How to Solve It Automatically: Selection Among Problem-Solving Methods

Eugene Fink
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213
eugene@cs.cmu.edu, http://www.cs.cmu.edu/~eugene

Good ideas are based on past experience.
— G. Polya, *How to Solve It.*

Abstract

The choice of an appropriate problem-solving method, from available methods, is a crucial skill for experts in many areas. We describe a technique for the automatic selection among methods, which is based on a statistical analysis of their past performances.

We formalize the statistical problem involved in selecting an efficient problem-solving method, derive a solution to this problem, and describe a method-selection algorithm. The algorithm not only chooses among available methods, but also decides when to abandon the chosen method, if it proves to take too much time. We give empirical results on the use of this technique in selecting among search engines in the PRODIGY planning system.

1 Introduction

The choice of an appropriate problem-solving method is one of the main themes of Polya's famous book *How to Solve It* (Polya 1957). Polya showed that the selection of an effective approach to a problem is a crucial skill for a mathematician. Psychologists have accumulated much evidence that confirms Polya's pioneering insight: the performance of experts depends on their ability to choose the right approach to a problem (Newell and Simon 1972).

The purpose of our research is to automate the selection of a problem-solving method. This research is motivated by work on the PRODIGY planning system, which includes several search engines (Veloso and Stone 1995). First, we need to provide a mechanism for deciding *which search engine is appropriate* for a given problem. Second, since programs in the real world cannot run forever, we need some means to decide *when to interrupt an unsuccessful search.*

Researchers have long realized the importance of automatic evaluation and selection of search algorithms, and developed techniques for various special cases of this problem. In particular, Horvitz described a framework for evaluating algorithms based on trade-offs between computation cost and solution quality, and used this framework in automatic selection of a sorting algorithm (Horvitz 1988). Breese and Horvitz designed a decision-theoretic algorithm that evaluates different methods of belief-network inference and selects the optimal method (Breese and Horvitz 1990). Hansson and Mayer (1989), and Russell (1990) applied related evaluation and selection techniques to the problem of choosing promising branches of the search space.

Russell, Subramanian, and Parr formalized a general problem of selecting among alternative problem-solving methods and used dynamic programming to solve some special cases of this problem (Russell *et al.* 1993). Minton developed an inductive learning system that configures constraint-satisfaction programs, by selecting among alternative search strategies (Minton 1996).

Hansen and Zilberstein studied trade-offs between running time and solution quality in simple any-time algorithms, and designed a dynamic-programming technique for deciding when to terminate the search (Hansen and Zilberstein 1996). Mouaddib and Zilberstein developed a similar technique for hierarchical knowledge-based algorithms (Mouaddib and Zilberstein 1995).

We found that the previous results are not applicable to the problem of selecting among PRODIGY search algorithms because the developed techniques rely on the analysis of a sufficiently large sample of past performance data. When we apply PRODIGY to a new domain, or use new heuristics or control rules, we usually have little or no prior data. Acquiring more data is impractical, because experimentation is much more expensive then solving a given problem.

We therefore develop a novel selection technique, which makes the best use of the available data, even when they do not provide an accurate estimate. We combine the exploitation of past data with exploration of new alternatives, which enables us to collect additional performance data as the system solves given problems.

We also consider the task of setting a time bound for the chosen problem-solving method. The previous results for deciding on the time of terminating any-time algorithms are not applicable to this task because our problem solvers do not use any-time behavior and do not satisfy the assumptions used in the past studies. We provide a statistical technique for selecting time bounds. We demonstrate that determining an appropriate bound is as crucial for efficient problem solving as choosing the right method.

Our techniques are aimed at selecting a method and time bound *before* solving a given problem. We do not provide a means for switching a method or revising the selected bound *during* the search for a solution. Developing such a means is an important open problem.

Even though the selection among PRODIGY algorithms provided a motivation for our work, the *developed technique does not rely on specific properties of the* PRODIGY *system.* The selection algorithm is applicable to choosing among multiple problem-solving methods in any AI system. It is equally effective for small and large-scale domains. The selection takes little computation and its running time is usually negligible compared to the problem-solving time.

We first formalize the statistical problem of estimating the expected performance of a method, based on the past experience (Section 2). We derive a solution to this problem (Section 3) and use it in selecting a method and time bound (Section 4). We then describe the use of a heuristical measure of problem complexity to improve the performance estimate (Section 5). Note that we do not need a perfect estimate; *we only need accuracy sufficient for selecting the right method and for setting a close-to-optimal time bound.*

We give empirical results on the use of our technique to select among PRODIGY planning engines. We show that it chooses an appropriate engine and time bound. The time of making a selection is three orders of magnitude smaller than PRODIGY's planning time (Section 6).

The generality of our statistical technique makes it applicable to practical problems outside artificial intelligence. We illustrate it by applying the learning algorithm to decide how long one should wait on the phone, before hanging up, when there is no answer.

2 Motivating Example

Suppose that we use PRODIGY to construct plans for transporting packages between different locations in a city (Veloso 1994). We consider the use of three planning methods. The first of them is based on several control rules, designed by Veloso (1994) and Pérez (1995) to guide PRODIGY's search in the transportation domain. This method applies the selected planning operators as early as possible; we call it APPLY. The second method uses the same control rules and a special rule that delays the operator application and forces more emphasis on the backward search (Veloso and Stone 1995); we call it DELAY. The distinction between APPLY and DELAY is similar to that between the SAVTA and SABA planners, implemented by Veloso and Stone (1995) . The third method, ALPINE (Knoblock 1994), is a combination of APPLY with an abstraction generator, which determines relative importance of elements of a planning domain. ALPINE first ignores the less important elements and builds a solution outline; it then refines the solution, taking care of the initially ignored details.

Experiments have demonstrated that delaying the operator execution improves efficiency in some domains, but slows PRODIGY down in others (Stone *et al.* 1994); abstraction sometimes gives drastic time savings and sometimes worsens the performance (Knoblock 1991; Bacchus and Yang 1992). The most reliable way to select an efficient method for a given domain is by empirical comparison.

