Comparing Planners: Beyond Coverage Tables

Caleb Hill[1], **Stephen Wissow**[2], **Wheeler Ruml**[2]

[1]Department of Mathematics and Statistics, [2]Department of Computer Science
University of New Hampshire, USA
caleb.hill@unh.edu, sjw@cs.unh.edu, ruml@cs.unh.edu

Abstract

It is currently common practice in the automated planning community to compare two planners by providing a table of coverage by domain. Surprisingly, there is no commonly-used quantitative test for whether one planner is better than another, even within a single domain. Given the coverage table, researchers merely draw informal conclusions. In this paper, we articulate several desirable features of a formal statistical comparison of planner running times. We evaluate classical statistical tests against these desiderata and we present a new test, Bootstrapped Exponential Estimates (BEE), that is explicitly designed for planners. We hope this work initiates discussion on how the planning community can best formalize quantitative comparison.

Introduction

The purpose of implementing planners and evaluating their empirical behavior, as in the International Planning Competition, is to formally measure which algorithms are faster under what circumstances. ICAPS papers are often full of sophisticated concepts, definitions, and theorems, but current practice in experimental evaluation remains informal. Results are usually presented in terms of coverage, which is the number of planning tasks that were solvable within a per-task time bound. Many researchers assume that, because planner running time often increases exponentially as problems grow in size, solving even a few additional tasks is highly significant. But evaluation methods such as comparing coverage numbers lack nuance in that they do not provide any measure of certainty of the result. Given the usefulness of quantitative statistical analysis in the social and physical experimental sciences, not to mention the rest of computer science and AI, a formal measure suitable for planner benchmarking would likely benefit the planning community.

We fully acknowledge that, in some cases, coverage numbers alone can indeed give a clear picture of which planner is superior. But sometimes they can't. And furthermore, having a sensitive statistical test can bring other benefits. For example, it could be used during experimentation in order to perform just enough tests in order to reach a reliable result, saving the experimenter (or IPC organizer) computation time and therefore energy and likely atmospheric carbon.

We focus on the simplest, most fundamental problem: comparing two planners on a single domain. This is the first step towards comparing more than two planners across multiple domains. Our work aims to develop a metric that is precise, that is based on clearly articulated assumptions, that quantitatively captures uncertainty, and that respects the exponential growth of problem difficulty. Our work is in a preliminary stage and we do not claim to have developed the ultimate statistic. However, our work articulates several of the issues at play and presents two types of statistics that have not been used in the planning community but that we believe hold promise for comparing planners.

Desiderata

We start with a few desired characteristics of a test for comparing two planners. An ideal metric would:

1. be based on precise and easy-to-understand assumptions that are hopefully few in number;
2. return a numerical score that quantifies the certainty with which we can say that planner A is better than planner B;
3. provide an interpretable score that is easy to understand. For example: if assumptions X, Y, and Z hold, then given the data, what is the probability that planner A is better than planner B;
4. take running times as input, rather than just coverage, as this can help distinguish between planners with identical coverage;
5. tolerate censored running times — measurements where all running times above a known time bound were not obtained;
6. take the time bound into account, so that running times that were censored by a high bound would influence the score more than running times censored by a bound that is not as high;
7. take into account the fact that some planning tasks are much harder than others, so if A solves a much harder task than B is able to, that means more than if A were to solve a problem that is only slightly harder than the hardest problem B solved, or an easy problem that B happened to not solve;
8. take into account information about the planning tasks that might help predict their difficulty. For example, the

parameters to a stochastic problem generator often include the number of objects in the domain, or other size information, and task difficulty is often correlated with this;

9. when assessing the difficulty of a task, take into account if multiple planners take a long time (or a short time) on that task, with more planners lending more evidence for the difficulty of that task;
10. in the absence of other evidence, assume that planner running time increases roughly exponentially as problems become larger (we consider it a subjective issue of semantics whether one ascribes this to difficulty increasing exponentially with problem size or to planner running time increasing exponentially with problem difficulty); and,
11. be relatively quick and easy to compute.

We will use these to determine how appropriate a given evaluation metric might be for assessing planner performance.

Any test for detecting which of two planners is better encodes, implicitly or explicitly, a definition of better. One common definition in computer science is: has a running time that grows more slowly. This implies that, for sufficiently large problems and the infinite number of problems larger than that, the better planner will be faster.

Being able to determine which planner is better does not necessarily imply that we are able to predict their running times on new instances, merely which one will be faster. Although if one were able to predict running time, presumably based on some domain features, this clearly implies being able to decide which will be faster. Such detailed predictions might also be useful in gaining more detailed understanding of planner behavior, aiding in algorithm research.

Classical Approaches

In this section, we consider several well-established statistical tests and evaluate them against our list of desiderata. For complete explanations with examples, see Rice (2007); for a practical guide, see Warne (2020).

In the terminology of statistics, our data are paired, meaning that for each planning task, we have the (possibly censored) running times of planner A and planner B.

