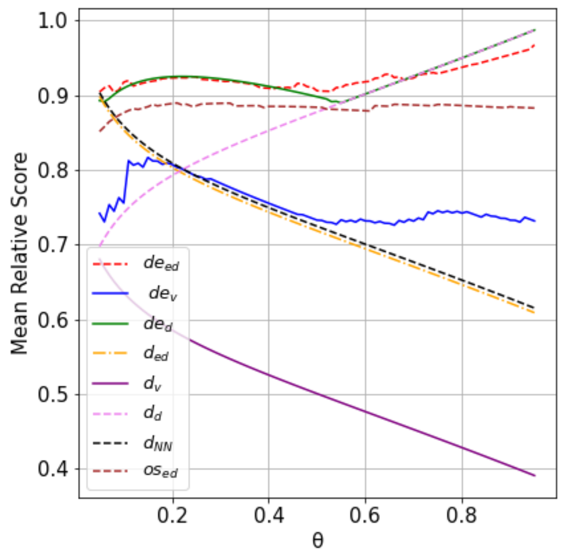
Extending this game to be fully sequential is non-trivial. For example, it was shown by Hay et al. (2012) that even given a very simple measurement model, it is possible for an optimal stopping policy to be infinite. So, in general, optimally solving such a metareasoning problem is hopeless. In our case, we are not even given the measurement model a priori, making the problem even harder to handle.

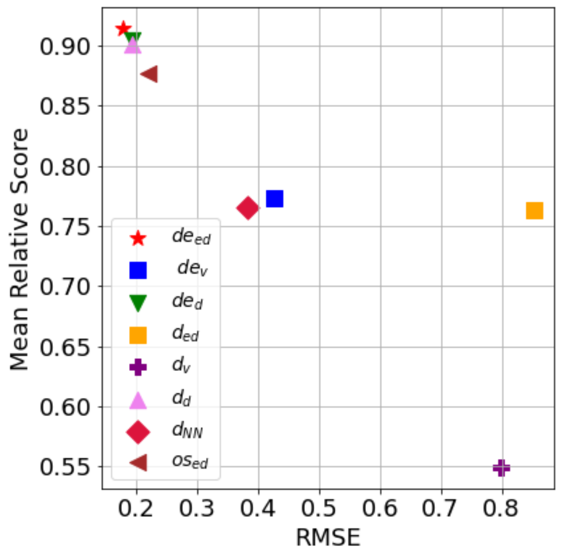
Therefore the following two-shot PSG would be a natural next step. The options are: a) Stop search immediately (“shut up”) and b) Continue search to the bitter end (“put up”) as in the BPSG, but also now c) Probe: search for some t time (or expansions) for a given cost c, and only then make a (final) stopping decision (only one probe allowed). We leave this extension for future work.

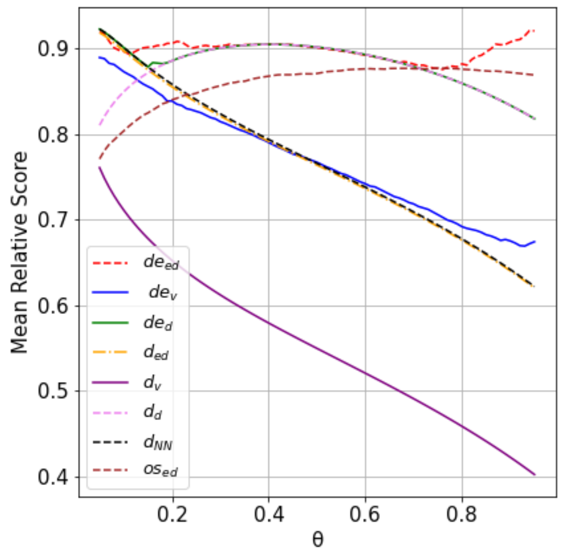
Acknowledgements

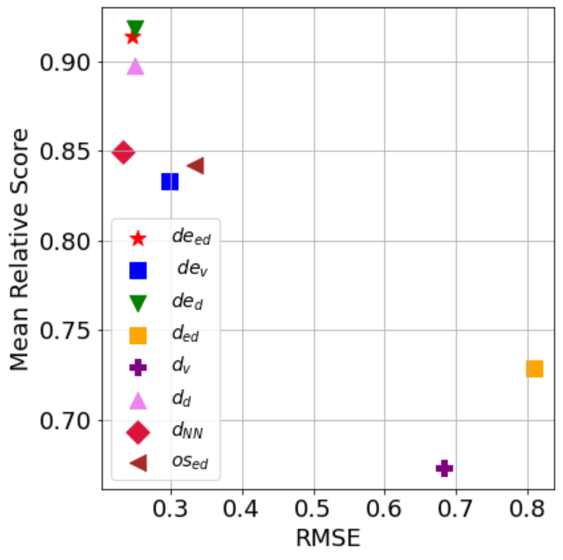
This research was supported by Grant No. 2019730 from the United States-Israel Binational Science Foundation (BSF) and by Grant No. 2008594 from the United States National Science Foundation (NSF).