The application of a method to a problem gives one of three outcomes: it may solve the problem; it may terminate with failure, after exhausting the available search space without finding a solution; or we may interrupt it, if it reaches some pre-set time bound without termination. In Table 1, we give the results of solving thirty transportation problems, by each of the three methods. We denote successes by *s*, failures by *f*, and hitting the time bound by *b*. Note that our data are only for illustrating the selection problem, and *not* for the purpose of a general comparison of these planners. Their relative performance may be very different in other domains.

#	time (sec) and outcome			# of
	APPLY	DELAY	ALPINE	packs
1	1.6 s	1.6 s	1.6 s	1
2	2.1 s	2.1 s	2.0 s	1
3	2.4 s	5.8 s	4.4 s	2
4	5.6 s	6.2 s	7.6 s	2
5	3.2 s	13.4 s	5.0 s	3
6	54.3 s	13.8 f	81.4 s	3
7	4.0 s	31.2 f	6.3 s	4
8	200.0 b	31.6 f	200.0 b	4
9	7.2 s	200.0 b	8.8 s	8
10	200.0 b	200.0 b	200.0 b	8
11	2.8 s	2.8 s	2.8 s	2
12	3.8 s	3.8 s	3.0 s	2
13	4.4 s	76.8 s	3.2 s	4
14	200.0 b	200.0 b	6.4 s	4
15	2.8 s	2.8 s	2.8 s	2
16	4.4 s	68.4 s	4.6 s	4
17	6.0 s	200.0 b	6.2 s	6
18	7.6 s	200.0 b	7.8 s	8
19	11.6 s	200.0 b	11.0 s	12
20	200.0 b	200.0 b	200.0 b	16
21	3.2 s	2.9 s	4.2 s	2
22	6.4 s	3.2 s	7.8 s	4
23	27.0 s	4.4 s	42.2 s	16
24	200.0 b	6.0 s	200.0 b	8
25	4.8 s	11.8 f	3.2 s	3
26	200.0 b	63.4 f	6.6 f	6
27	6.4 s	29.1 f	5.4 f	4
28	9.6 s	69.4 f	7.8 f	6
29	200.0 b	200.0 b	10.2 f	8
30	6.0 s	19.1 s	5.4 f	4

Table 1: Performance of APPLY, DELAY, and ALPINE on thirty transportation problems.

A glance at the data reveals that APPLY's performance in this domain is probably best among the three. We use statistical analysis to confirm this intuitive conclusion. We also show how to choose a time bound for the selected method.

We may evaluate the performance along several dimensions, such as the percentage of solved problems, the average success time, and the average time in case of a failure or interrupt. To compare different methods, we need to specify a utility function that takes into account all these dimensions.

We assume that we have to pay for running time and that we get a certain reward R for solving a problem. If a method solves the problem, the overall gain is $(R - time)$. In particular, if $R < time$, then the "gain" is negative and we are better off not trying to solve the problem. If the method fails or hits the time bound, the "gain" is $(-time)$. We need to estimate the expected gain for all candidate methods and time bounds, and select the method and bound that maximize the expectation, which gives us the following statistical problem.

Problem *Suppose that a method solved n problems, failed on m problems, and was interrupted (upon hitting the time bound) on k problems. The success times were $s_1, s_2, \ldots, s_n$,* *the failure times were $f_1, f_2, ..., f_m$, and the interrupt times were $b_1, b_2, ..., b_k$. Given a reward R for solving a new problem and a time bound B, estimate the expected gain for this reward and time bound, and determine the standard deviation of the estimate.*

We use the *stationarity assumption* (Valiant 1984): we assume that the past problems and the new problem are drawn randomly from the same population, using the same probability distribution. We also assume that the method's performance does not improve over time.

3 Statistical Foundations

We now derive a solution to the statistical problem. We assume, for convenience, that success, failure, and interrupt times are sorted in the increasing order; that is, $s_1 \leq ... \leq s_n$, $f_1 \leq ... \leq f_m$, and $b_1 \leq ... \leq b_k$. We first consider the case when the time bound B is no larger than the lowest of the past bounds, $B \leq b_1$. Let c be the number of success times that are no larger than B; that is, $s_c \leq B < s_{c+1}$. Similarly, let d be the number of failures within B; that is, $f_d \leq B < f_{d+1}$.

We estimate the expected gain by averaging the gains that would be obtained in the past, if we used the reward R and time bound B. The method would solve c problems, earning the gains R–s_1, R–s_2, ..., R–s_c. It would terminate with failure d times, resulting in the negative gains $-f_1, -f_2, ..., -f_d$. In the remaining $n+m+k-c-d$ cases, it would hit the time bound, each time earning $-B$. The expected gain is equal to the mean of all these $n+m+k$ gains:

$$\frac{\sum_{i=1}^{c}(R-s_i) - \sum_{j=1}^{d} f_j - (n+m+k-c-d)B}{n+m+k}. \qquad (1)$$

Since we have computed the mean for a random selection of problems, it may be different from the mean of the overall problem population. We estimate the standard deviation of the expected gain using the formula for the deviation of a sample mean:

$$\sqrt{\frac{Sqr - Sum^2/(n+m+k)}{(n+m+k)(n+m+k-1)}}, \text{ where} \qquad (2)$$

$$Sum = \sum_{i=1}^{c}(R-s_i) - \sum_{j=1}^{d} f_j - (n+m+k-c-d)B,$$

$$Sqr = \sum_{i=1}^{c}(R-s_i)^2 + \sum_{j=1}^{d} f_j^2 + (n+m+k-c-d)B^2.$$

In Figure 1, we show the dependency of the expected gain on the time bound for our three methods. We give the dependency for three different values of the reward R, 10.0 (dash-and-dot lines), 30.0 (dashed lines), and 100.0 (solid lines). The dotted lines show the standard deviation for the 100.0 reward: the lower line is "one deviation below" the estimate, and the upper line is "one deviation above."