Sign Test

The sign test is perhaps the simplest approach to comparing planners. This test's null hypothesis is that the median difference in the planners' running times is zero; that is, the probability that planner A solves a problem faster than planner B is 0.5. The sign test treats runtimes from two planners like a Bernoulli random variable, with each paired data point—each of the two planners' runtimes on a single problem instance—as a sample from the random variable. That is, for any two specific paired runtimes y_A and y_B for planners A and B on a specific problem instance, it is either true or false that $y_A < y_B$. The frequency of $y_A < y_B$ being true over the paired data points from all the problem instances in the domain represents the probability that planner A solves a problem faster than planner B.

One strength of the sign test is that it makes very few assumptions. It is non-parametric: because it reduces the pairs of running times down to flips of a coin of unknown bias, it doesn't assume anything about the shape of the distribution from which running times are drawn. It does assume that the probability distribution of signs (i.e., the Bernoulli's probability of heads) is stationary. It can handle censored data when one planner hits the time limit and the other doesn't, as it is clear in such cases which sign is implied. When considering our list of desiderata we see that the sign test only satisfies 1 (has precise, easy-to-understand assumptions), 2 (quantifies uncertainty), 3 (provides interpretable result), 4 (takes runtimes as input), and 5 (handles censored data). Because it ignores the magnitude of the difference in running time between two planners and uses only the sign, we expect it to be a relatively insensitive test. For example, if the timings of planner A are 1, 2, 3 and the timings of planner B are 4, 5, 6, because the test is insensitive to magnitudes it will regard this the same as if B's times had been 100, 200, 300. And if instead B's timings had been 0.5, 1.9, 3000, the test will conclude B is the better planner, ignoring B's significantly worse scaling.

Wilcoxon Rank-Sum Test

The Wilcoxon rank-sum test is a generalization of the sign test. The sign test only requires data to be pairwise compared; the Wilcoxon test gains insight by not only ordering but ranking data. Like the sign test, the Wilcoxon is non-parametric and its null hypothesis is that the probability planner A solves a randomly chosen problem faster than planner B is 0.5. The test works by supposing all the runtimes for two planners are uniformly shuffled. Given this uniform shuffle, one computes the probability of observing the test statistic: the sum of the runtime ranks of, e.g., planner A. If this probability is sufficiently low, the test rejects the random shuffle hypothesis. The Wilcoxon test satisfies desired characteristics 1, 2, 4, and 5. Despite these positive characteristics, the test leaves much to be desired. In particular, as explained by (Hart 2001), it presupposes that the two distributions being compared are only shifts of one another—that is, they have the same shape and spread—which is not guaranteed for planner runtimes. It can be confounded if they have different shape. And like the Sign Test, it is insensitive to runtime magnitude and scaling.

Paired t-test

While the Sign and Wilcoxon tests are nonparametric, meaning they make no assumptions as to the shape of the data they analyze, the paired t-test by contrast assumes that the pair differences are themselves normally distributed. Specifically, the paired t-test's null hypothesis is that the mean of the differences is zero. It tests this hypothesis by considering the differences in running times for the two planners and testing whether the mean of the distribution of the differences is significantly different from zero. Unfortunately, the assumption of normally distributed pair differences does not hold for exponentially-growing runtimes of two planners scaling at different rates, even if one assumes Gaussian noise. Additionally, in order to calculate the differences, the

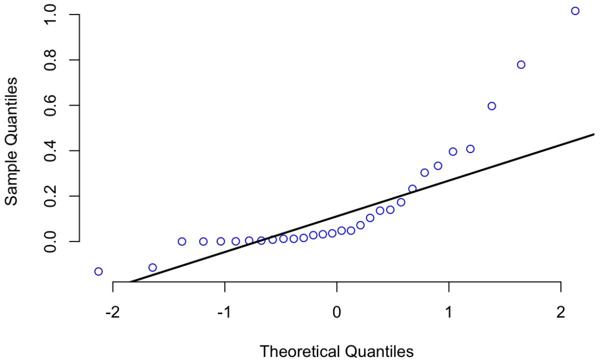

Figure 1: Normal quantile-quantile plot for running time differences between cg and cgpo on the gripper domain.

	cgpo+	cgpo-
cg+	67	2
cg-	1	80

Table 1: Contingency table for cg and cgpo on all autoscale benchmarks with a single linear parameter.

paired t-test requires specific running times for both planners: it cannot handle censored running times.

Experimentally, we observed the assumption of normality to be false. Figure 1 shows a normal quantile-quantile plot for the runtime differences between using greedy search (eager) in FastDownward with the causal graph heuristic with and without preferred operators on the gripper domain (for brevity, we refer to these configurations as ‘cg’ and ‘cgpo’ henceforth).[1] Quantile-quantile plots help show how well data follow a certain distribution. In this case, we compare the empirical runtime differences, whose quantiles are expressed along the vertical axis, with a normal distribution, whose quantiles are expressed along the horizontal axis. The line is a linear fit of the resulting points. If the runtime differences were distributed normally, we would expect the plotted points to follow the line more closely. This makes intuitive sense, as we might expect the gap between two planners to grow as problems become more difficult, rather than having a central tendency.

In summary, the paired t-test, despite its popularity more generally, is inapplicable to planner running times. Also, it only has desired characteristics 1, 2, 4, and 11. It considers neither the size nor difficulty of the tasks solved by the planners.