(a) 15 Puzzle with MD Heuristic


(a) 15 Puzzle with MD Heuristic


(b) 15 Puzzle with MD+LC Heuristic


(b) 15 Puzzle with MD+LC Heuristic


(c) Pancake Sorting with Gap Heuristic

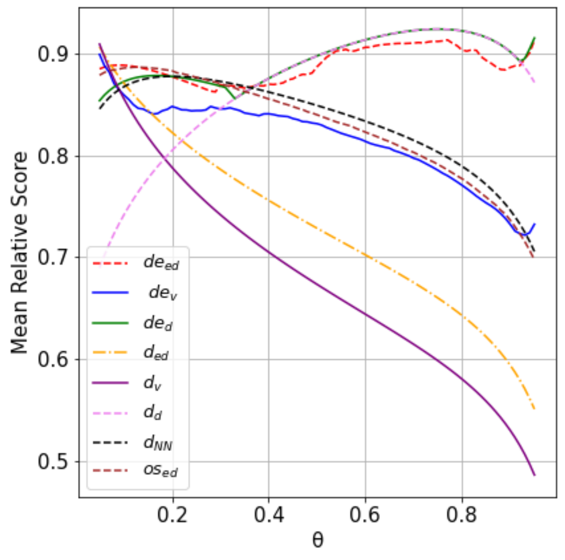

(c) Pancake Sorting with Gap Heuristic

Figure 1: Mean Relative Score vs. progress pred. RMSE

Figure 2: Plots of Relative Score vs. θ, varied benchmarks

References

Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; and Koyama, M. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In *KDD*, 2623–2631. ACM.

Boddy, M. S.; and Dean, T. L. 1989. Solving Time-Dependent Planning Problems. In Sridharan, N. S., ed., *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, 979–984. Morgan Kaufmann.

Dionne, A. J.; Thayer, J. T.; and Ruml, W. 2011. Deadline-Aware Search Using On-Line Measures of Behavior. In *SoCS*.

Everitt, T.; and Hutter, M. 2015a. Analytical Results on the BFS vs. DFS Algorithm Selection Problem. Part I: Tree Search. In Pfahringer, B.; and Renz, J., eds., *AI 2015: Advances in Artificial Intelligence - 28th Australasian Joint Conference, Canberra, ACT, Australia, November 30 - December 4, 2015, Proceedings*, volume 9457 of *Lecture Notes in Computer Science*, 157–165. Springer.

Everitt, T.; and Hutter, M. 2015b. Analytical Results on the BFS vs. DFS Algorithm Selection Problem: Part II: Graph Search. In Pfahringer, B.; and Renz, J., eds., *AI 2015: Advances in Artificial Intelligence - 28th Australasian Joint Conference, Canberra, ACT, Australia, November 30 - December 4, 2015, Proceedings*, volume 9457 of *Lecture Notes in Computer Science*, 166–178. Springer.

Freund, Y.; and Schapire, R. E. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. Syst. Sci.*, 55(1): 119–139.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on System Science and Cybernetics*, SSC-4(2): 100–107.

Hay, N.; Russell, S. J.; Tolpin, D.; and Shimony, S. E. 2012. Selecting Computations: Theory and Applications. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, 346–355.

Helmert, M. 2010. Landmark Heuristics for the Pancake Problem. In *SoCS 2010*, 109–110. SoCS Press.

Hiraishi, H.; Ohwada, H.; and Mizoguchi, F. 1998. Time-Constrained Heuristic Search for Practical Route Finding. In *PRICAI*. Springer.

Ho, T. K. 1995. Random decision forests. In *ICDAR*, 278–282. IEEE Computer Society.

Korf, R. E. 1985. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. In *Artificial Intelligence Volume 27, Issue 1*, 97–109.

Lelis, L. H. S.; Stern, R.; Arfaee, S. J.; Zilles, S.; Felner, A.; and Holte, R. C. 2016. Predicting optimal solution costs with bidirectional stratified sampling in regular search spaces. *Artificial Intelligence*, 230: 51–73.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; VanderPlas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.*, 12: 2825–2830.

Russell, S. J.; and Wefald, E. 1991. Principles of Metareasoning. *Artificial Intelligence*, 49(1-3): 361–395.

Shperberg, S. S.; Coles, A.; Cserna, B.; Karpas, E.; Ruml, W.; and Shimony, S. E. 2019. Allocating Planning Effort When Actions Expire. In *AAAI 2019*, 2371–2378. AAAI Press.

Shperberg, S. S.; Coles, A.; Karpas, E.; Ruml, W.; and Shimony, S. E. 2021. Situated Temporal Planning Using Deadline-aware Metareasoning. In *ICAPS*.

Sudry, M.; and Karpas, E. 2022. Learning to Estimate Search Progress Using Sequence of States. In *ICAPS*.

Thayer, J.; Stern, R.; and Lelis, L. 2012. Are We There Yet? —Estimating Search Progress. In *SoCS*.