We now consider the case when B is larger than e of the past interrupt times; that is, $b_e < B \leq b_{e+1}$. For example, suppose that we interrupted ALPINE on problem 4 after 4.5 seconds and on problem 7 after 5.5 seconds, obtaining the data shown in Table 2(a), and that we need to estimate the gain for $B = 6.0$.

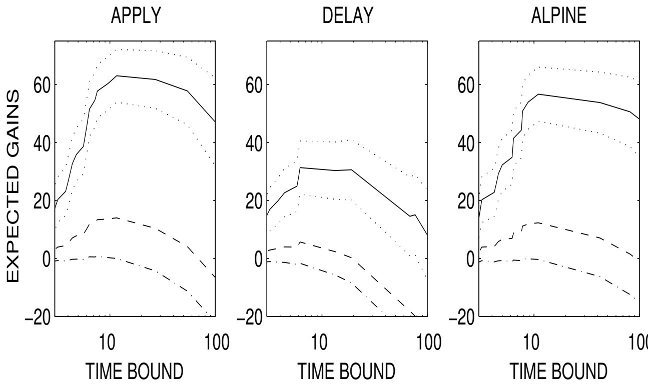

Figure 1: Dependency of the expected gain on the time bound, for rewards of 10.0 (dash-and-dot lines), 30.0 (dashed lines), and 100.0 (solid lines). The dotted lines show the standard deviation of the gain estimate for the 100.0 reward.

(a)

#	ALPINE's time
1	1.6 s
2	2.0 s
3	4.4 s
4	4.5 b
5	5.0 s
6	81.4 s
7	5.5 b
8	200.0 b
9	8.8 s
...	...
29	10.2 f
30	5.4 f

→ (b)

weight	time
1.000	1.6 s
1.000	2.0 s
1.000	4.4 s
–	–
1.048	5.0 s
1.048	81.4 s
1.048	5.5 b
1.048	200.0 b
1.048	8.8 s
...	...
1.048	10.2 f
1.048	5.4 f

→ (c)

weight	time
1.000	1.6 s
1.000	2.0 s
1.000	4.4 s
–	–
1.048	5.0 s
1.118	81.4 s
–	–
1.118	200.0 b
1.118	8.8 s
...	...
1.118	10.2 f
1.048	5.4 f

Table 2: Distributing the chances of interrupt times among the larger-time outcomes.

We cannot use $b_1, b_2, ..., b_e$ directly in the gain estimate, since the use of the time bound B would cause the method to run beyond these old bounds. Instead, we "re-distribute" the corresponding probabilities among the other outcomes.

If we had not interrupted the method at b_1 in the past, it would have succeeded or failed at some larger time, or hit a larger time bound. We may estimate the expected outcome using the data on the past problem-solving episodes in which the method ran beyond b_1. We thus remove b_1 from the sample and distribute its chance to occur among all the higher-time outcomes. In the example of Table 2(a), b_1 is 4.5 and there are 21 problems with larger times. We thus remove 4.5 from the sample data and increase the weights of the larger-time outcomes from 1 to $1 + \frac{1}{21} = 1.048$ (see Table 2b).

We next distribute the weight of b_2 among the larger-than-b_2 times. In our example, b_2 is 5.5 and there are 15 problems with larger times. We distribute b_2's weight, 1.048, among these 15 problems, thus increasing their weight to $1.048 + \frac{1.048}{15} = 1.118$ (Table 2c). We repeat this process for

$b_3, ..., b_e$.

We denote the resulting weights of $s_1, ..., s_c$ by $u_1, ..., u_c$, and the weights of $f_1, ..., f_d$ by $v_1, ..., v_d$. All success, failure, and interrupt times larger than B have the same weight, which we denote by w. We have thus obtained $(n+m+k-e)$ weighted times. We use them to compute the expected gain:

$$\frac{\sum_{i=1}^{c} u_i(R - s_i) - \sum_{j=1}^{d} v_j f_j - (n + m + k - c - d - e)wB}{n + m + k}. \tag{3}$$

Similarly, we use the weights in estimating the standard deviation of the expected gain:

$$\sqrt{\frac{Sqr - Sum^2/(n + m + k)}{(n + m + k)(n + m + k - e - 1)}}, \text{ where} \tag{4}$$

$$Sum = \sum_{i=1}^{c} u_i(R - s_i) - \sum_{j=1}^{d} v_j f_j - (n+m+k-c-d-e)wB,$$

$$Sqr = \sum_{i=1}^{c} u_i(R - s_i)^2 + \sum_{j=1}^{d} v_j f_j^2 + (n+m+k-c-d-e)wB^2.$$

The application of these formulas to the data in Table 2(a), for ALPINE with reward 30.0 and time bound 6.0, gives the expected gain of 6.1 and the standard deviation of 3.0.

We have assumed in the derivation that the execution cost is proportional to the running time. We may readily extend the results to any other monotone dependency between time and cost, by replacing the terms $(R - s_i)$, $(-f_i)$, and $(-B)$ with more complex functions.

Note that we do *not* use past rewards in the statistical estimate. The reward R may differ from the rewards earned on the sample problems. We may extend our results to situations when the reward is a function of the solution quality, rather than a constant, but it works only if this function does *not* change from problem to problem. We replace the terms $(R - s_i)$ by $(R_i - s_i)$, where R_i is the reward for the corresponding problem. The resulting expression combines the estimate of the expected reward and expected running time.

In the full paper (Fink 1997), we describe an efficient algorithm that computes the gain estimates and estimate deviations, for multiple values of the time bound B. The algorithm determines weights and finds gain estimates in one pass through the sorted list of success, failure, and interrupt times, and time-bound values. For M time bounds and a sample of N problems, the time complexity is $O(M + N)$. The complexity of pre-sorting the lists is $O((M + N) \log(M + N))$, but in practice it takes much less time than the rest of the computation. We implemented the algorithm in Common Lisp and tested it on Sun 5. Its running time is about $(M + N) \cdot 3 \cdot 10^{-4}$ seconds.