McNemar’s Test

McNemar’s test analyzes paired coverage data from two planners by using a *contingency table* of those planners’ coverage. For example, Table 1 counts the number of instances solved (“+”) or unsolved (“-”) by each planner The null hypothesis of McNemar’s test is that the probabilities of the off-diagonal cells (i.e., upper-right and lower-left) are equal, which we would interpret to mean that the two planners are equally likely to not solve a problem.

McNemar’s test ignores running time, and is thus insensitive to magnitude and scaling, and it depends on having censored data. The test also lacks ease of interpretation, as one must understand exactly the implications of the null or alternative hypotheses; this is not immediately obvious. For instance, Torralba, Seipp, and Sievers (2021) demonstrate that under a given experimental time limit, one may observe equal coverage even when one planner solved every instance faster than another by a constant factor. In summary, McNemar’s test only satisfies desiderata 2 (quantifies uncertainty) and 5 (handles censored data).

There is precedent for using McNemar’s test in the context of portfolio planning. Seipp et al. (2012) apply McNemar’s test to compare their uniform portfolio with the winner of the 2011 International Planning Competition, and the result of the test, a p-value of $p = 0.0002$, is strong evidence in favor of the claim that their portfolio plans better than the IPC winner. This claim, however, does not directly address the system’s parameters.

All of the above statistical tests are focused on detecting whether or not one planner is faster than another. They do not attempt to quantify how much their performance differs.

A Generative Model of Running Time

We present a model that has desired characteristics 1–5, 10, and 11. In concert with desideratum 10, we assume that *problem difficulty* increases exponentially as problem size increases. This is a common assumption: Torralba, Seipp, and Sievers (2021) bake this assumption into their definition of what they call ‘linear parameters’. We will assume that difficulty increases exponentially along some one-dimensional subspace of the parameter space of each domain’s problem generator. We assume additionally that planner running time is linear in difficulty, although one might imagine other monotone choices.

Let $\mathbb{I}$ be the collection of problem instances belonging to a given domain, and let $\mathbb{T} = (0, \infty)$ be the set of possible running times. Informally, G takes in a random seed r and a N-tuple of parameters ρ and returns a problem to be solved.

Definition 1. *A* **problem generator** *is a stochastic function* $G : \mathbb{N} \times \mathbb{R}^N \to \mathbb{I}$.

Once a problem has been generated, Φ then solves the problem in some amount of time.

Definition 2. *A* **planner running time function** *is a function of instances* $\Phi : \mathbb{I} \to \mathbb{T}$.

When there is no danger of confusion, we will conflate a planning algorithm with its running time function.

Here comes our major assumptions about planners’ interactions with problems. Let $\mathbb{D} = (0, \infty)$. We will use $\mathbb{D}$ as a measure of the difficulty of a given problem instance.

Definition 3. *A* **difficulty function** *is a function* $d : \mathbb{I} \to \mathbb{D}$*, such that for any planner* Φ*,*

$$d(x) \approx d(y) \implies \Phi(x) \approx \Phi(y). \qquad (1)$$

We make the following crucial assumption.

[1] See https://www.fast-downward.org/.

Assumption 1. *Such a difficulty function d exists.*

This allows us to analyse the performance of a planner via its performance on problems of similar difficulty; that is, we may approximately factor a planner running time function through difficulty.

Definition 4. *Given a planner Φ and difficulty function d, a* **pure running time function** *for Φ is a function $\overline{\Phi} : \mathbb{D} \to \mathbb{T}$ such that $\Phi(x) \approx \overline{\Phi}(d(x))$.*

In the definition for a pure running time function for Φ, the equation

$$\Phi(x) \approx \overline{\Phi}(d(x)) \tag{2}$$

may be visualized by considering the following *approximately commuting diagram*[2]

$$\begin{tikzcd} \mathbb{N} \times \mathbb{R}^N \arrow[r, "G"] & \mathbb{I} \arrow[rr, "\Phi"] \arrow[dr, "d"] & & \mathbb{T} \\ & & \mathbb{D} \arrow[ur, "\overline{\Phi}"] & \end{tikzcd} \tag{3}$$

With these definitions, we are interested in the running times $\Phi \circ G(r, \rho)$. Assumption 1 allows us to instead consider $\overline{\Phi} \circ d \circ G(r, \rho)$.

We perform our analysis in the case that $N = 1$, and $\rho = n \in \mathbb{R}$ (in some cases we take ρ to be an element of a one-dimensional subspace of parameter space). Motivated by Equation 2 (the lower triangle of Diagram 3), we make the following additional assumptions.

Assumption 2. *Fix $r = r_0$. Difficulty is approximately exponential in the generator parameter:*

$$d \circ G(r_0, n) \approx a2^{kn} \tag{4}$$

where a and k are positive.

The motivation for Assumption 2 comes from the presence of linear parameters in the work of Torralba, Seipp, and Sievers (2021). These are parameters whose variation, when all others are fixed, results in exponential increase in running time.

Assumption 3. *Pure running time is approximately linear in problem difficulty:*

$$\overline{\Phi}(d) \approx md \tag{5}$$

where m is positive.

In Assumptions 2 and 3 we take "$\approx$" to mean that there are some unmodeled phenomena.

