Automatic Evaluation and Selection of Problem-Solving Methods: Theory and Experiments

Eugene Fink

School of Computer Science, Carnegie Mellon University

5000 Forbes Avenue, Pittsburgh, PA 15213

e.fink@cs.cmu.edu, www.cs.cmu.edu/∼eugene

Abstract

The choice of the right problem-solving method, from available methods, is a crucial skill for experts in many areas. We present a technique for automatic selection among methods based on analysis of their past performances. We formalize the statistical problem involved in choosing an efficient method, derive a solution to this problem, and describe a selection algorithm. The algorithm not only chooses among available methods, but also decides when to abandon the chosen method if it takes too much time. We then extend the basic statistical technique to account for problem sizes and similarity among problems.

Keywords: Learning, problem solving, statistical analysis.

1 Introduction

The choice of the right problem-solving method is one of the main themes of Polya's [1957] famous book *How to Solve It*. Polya showed that selection of an effective approach to a problem is a crucial skill for a mathematician. Psychologists have accumulated much evidence that confirms Polya's insight: the performance of experts depends on their ability to find the right approach to a given task [Newell and Simon, 1972; Gentner and Stevens, 1983; Simon, 1989; Tabachneck-Schijf *et al.*, 1997].

The purpose of our research is to automate selection of a problem-solving method; it is motivated by work on the PRODIGY system, which includes several search engines [Veloso and Stone, 1995; Veloso *et al.*, 1995]. We need to provide a mechanism for deciding *which search engine is appropriate* for a given problem. Furthermore, since programs in the real world cannot run forever, we need to decide *when to interrupt an unsuccessful search.*

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

Russell *et al.* [1993] formalized a general problem of selecting among alternative problem-solving methods and used dynamic programming to solve some special cases. Minton developed an inductive learning system that configured constraint-satisfaction programs, by selecting among alternative search strategies [Minton, 1996; Allen and Minton, 1996].

Hansen and Zilberstein [1996] 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. Mouaddib and Zilberstein [1995] developed a similar technique for hierarchical knowledge-based algorithms.

Howe *et al.* [1999] built the meta-planner system, which integrated six planners and chose among them based on features of a given problem. It used linear regression to compute the expected running time of each planner; however, the regression was based on several features of classical planning, which may not generalize to other tasks.

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

We have therefore developed a novel selection technique, which makes the best use of the available data even when they do not allow an accurate estimate, and combines exploitation of past data with exploration of new alternatives. We have also considered the task of setting a time bound for the chosen method. The previous techniques for interrupting any-time algorithms are not applicable, because PRODIGY does not use any-time behavior and does not satisfy the assumptions of the past studies. We describe a statistical technique for selecting time bounds, and demonstrate that setting an appropriate bound is as crucial as choosing the right method. Although selection among PRODIGY engines has provided a motivation for this work, the *developed technique does not rely on specific properties of* PRODIGY, and it can choose among multiple problem-solving methods in any artificial-intelligence system.

We formalize the problem of estimating the expected performance of a method (Section 2), derive a solution to this problem (Section 3), and use it in selecting a method and time bound (Section 4). Note that we do not need a perfect estimate of the expected performance; we only need accuracy sufficient for selecting the right method and a close-to-optimal time bound. We give results of using the developed technique to select among PRODIGY search engines (Section 5). Then, we describe the use of a heuristic measure of problem complexity (Section 6) and similarity among problems (Section 7) to improve performance estimates. Finally, we test the technique on artificially generated performance data (Section 8).

2 Motivating example

We give an example of a method-selection task in PRODIGY and use it to formalize the statistical selection problem. We use PRODIGY to construct plans for transporting packages between different locations in a city [Veloso, 1994], and consider three alternative search methods.