4 Selection of a Method and Time Bound

We describe the use of the statistical estimate to choose among problem-solving methods and to determine appropriate time bounds. We provide heuristics for combining the exploitation of past experience with exploration of new alternatives.

The basic technique is to estimate the gain for a number of time bounds, for each available method, and select the method and time bound with the maximal gain. For example, if the reward in the transportation domain is 30.0, then the best choice is APPLY with time bound 11.6, which gives the expected gain of 14.0. This choice corresponds to the maximum of the dashed lines in Figure 1. If the expected gain for all time bounds is negative, then we are better off not solving the problem at all. For example, if the only available method is DELAY and the reward is 10.0 (see the dash-and-dot line in Figure 1), we should skip the problem.

For each method, we use its past success times as candidate time bounds. We compute the expected gain only for these bounds. If we computed the gain for some other time bound B, we would get a smaller gain than for the closest lower success time s_i (where $s_i < B < s_{i+1}$), because extending the time bound from s_i to B would not increase the number of successes on the past problems.

We now describe a technique for incremental learning of the performance of available methods. We assume that we begin with no past experience and accumulate performance data as we solve more problems. For each new problem, we use the statistical analysis to select a method and time bound. After applying the selected method, we add the result to the performance data.

We need to choose a method and time bound even when we have no past experience. Also, we sometimes need to deviate from the maximal-expectation selection in order to explore new opportunities. If we always used the selection that maximizes the expected gain, we would be stuck with the method that yielded the first success, and we would never set a time bound higher than the first success time.

We have not constructed a statistical model for combining exploration and exploitation. Instead, we provide a heuristical solution, which has proved to work well for selecting among PRODIGY planners. We first consider the selection of a time bound for a fixed method, and then show how to select a method.

If we have no previous data on a method's performance, we set the time bound equal to the reward. Now suppose that we have accumulated some data on the method's performance, which enable us to determine the bound with the maximal expected gain. To encourage exploration, we select the largest bound whose expected gain is "not much different" from the maximum.

Let us denote the maximal expected gain by $g_{\max}$ and its standard deviation by $\sigma_{\max}$. Suppose that the expected gain for some bound is g and its deviation is σ. The expected difference between the gain g and the maximal gain is $g_{\max} - g$. If our estimates are normally distributed, the standard deviation of the expected difference is $\sqrt{\sigma_{\max}^2 + \sigma^2}$. This estimate of the deviation is an approximation, because the distribution for small samples may be Student's rather than normal, and because $g_{\max}$ and g are not independent, as they are computed from the same data.

We say that g is "not much different" from the maximal gain if the ratio of the expected difference to its deviation is bounded by some constant. We set this constant to 0.1, which tends to give good results: $\frac{g_{\max} - g}{\sqrt{\sigma_{\max}^2 + \sigma^2}} < 0.1$. We thus select the largest time bound whose gain estimate g satisfies this condition.

We present the results of this selection strategy in Figure 2.

We ran each of the three methods on the thirty transportation problems from Table 1, in order. The horizontal axes show the problem's number (from 1 to 30), whereas the vertical axes are the running time. The dotted lines show the selected time bounds, whereas the dashed lines mark the time bounds that give the maximal gain estimates. The solid lines show the running time; they touch the dotted lines where the methods hit the time bound. The successfully solved problems are marked by circles and the failures are shown by pluses.

APPLY's total gain is 360.3, which makes an average of 12.0 per problem. If we used the maximal-gain time bound, 11.6, for all problems, the gain would be 14.0 per problem. Thus, the use of incremental learning yielded a near-maximal gain, in spite of the initial ignorance. APPLY's estimate of the maximal-gain bound, after solving all problems, is 9.6. It differs from the 11.6 bound, found from Table 1, because the use of bounds that ensure a near-maximal gain prevents sufficient exploration.

DELAY's total gain is 115.7, or 3.9 per problem. If we used the data in Table 1 to find the optimal bound, which is 6.2, and solved all problems with this bound, we would earn 5.7 per problem. Thus, the incremental-learning gain is about two-thirds of the gain that could be obtained based on the advance knowledge. Finally, ALPINE's total gain is 339.7, or 11.3 per problem. The estimate based on Table 1 gives the bound 11.0, which would result in earning 12.3 per problem. Unlike APPLY, both DELAY and ALPINE eventually found the optimal bound.

Note that the choice of a time bound has converged to the optimal in two out of three experiments. Additional tests have shown that the insufficient exploration prevents finding the optimal bound in about half of all cases (Fink 1997). We tried to encourage more exploration by increasing the bound on $\frac{g_{\max}-g}{\sqrt{\sigma_{\max}^2+\sigma^2}}$ from 0.1 to 0.2. The selected bound then converges to the optimal more often; however, the overall performance worsens due to larger time losses on unsolved problems.

We next describe the use of incremental learning to select a problem-solving method. If we have no past data for some

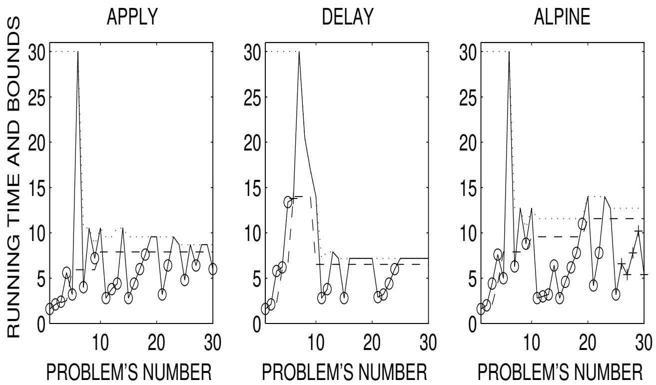

Figure 2: Results of the incremental learning of a time bound: running times (solid lines), time bounds (dotted lines), and maximal-gain bounds (dashed lines). The successes are marked by circles (o) and the failures by pluses (+).