Putting these assumptions together, we get:

Problem 1. *Consider a fixed problem generator G with difficulty function d; fix $r = r_0$. Let $\overline{\Phi}_A$ and $\overline{\Phi}_B$ be two pure running time functions for two planners. Suppose that we have*

$$\overline{\Phi}_A \circ d \circ G(r_0, n) = m_1 2^{k_1 n} \tag{6}$$
$$\overline{\Phi}_B \circ d \circ G(r_0, n) = m_2 2^{k_2 n}$$

Given running time data for both planners, estimate the probability $P(k_1 < k_2)$.

To interpret these equations, we start by noting that the number of nodes in a naive search tree for a planning problem of size n is exponential in n. We can then interpret the slope parameter m as modeling the time it takes a planner to expand a node. As for the scale parameter k, we note that Korf, Reid, and Edelkamp (2001) found that the effect of a more accurate heuristic on a search tree of size $O(b^d)$ is to reduce the depth term d in the exponent. Put simply, a heuristic of average value h resulted in $O(b^{d-h})$ nodes being expanded. Our intuition is that, as size increases, the average heuristic value will as well, resulting in $m2^{kn}$ nodes for some planner-dependent values of m and k.

We observed a degree of subexponential growth in the planner runtimes we examined. Following Janke and Tinsley (2005), we examined other linear models of the form

$$f_1(runningtime) = b + af_2(size) \tag{7}$$

For example, a plot of $\log$ planner running time against $\log$ problem size showed a more linear relationship than when inspecting a plot of $\log$ planner running time against problem size. The left panel of Figure 2 shows one such $\log$-$\log$ plot of running times for ffpo, cgpo, and cg on the Autoscale benchmarks in the gripper domain.

The linearity of the $\log$-$\log$ plots might encourage one to assume the model in Equation 7 where $f_1(x) = f_2(x) = \log(x)$. However, we retain the model proposed in Problem 1 for two reasons. First, the exponential model seems to fit the runtime data better for larger problems than the general linear model: whereas the exponential fit closely matches the runtimes of the larger problem instances in the center panel of Figure 2, the general linear model fit in the right panel appears to diverge from the empirical data as problem size increases. Second, the exponential model aligns more closely with the behavior we would expect from the types of search algorithms under consideration.

Bootstrapped Exponential Estimates

We now introduce a statistical comparison method that instantiates the ideas above into a concrete algorithm, which we call Bootstrapped Exponential Estimates (BEE).[3] In overview: we will fit Eq 6 for each planner and then use a resampling technique called bootstrapping to derive probability distributions that quantify our uncertainty about the value of the scale parameters k. BEE returns an estimate of the probability that one planner's k is less than the other's. We focus on the scale parameter k rather than the slope parameter m as it is more important for asymptotic behavior.

To understand bootstrapping, consider the example of estimating the mean of a Gaussian from n samples. If we didn't have access to a formula for the confidence interval around the sample mean, how might we estimate the uncertainty of our estimate? The uncertainty comes from the fact that, if we were to take another set of n samples, they would almost certainly be different, resulting in a different estimate of the mean. In bootstrapping, we simulate this process many times, creating r new hallucinated datasets, each containing n samples and resulting in its own estimate of

[2] The diagram would be said to *commute* if $\Phi(x) = \overline{\Phi}(d(x))$. The weakening of "$=$" to "$\approx$" is what makes the diagram *approximately commute*.

[3] Code at https://github.com/chill1017/ComparingPlanners.

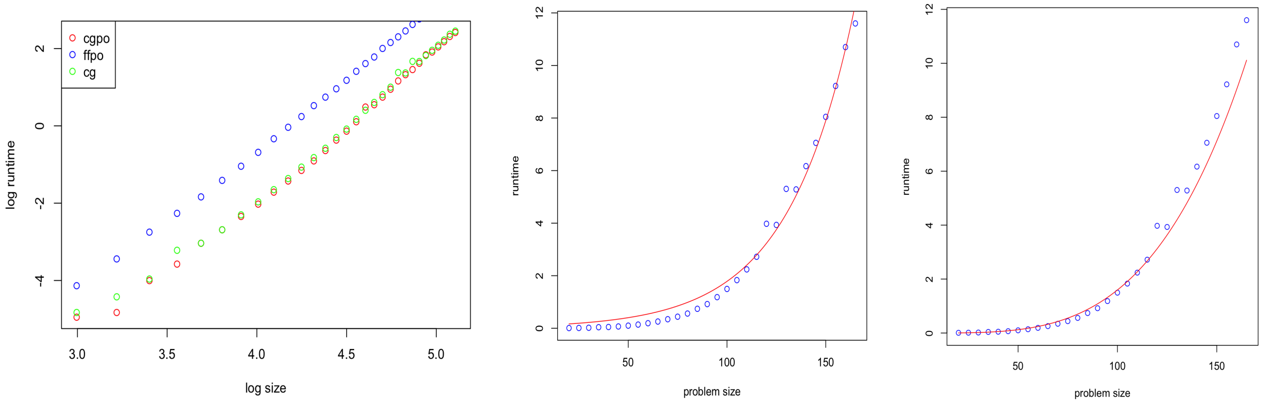

Figure 2: Left: log-log plot of running times. Center: exponential fit. Right: GLM fit for $\log(runtime) = b + a\log(size)$.

```
1. let there be data for planner $i$
2. fit exponentials to get $\hat{m}_i$ and $\hat{k}_i$
3. estimate the residuals of the exponential fits as $\epsilon_i$
4. do $r$ times
5.     simulate data for each planner using $\hat{m}_i$, $\hat{k}_i$, and $\epsilon_i$
6.     fit exponentials and extract estimates for $m_i$ and $k_i$
```

Figure 3: A sketch of BEE.

the mean. To create each dataset, we use the mean and variance of the original data to create a Gaussian from which to sample. The r hallucinated means form a distribution representing the uncertainty of our original sample mean and the 2.5th and 97.5th percentiles would form an estimate of the 95% confidence interval.