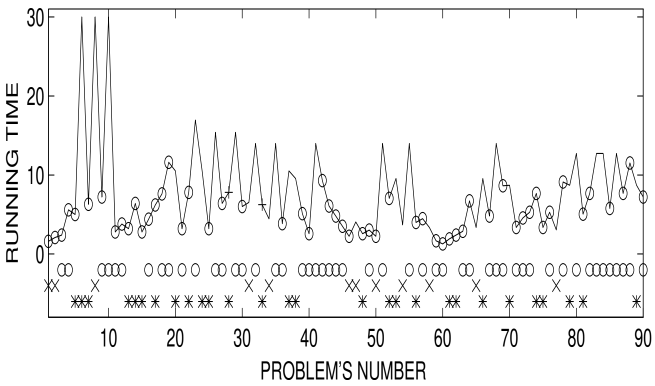

Figure 3: Results of the incremental selection of a method and time bound, on ninety transportation problems. The graph shows the running times (solid line), successes (o), and failures (+). The three rows of symbols below the solid line show the selection made among APPLY (o), DELAY (x), and ALPINE ($*$).

method, we select this unknown method, thus encouraging exploration.

If we have past data for all methods, we first select a time bound for each method. Then, for each method and its selected bound, we find the probability that it is the best among the methods. We use the statistical t-test to estimate the probability that some method is better than another one. We compute the probability that a method is best as the product of the probabilities that it outperforms individual methods. This computation is an approximation, since the probabilities that we multiply are not independent.

Finally, we make a weighted random selection among the methods; the chance of selecting a method is equal to the probability that it is best among the methods. This strategy leads to the frequent application of methods that perform well, but also encourages some exploratory use of poor performers.

We show the results of using this selection strategy in the transportation domain, for the reward of 30.0, in Figure 3. In this experiment, we first use the thirty problems from Table 1 and then sixty additional transportation problems. The horizontal axis shows the problem's number, whereas the vertical axis is the running time. The rows of symbols below the curve show the selection of a planner: a circle for APPLY, a cross for DELAY, and an asterisk for ALPINE.

The total gain is 998.3, or 11.1 per problem. The selection converges to the use of APPLY with the time bound 12.7, which is optimal for this set of ninety problems. If we used this selection on all the problems, we would earn 13.3 per problem.

5 Use of Problem Sizes

We have considered the task of finding a problem-solving method and time bound that will work well for most problems in a domain. If we can estimate the sizes of problems, we improve the performance by adjusting the time bound to a

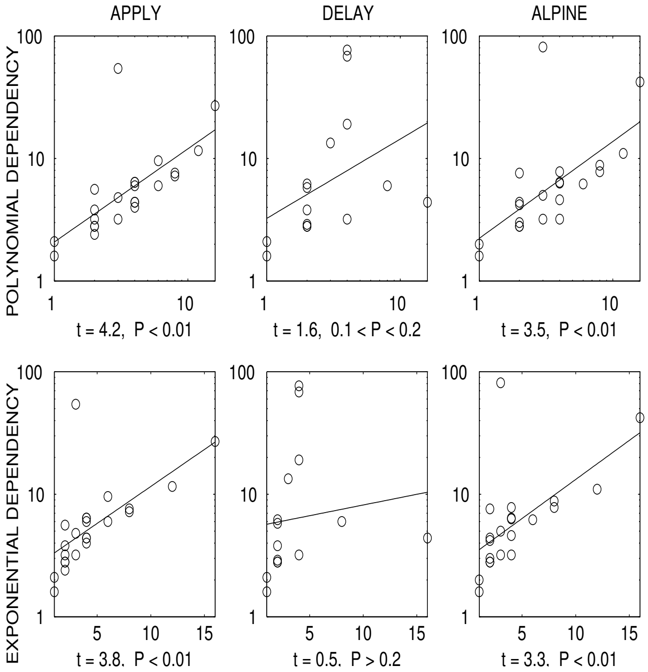

Figure 4: The dependency of the success time on the problem size. The top graphs show the regression for a polynomial dependency, and the bottom graphs are for an exponential dependency.

problem size.

We define a *problem size* as an easily computable positive value that correlates with the problem complexity: the larger the value, the longer it usually takes to solve the problem. Finding an accurate measure of complexity is often a difficult task, but many domains provide at least a rough complexity estimate. In the transportation domain, we estimate the complexity by the number of packages to be delivered. In the rightmost column of Table 1, we show the number of packages in each problem.

We use regression to find the dependency between the sizes of the sample problems and the times to solve them. We use separate regressions for success times and for failure times. We assume that the dependency of time on size is either polynomial or exponential. If it is polynomial, the logarithm of time depends linearly on the logarithm of size; for an exponential dependency, the time logarithm depends linearly on size. We thus use linear least-square regression to find both polynomial and exponential dependencies.

In Figure 4, we give the results of regressing the success times for the transportation problems from Table 1. The top three graphs show the polynomial dependency of the success time on the problem size, whereas the bottom graphs are for the exponential dependency. The horizontal axes show the problem sizes (that is, the number of packages), and the vertical axes are the running time. The circles show the sizes and times of the problem instances; the solid lines are the regression results.

We evaluate the regression using the t-test. The t value is the ratio of the estimated slope of the regression line to the standard deviation of the slope estimate. The t-test converts the t value into the probability that using the regression is *no better* than ignoring the sizes and simply taking the mean time; this probability is called the P *value*. When the regression gives a good fit to sample data, t is large and P is small. In Figure 4, we give the t values and the corresponding intervals of the P value.

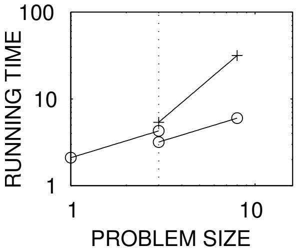

Figure 5: Scaling two success times (o) and a failure time (+) of DELAY to a 3-package problem.