BEE uses bootstrapping to estimate the uncertainty surrounding the scale parameter k of the exponential function of Eq 6 for each planner. An overview is given in Figure 3. In a given problem domain, we start by fitting an exponential function to the original runtime data for a given planner. However, instead of simply using the slope and scale parameters of this single exponential fit for comparison with other planners, we estimate our uncertainty using bootstrapping. To create hallucinated data, we first take the residuals from the original fit and use them to estimate the distribution of the unexplained 'noise' in the original data. We then (1) sample from this estimated distribution of runtime noise, and, together with the first exponential fit to the original runtime data, use the sampled noise to (2) hallucinate new running time data. We finally (3) fit a new exponential function to this batch of new, hallucinated runtime data, and record the resulting slope (m) and scale (k) values. The sampling (1), hallucination (2), and fit (3) all occur in each bootstrap iteration, and, for r bootstrap iterations, result in a set of r (m, k) tuples for the given planner. This process is performed for each planner in the given problem domain. Comparing different planners' sets of (m, k) values provides direct answers to questions like $P(m_i < m_j)$ and $P(k_i < k_j)$ for estimated slope m and scale k parameters for algorithms i, j run on problem instances from a given domain.

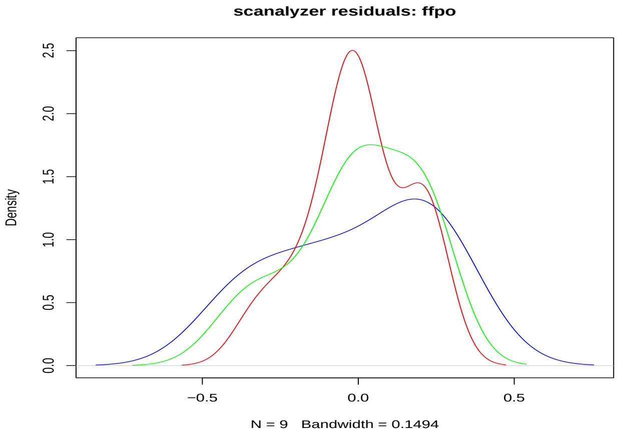

Figure 4: Distributions of additive log-runtime residuals.

Both to perform the first exponential fit to the original runtime data, and also to fit a new exponential function to each bootstrap iteration's batch of hallucinated data, we first transform the data from runtime space into log-runtime space by taking the log of each point's runtime and fitting a line to the result. We then exponentiate the line to produce the exponential fit to the data in runtime space.

We assume multiplicative log-normal noise in the original runtime data (i.e., additive Gaussian noise in log-runtime space), so we estimate the noise Gaussian from the linear fit residuals in log-runtime space; see Figure 4 for a Kernel density estimate of the additive residuals in log-runtime space. To hallucinate new data, we walk along the x values of the original runtime data, computing for each the runtime predicted by the first exponential fit to the original runtime data, and multiply each predicted runtime by the exponentiation of a sample from the noise Gaussian. The result is a new batch of hallucinated data, for the same problem sizes,

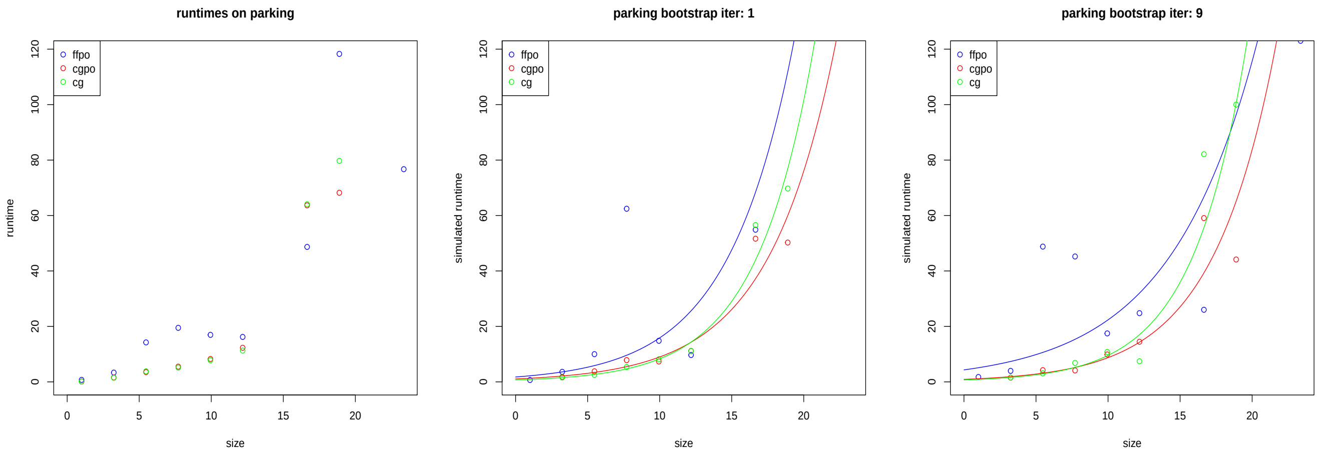

Figure 5: Running times (left) and example bootstrapped data and corresponding fits (center and right).

generated by combining the first exponential fit to the original data with random sampling of the noise distribution estimated from the first fit's residuals. Note that when estimating our residual error model and generating new data, we don't consider any data with $y < 0.25$ out of concern that any measurement noise in the original data could lead to unrealistically large estimates of multiplicative error.

Figure 5 shows an example of bootstrapping using data from the parking domain. The left panel shows the original runtime data for all three algorithm configurations considered on the parking domain. The center and right panels show example hallucinated data and their fits for two different bootstrap iterations. In the original data, it is not obvious which planner is scaling better and we see that the fits in the two hallucinated datasets differ in which curve is the least steep (red in the center and likely blue in the right). This illustrates how the bootstrapped data can be useful in reflecting the uncertainty in the original data, even though the bootstrapped data is hallucinated.

BEE has desired characteristics 1 (precise, easy-to-understand assumptions), 2 (quantifies uncertainty), 3 (provides interpretable result), 4 (takes running times as input), 5 (tolerates censored data), 10 (assumes running times are roughly exponential in problem size), and 11 (quickly and easily computable).

Results

We ran BEE on running times of cg, cgpo, and ffpo on the autoscale benchmarks with a single linear parameter: blocksworld, childsnack, gripper, parking, and scanalyzer. Figure 6 shows the estimated distribution of the scale parameter for each planner on gripper. The probability that BEE assigns to planner A being better than planer B is the probability that a sample from A's distribution is less than a sample from B's distribution.

Table 2 shows the resulting pairwise comparisons. Each entry is the estimated probability that the row planner's scale parameter is lower than that of the column planner's: $P(k_{\text{row}} < k_{\text{column}})$. Every planner had complete coverage in gripper under our 2 min time limit. As Figure 6 suggests, BEE judges cgpo to be likely better than ffpo, and cg is likely better than ffpo. However, the test is not able to confidently distinguish cg and cgpo. Similarly, Table 3 shows the pairwise scale parameter comparisons in the parking domain, a domain in which not all problem instances were solved.

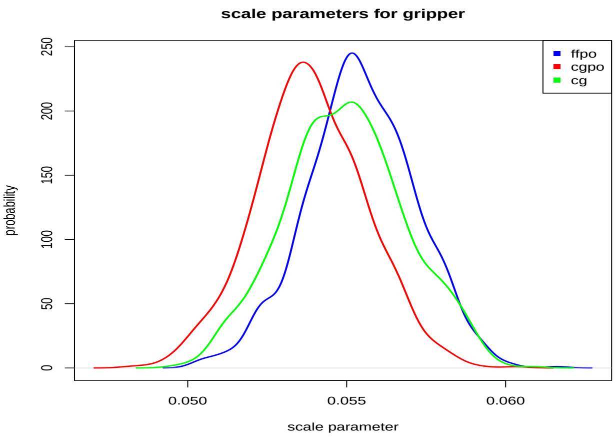

Figure 6: BEE's distributions for planners' scale parameters.

	ffpo	cgpo	cg
ffpo	0.4995	0.107552	0.12147
cgpo	0.892448	0.4995	0.495829
cg	0.87853	0.504171	0.4995

Table 2: Pairwise comparisons of ffpo, cgpo, and cg's scale parameters on the gripper domain.

Next, we investigate the sensitivity of BEE as the number of planning tasks solved increases. Figure 7 shows BEE (left panel) and the sign test (right panel) as we consider longer and longer prefixes of the running times data. The right panels show the p-value for given sample sizes at a standard