We use the regression only if the probability P is smaller than a certain bound. In our experiments, we set this bound to 0.2; that is, we use problem sizes only for $P < 0.2$. We use the 0.2 value rather than more "customary" 0.05 or 0.02 because an early detection of a dependency between sizes and times is more important for the overall efficiency than establishing a high certainty of the dependency. We select between the polynomial and exponential regression based on the value of t: we prefer the regression with the larger t. In our example, the polynomial regression wins for all three methods.

Note that least-square regression and the related t-test make quite strong assumptions about the nature of the distribution. First, for problems of fixed size, the distribution of the time logarithms must be normal; that is, time must be distributed log-normally. Second, for all problem sizes, the standard deviation of the distribution must be the same. The regression, however, usually provides a good approximation of the dependency between size and time even when these assumptions are not satisfied.

The use of a problem size in estimating the gain is based on "scaling" the times of sample problems to the given size. We illustrate the use of regression in Figure 5, where we scale DELAY's times of a 1-package success, an 8-package success, and an 8-package failure for estimating the gain on a 3-package problem (the 3-package size is marked by the vertical dotted line). We use the slope of the success regression in scaling success times, and the slope of the failure regression in scaling failures.

The slope of scaling an interrupt should depend on whether the algorithm would succeed or fail if we did not interrupt it; however, we do not know which outcome would occur. We use a simple heuristic of choosing between the success and failure slope based on which of them has smaller P.

For a sample of N problems, the overall time of computing the polynomial and exponential regression slope, selecting between the two regressions, and scaling the sample times is $N \cdot 9 \cdot 10^{-4}$ seconds. For the incremental learning of a

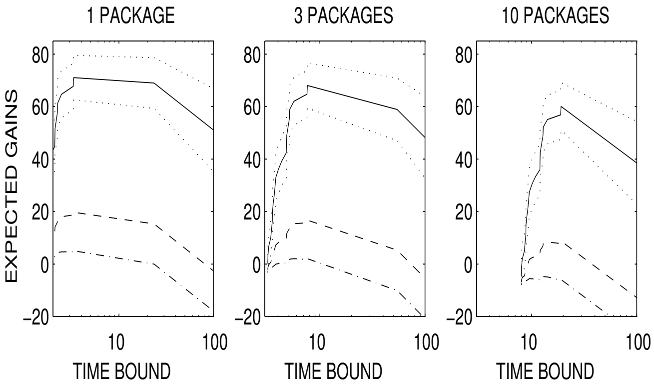

Figure 6: Dependency of APPLY's expected gain on the time bound, for rewards of 10.0 (dash-and-dot lines), 30.0 (dashed lines), and 100.0 (solid lines). The dotted lines show the standard deviation of the gain estimate for the 100.0 reward.

time bound, we implemented a procedure that incrementally updates the slope and t value after adding a new problem to the sample. This procedure reduces the amortized time of the statistical computation to $(N \cdot 2 + 7) \cdot 10^{-4}$ seconds per problem.

After scaling the sample times to a given size, we use the technique of Section 3 to compute the gain estimate and its standard deviation. The only difference is that we reduce the second term in the denominator for the deviation (Formula 4) by 2, as the success and failure regressions reduce the number of degrees of freedom of the sample. Thus, we compute the deviation as follows: $\sqrt{\frac{SqrSum - Sum^2/(n+m+k)}{(n+m+k)\cdot(n+m+k-e-3)}}$.

In Figure 6, we show the dependency of the expected gain on the time bound when using APPLY on 1-package, 3-package, and 10-package problems. If we use the problem sizes in the incremental-selection experiments of Section 3, we gain 12.2 per problem in learning a bound for APPLY (vs. the 12.0 gain obtained without the use of sizes), 4.7 in learning for DELAY (vs. 3.9 without the use of sizes), 11.9 in learning for ALPINE (vs. 11.3), and 11.8 in the method-selection experiment (vs. 11.1). The average running time of regression and scaling is 0.03 seconds per problem. Thus, the time of the statistical computation for using problem sizes is much smaller than the resulting gain increase.

The results show that the use of problem sizes increases the gain, though not by much. We have conducted additional experiments with artificially generated data (Fink 1997) and demonstrated that, if running times better correlate with problem sizes, the gain increase becomes more significant.

6 Empirical Examples

We have demonstrated the effectiveness of the statistical selection in a simple transportation domain. We now give results in two other domains.

We first consider an extension to the transportation domain, in which we use airplanes to carry packages between cities and vans for the local delivery within cities (Veloso 1994). In Table 3, we give the performance of APPLY, DELAY, and ALPINE on thirty problems.

#	time (sec) and outcome			# of
	APPLY	DELAY	ALPINE	packs
1	4.7 s	4.7 s	4.7 s	1
2	96.0 s	9.6 f	7.6 f	2
3	5.2 s	5.1 s	5.2 s	1
4	20.8 s	10.6 f	14.1 s	2
5	154.3 s	31.4 s	7.5 f	2
6	2.5 s	2.5 s	2.5 s	1
7	4.0 s	2.9 s	3.0 s	1
8	18.0 s	19.8 s	4.2 s	2
9	19.5 s	26.8 s	4.8 s	2
10	123.8 s	500.0 b	85.9 s	3
11	238.9 s	96.8 s	76.6 s	3
12	500.0 b	500.0 b	7.6 f	4
13	345.9 s	500.0 b	58.4 s	4
14	458.9 s	98.4 s	114.4 s	8
15	500.0 b	500.0 b	115.6 s	8
16	35.1 s	21.1 s	6.6 f	2
17	60.5 s	75.0 f	13.7 s	2
18	3.5 s	3.4 s	3.5 s	1
19	4.0 s	3.8 s	4.0 s	1
20	232.1 s	97.0 s	9.5 f	2
21	60.1 s	73.9 s	14.6 s	2
22	500.0 b	500.0 b	12.7 f	2
23	53.1 s	74.8 s	15.6 s	2
24	500.0 b	500.0 b	38.0 s	4
25	500.0 b	213.5 s	99.2 s	4
26	327.6 s	179.0 s	121.4 s	6
27	97.0 s	54.9 s	12.8 s	6
28	500.0 b	500.0 b	16.4 f	8
29	500.0 b	500.0 b	430.8 s	16
30	500.0 b	398.7 s	214.8 s	8

Table 3: Performance in the extended transportation domain.

We present the results of the incremental learning of a time bound for APPLY, DELAY, and ALPINE, for the reward of 400.0, in the top three graphs of Figure 7 (the legend is the same as in Figure 2). We obtained these results without the use of problem sizes. The APPLY learning gives the gain of 110.1 per problem and eventually selects the bound 127.5. The optimal bound for this set of problems is 97.0. If we used the optimal bound for all problems, we would earn 135.4 per problem. If we estimate the problem sizes by the number of packages to be delivered, and use these sizes in learning the time bound for APPLY, we gain 121.6 per problem.

DELAY gains 131.1 per problem and chooses the 105.3 bound at the end of the learning. The actual optimal bound for DELAY is 98.4, the use of which on all problems would give 153.5 per problem. The use of the problem size in the incremental learning gives 137.4 per problem.

Finally, ALPINE gains 243.5 per problem and chooses the bound 127.6. The optimal bound for ALPINE is 430.8, the use of which would give the per-problem gain of 255.8. The use of problem sizes gives the 248.3 per-problem gain. (ALPINE outperforms APPLY and DELAY because it uses abstraction, which separates the problem of between-city air transporta-

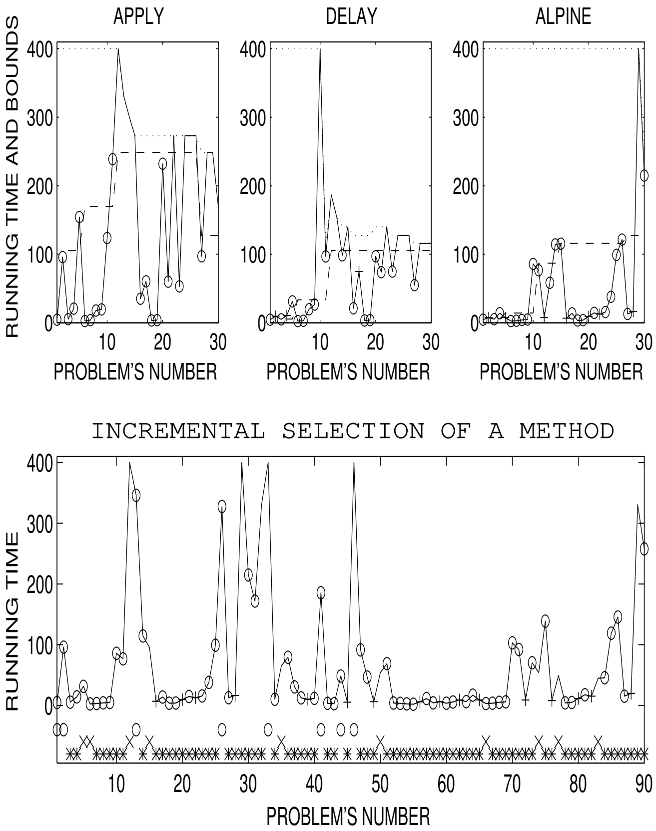

Figure 7: Incremental learning of time bounds (top three graphs) and selection of a method (bottom graph) in the extended transportation domain.

tion from the problem of within-city deliveries.)

The bottom graph in Figure 7 shows the results of the incremental selection of a method, without the use of problem sizes (we use the same legend as in Figure 3). We first use the thirty problems from Table 3 and then sixty other problems. The method converges to the choice of ALPINE with the time bound 300.6 and gives the gain of 207.0 per problem. The optimal choice for this set of problems is the use of ALPINE with the time bound 517.1, which would yield 255.6 per problem. An incremental-selection experiment with the use of problem sizes gives the 229.4 per-problem gain.

We next apply our technique to the bound selection when calling to a friend on the phone. We determine how many seconds (or rings) you should wait for an answer before hanging up. The reward for reaching your party may be determined by the time that you are willing to wait in order to talk now, as opposed to hanging up and calling later. In Table 4, we give the times for sixty calls, rounded to 0.05 seconds[1]. A success occurred when our party answered the phone. A reply by an answering machine was considered a failure.

The graph on the left of Figure 8 shows the dependency of the expected gain on the time bound, for the rewards of

#	time	#	time	#	time
1	5.80 f	21	200.00 b	41	12.60 s
2	8.25 s	22	200.00 b	42	26.15 f
3	200.00 b	23	10.50 s	43	7.20 s
4	5.15 s	24	14.45 f	44	16.20 f
5	8.30 s	25	11.30 f	45	8.90 s
6	200.00 b	26	10.20 f	46	4.25 s
7	9.15 s	27	4.15 s	47	7.30 s
8	6.10 f	28	14.70 s	48	10.95 s
9	14.15 f	29	2.50 s	49	10.05 s
10	200.00 b	30	8.70 s	50	6.50 s
11	9.75 s	31	6.45 s	51	15.10 f
12	3.90 s	32	6.80 s	52	25.45 s
13	11.45 f	33	8.10 s	53	20.00 f
14	3.70 s	34	13.40 s	54	24.20 f
15	7.25 s	35	5.40 s	55	20.15 f
16	4.10 s	36	2.20 s	56	10.90 s
17	8.25 s	37	26.70 f	57	23.25 f
18	5.40 s	38	6.20 s	58	4.40 s
19	4.50 s	39	24.45 f	59	3.20 f
20	32.85 f	40	29.30 f	60	200.00 b

Table 4: Waiting times (seconds) in phone-call experiments.