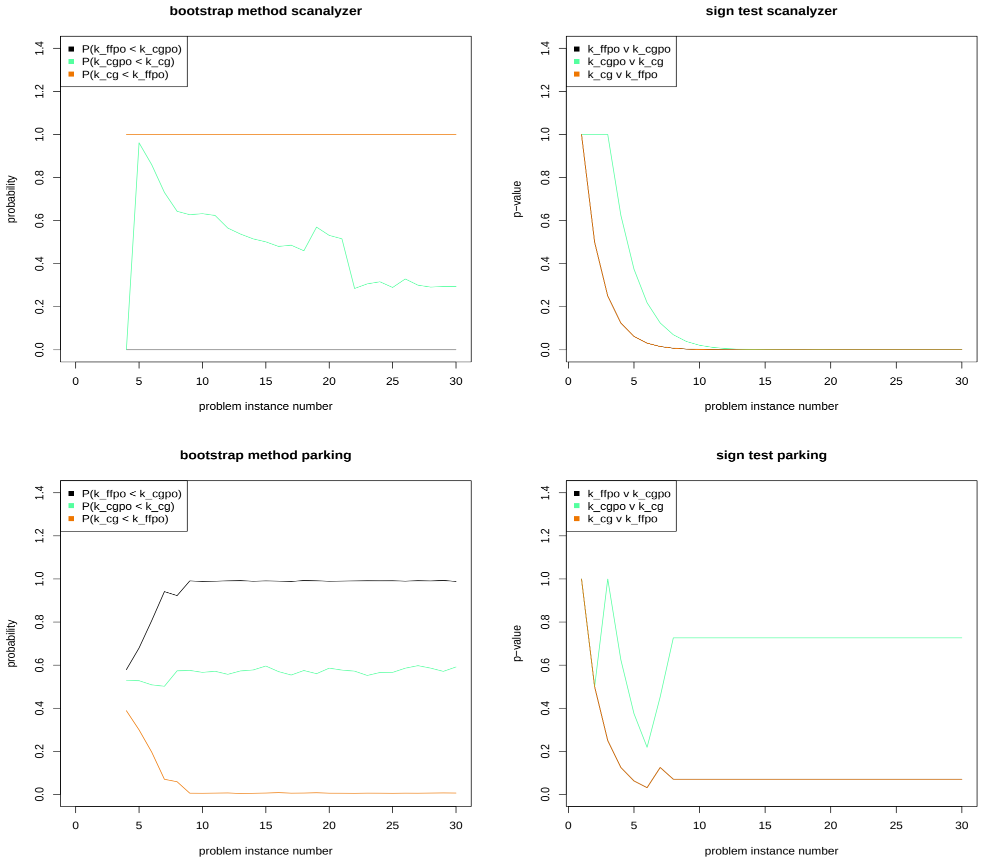

Figure 7: Comparisons as more data are considered: BEE (left) and sign test (right); scanalyzer (top) and parking (bottom).

	ffpo	cgpo	cg
ffpo	0.4995	0.933775	0.957738
cgpo	0.066225	0.4995	0.668305
cg	0.042262	0.331695	0.4995

Table 3: Pairwise comparisons of ffpo, cgpo, and cg's scale parameters on the parking domain.

significance level $\alpha = 0.05$. The left panels show the probability estimates for $P(k_{\text{cg}} < k_{\text{ffpo}})$, $P(k_{\text{cgpo}} < k_{\text{cg}})$ and $P(k_{\text{ffpo}} < k_{\text{cgpo}})$. The subsets of the data we used to generate these probabilities were the first s problem instances, where $s = 4, 5, \ldots, 30$.

The top left plot shows that, in the scanalyzer domain, BEE becomes fully confident that the scale parameters of cg and cgpo are less than that of ffpo after seeing the running times for only the first four planning tasks. On the other hand, the sign test (top right) requires a longer prefix of timing data to become sure of its conclusion. In the parking domain (bottom row), we see that BEE develops certainty that the scale parameters of cg and cgpo are less than that of ffpo more slowly. Interestingly, it consistently returns about a 0.5 probability that the scale parameter of cg is less than that of cgpo. Manual inspection of the runtime data shows that this conclusions appears justified: there is little difference between the two planners' performance.

Future Work

Future work is warranted in improving our modeling of planner data—our bootstrapped data was sometimes unrealistically noisy because poor fits to the original data gave rise to high variance in the estimated error distribution. Examination of the error residuals in these cases suggested that the remaining unmodeled errors could be captured by a quadratic term (e.g., an overall model such as $y = (c_1x^2 + c_2x + c_3)2^{c_4x}$). Figure 8 shows an example in the gripper domain; note the residuals are expressed here as multiples of the original exponential fit's prediction. Having a better model would result in better bootstrap data, resulting in more accurate estimates of uncertainty. However, such a model is also harder to fit.

A Bayesian Model

We briefly discuss an alternative approach to solving Problem 1 by way of Bayesian inference via probabilistic programming. Bayesian inference transforms (possibly uninformative) prior assumptions about parameter distributions into

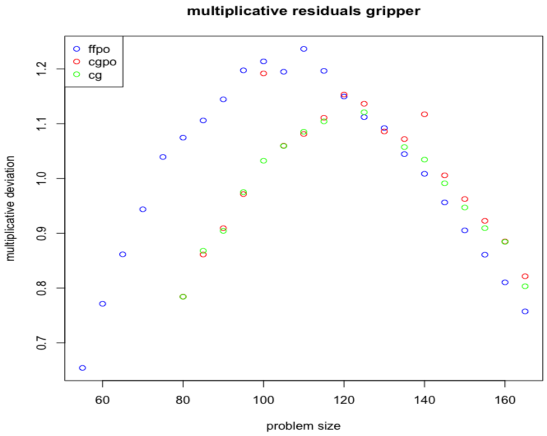

Figure 8: Multiplicative residuals from the exponential fit.

more accurate posterior distributions by conditioning on observed data. The key to this method is Bayes's Theorem, which allows one to compute the informed posterior. The statement boils down to

$$f(\theta|y) \propto f(y|\theta)f(\theta)$$

where $f(\theta)$ denotes the prior distribution on the parameter θ and $f(y|\theta)$ denotes the conditional distribution of the observed data y given a parameter value θ. See Hoff (2009) for more details. The value we are concerned with is $f(\theta|y)$, which is the posterior density for θ values given the observed data. Aside from specifying priors, Bayesian inference software packages, we hoped, would require few initial assumptions.