30.0 (dash-and-dot line), 90.0 (dashed line), and 300.0 (solid line). The optimal bound for the 30.0 and 90.0 rewards is 14.7 (three rings), whereas the optimal bound for 300.0 is 25.5 (five rings).

The graph on the right of Figure 8 shows the results of selecting the bounds incrementally, for the reward of 90.0. The learned bound converges to the optimal bound, 14.7. The average gain obtained during the learning is 38.9 per call. If we used the optimal bound for all calls, we would earn 41.0 per call.

The experiments in the two PRODIGY domains and the phone-call domain show that the learning procedure usually finds an appropriate bound and yields a near-maximal gain. In the full paper (Fink 1997), we present a series of experiments with artificially generated time values, using normal, uniform, log-normal, and log-uniform distributions. The learning gives good results for all four distributions.

7 Conclusions and Open Problems

We have stated the task of selecting among problem-solving methods as a statistical problem, derived an approximate solution, and demonstrated experimentally its effectiveness in selecting an appropriate method and time bound. In the full paper (Fink 1997), we describe the use of similarity among problems to improve the accuracy of method selection. We have designed a module that uses similarity to choose relevant data from the library of past problem-solving episodes, which enables the selection algorithm to adjust the method and bound selection to specific features of a given problem.

Even though AI planning provided the motivation for our work, the generality of the statistical model makes it applicable to a wide range of real-life situations outside AI. The main limitation of applicability stems from the restrictions on the reward function. We plan to test the effectiveness

[1]We made these calls to sixty different people at their home numbers; we measured the time from the beginning of the first ring, skipping the static silence of the connection delays.

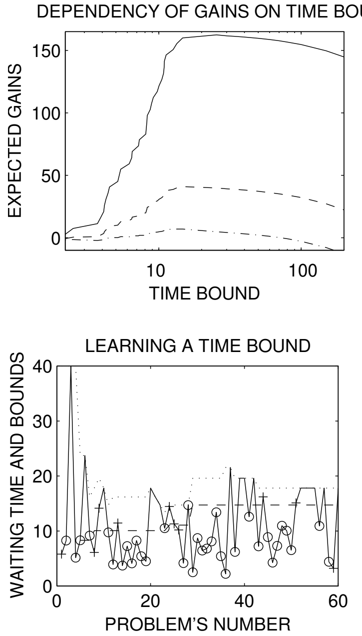

Figure 8: Dependency of the expected gain on the time bound in the phone-call domain, for the rewards of 30.0 (dash-and-dot line), 90.0 (dashed line), and 300.0 (solid line), and the results of the incremental learning of a time bound.

of the statistical-selection algorithm in other AI systems, as well as on method-selection tasks outside AI.

To make the model more flexible, we need to provide a mechanism for switching the method and revising the time bound in the process of search for a solution. We also plan to study the possibility of running competing problem-solving methods on parallel machines, which will require a further extension to the statistical model. Another open problem is to consider possible dependencies of the reward on the solution quality and enhance the model to account for such dependencies. Finally, we need to allow interleaving of several promising methods, and selecting new methods if the original selection has failed. Such a multi-method strategy is often more effective than sticking to one method.

Acknowledgements

I am grateful to Svetlana Vayner, who contributed many valuable insights. She helped to construct the statistical model for estimating the performance of problem-solving methods and provided a thorough feedback on all my ideas. I also owe thanks to Manuela Veloso, Martha Pollack, Henry Rowley, and anonymous reviewers for their valuable comments.

The research is sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and Defense Advanced Research Project Agency (DARPA) under grant number F33615-93-1-1330.

References

Bacchus, F., and Yang, Q. 1992. The expected value of hierarchical problem-solving. In *Proceedings of the Tenth National Conference on Artificial Intelligence*.

Breese, J. S., and Horvitz, E. J. 1990. Ideal reformulation of belief networks. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, 64–72.

Fink, E. 1997. Statistical selection among problem-solving methods. Technical Report CMU-CS-97-101, Department of Computer Science, Carnegie Mellon University.

Hansen, E. A., and Zilberstein, S. 1996. Monitoring the progress of anytime problem-solving. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1229–1234.

Hansson, O., and Mayer, A. 1989. Heuristic search and evidential reasoning. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, 152–161.

Horvitz, E. J. 1988. Reasoning under varying and uncertain resource constraints. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 111–116.

Knoblock, C. A. 1991. *Automatically Generating Abstractions for Problem Solving*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University. Technical Report CMU-CS-91-120.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68:243–302.

Minton, S. 1996. Automatically configuring constraint satisfaction programs: A case study. *Constraints: An International Journal* 1:7–43.

Mouaddib, A., and Zilberstein, S. 1995. Knowledge-based anytime computation. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 775–781.

Newell, A., and Simon, H. A. 1972. *Human Problem Solving*. Englewood Cliffs, NJ: Prentice Hall.

Pérez, M. A. 1995. *Learning Search Control Knowledge to Improve Plan Quality*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University. Technical Report CMU-CS-95-175.

Polya, G. 1957. *How to Solve It*. Garden City, NY: Doubleday, second edition.

Russell, S. J.; Subramanian, D.; and Parr, R. 1993. Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 338–344.

Russell, S. J. 1990. Fine-grained decision-theoretic search control. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, 436–442.

Stone, P.; Veloso, M. M.; and Blythe, J. 1994. The need for different domain-independent heuristics. In *Proceedings of the Second International Conference on AI Planning Systems*, 164–169.

Valiant, L. G. 1984. A theory of the learnable. *Communications of the ACM* 27:1134–1142.

Veloso, M. M., and Stone, P. 1995. FLECS: Planning with a flexible commitment strategy. *Journal of Artificial Intelligence Research* 3:25–52.

Veloso, M. M. 1994. *Planning and Learning by Analogical Reasoning*. Springer Verlag.