The Bayesian translation of Assumption 2 is

$$d \circ G(n) \sim N(a2^{kn}, \sigma^2) \tag{8}$$

where a, k have uninformative prior distributions. Combining this with Assumption 3 we get a formulation similar to Problem 1:

$$\overline{\Phi}_A \circ d \circ G(n) \sim N(m_1 2^{kn}, \sigma^2)$$
$$\overline{\Phi}_B \circ d \circ G(n) \sim N(m_2 2^{kn}, \sigma^2)$$

When conditioning on the available running time data, the hope is that we can obtain posterior distributions for each $d \circ G(n)$, m_1, and m_2. The hope of extra flexibility offered by Bayesian inference allows for one to make fewer assumptions on the form of the difficulty and running time functions. Overall, this method has the potential, we believe, to obtain ideal characteristics 1–5 and 9–11.

We were unable to fully implement this approach. Our initial attempt using the Stan language is shown in Listing 1. The problem we found has to do with the need to specify the model in greater detail. While Stan can often sample efficiently, our model is apparently complex enough that one needs to manually specify the likelihood function to aid the sampler. Specifically, a nonlinear variable transformation is needed (Stan Development Team 2024, Section 25.3 'Changes of variables'). Further work is needed to explore the potential of this approach.

Listing 1: Preliminary Stan code for the Bayesian inference approach to planner parameter estimation.

```
functions {
  real PhiBar(int n, real m, real k){
    return m*2^(k*n);
  }
}
data {
  int<lower=0> N;
  vector[N] Y;
}
parameteters {
  real<lower=0> k;
  real<lower=0> m;
  real<lower=0> sigma;
}
model {
  for(n in 1:N)
    Y[n] ~ normal(PhiBar(n,m,k), sigma);
  k ~ uniform(0, 2);
  m ~ uniform(0, 5);
  sigma ~ uniform(0, 1);
}
```

Discussion

An alternative perspective to comparing planners could be to consider them as adversarial players in a game, competing to solve a planning task first. If a given problem instance is considered a match or duel, then the models put forward by Minka, Cleven, and Zaykov (2018) or Elo (1978) may be used to estimate the relative 'strength' or 'win probability' of two planners on a given problem instance. However, even if the prediction of which planner will finish first can be made accurately, it is not obvious if a who-will-win prediction can also capture by-how-much. Even the more general system designed by Cowan (2023), of which TrueSkill (Minka, Cleven, and Zaykov 2018) is a special case, only predicts the win-loss outcome and provides no insight into the generator's or planner's parameters or scaling.

Ideally, a model of planner behavior would predict planner performance on the basis of features of the domain, rather than considering each domain in isolation. This would allow us to understand planner performance more deeply and predict performance in new domains.

Conclusion

We presented desiderata for a quantitative comparison of planner performance and found that classical statistical tests do not meet most of them. We presented a new method, Bootstrapped Exponential Estimates, that focuses on scaling behavior and has desired characteristics 2–5, 10, and 11. The largest drawback at this stage of development is that, as of now, we only have a method that analyzes problems generated from a one-dimensional subspace of problem parameter space. We hope in the future to see extensions of this method that utilize richer models for running time and use higher dimensional subsets of problem parameter space.

Acknowledgements

We are grateful for helpful discussions with Pei Geng, Ernst Linder, Marek Petrik, and Jendrik Seipp and for support from the United States National Science Foundation via NSF grant 2008594.

References

Cowan, A. 2023. Paired comparisons for games of chance. arXiv:2303.14857.

Elo, A. E. 1978. *The Rating of Chessplayers, Past and Present*. Arco Pub.

Hart, A. 2001. Mann-Whitney test is not just a test of medians: differences in spread can be important. *BMJ*, 323(7309): 391–393.

Hoff, P. D. 2009. *A First Course in Bayesian Statistical Methods*. Springer Publishing Company, Incorporated, 1st edition. ISBN 0387922997.

Janke, S. J.; and Tinsley, F. 2005. *Introduction to linear models and statistical inference*. Nashville, TN: John Wiley & Sons.

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of iterative-deepening-A*. *Artif. Intell.*, 129(1-2): 199–218.

Minka, T.; Cleven, R.; and Zaykov, Y. 2018. TrueSkill 2: An improved Bayesian skill rating system. Technical Report MSR-TR-2018-8, Microsoft.

Rice, J. A. 2007. *Mathematical statistics and data analysis*. Brooks/Cole.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning Portfolios of Automatically Tuned Planners. In *Proceedings of ICAPS*, 368–372.

Stan Development Team. 2024. Stan User's Guide. https://mc-stan.org/docs/stan-users-guide/ [Accessed: April 2, 2024].

Torralba, A.; Seipp, J.; and Sievers, S. 2021. Automatic Instance Generation for Classical Planning. In *Proceedings of ICAPS*, 376–384.

Warne, R. T. 2020. *Statistics for the social sciences*. Cambridge University Press, 2 edition.