The first of them is based on several control rules, designed by Veloso [1994] and Pérez [1995] to guide PRODIGY in the transportation domain. This method applies the selected actions as early as possible; we call it APPLY. The second method uses the same control rules, along with a special rule that delays the application and forces more emphasis on backward search

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

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

[Veloso and Stone, 1995]; we call it DELAY. The third method, ALPINE [Knoblock, 1994], is a combination of APPLY with an abstraction generator, which determines relative importance of domain elements. 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 shown that delaying the application improves efficiency in some domains, but slows PRODIGY down in others [Stone *et al.*, 1994]; abstraction sometimes saves time and sometimes worsens the performance [Knoblock, 1993; 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 these data are only for illustrating the selection problem, and not for a general comparison of these search techniques; their relative performance may be different in other domains. Also note that the selection technique does not rely on specific properties of PRODIGY; it is applicable to selection among multiple methods in any artificial-intelligence system.

Although each method outperforms the others on at least one problem (Table 1), a glance at the data reveals that APPLY's performance in this domain is probably the best among the three. We use statistical analysis to confirm this intuitive conclusion, and show how to choose a time bound for the selected method.

We may evaluate the performance of a method along several dimensions, such as the percentage of solved problems, average success time, and average failure time. To compare different methods, we need a utility function that unifies these dimensions. We assume that we are paying for running time and getting a certain reward R for solving a problem. If the method solves the problem, the overall gain is $(R - \mathit{time})$; in particular, if $R < \mathit{time}$, the "gain" is negative and we are better off avoiding the problem. If the method fails or hits the time bound, the

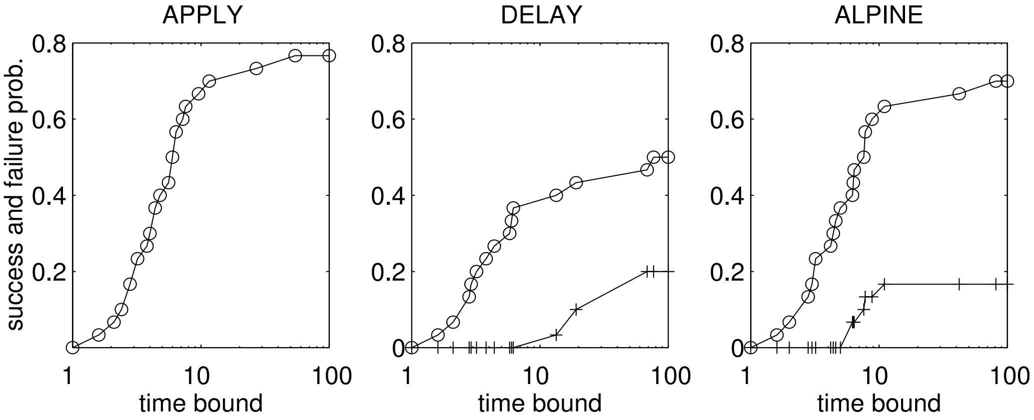


Figure 1: Dependency of the success (o) and failure (+) probabilities on the time bound.

"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 leads to the following problem.

Problem: *Suppose that a method has solved n problems, has failed on m problems, and has been interrupted upon hitting a time bound on k problems. The success times were $s_1, s_2, ..., s_n$, 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 and determine the standard deviation of the estimate.*

We use the *stationarity assumption* [Valiant, 1984]; that is, we assume that the past problems and the new problem are drawn randomly from the same population, using the same probability distribution, and that the methods' performance does not change over time.

3 Statistical foundations

We derive a solution to the statistical problem; for convenience, assume 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 probability of success by the fraction of problems solved within time B, which is $\frac{c}{n+m+k}$; similarly, the estimated probability of failure is $\frac{d}{n+m+k}$. For example, the chance that ALPINE with time bound 6.0 solves a transportation problem is $\frac{11}{30} = 0.37$, and the probability that it terminates with failure is $\frac{2}{30} = 0.07$.

In Figure 1, we show the dependency between the time bound, given in the horizontal axes using a logarithmic scale, and the estimated success and failure probabilities, in the transportation domain. The graphs do not include a failure estimate for APPLY, because its data contain no failures. We have computed the probabilities only for the points marked by circles and pluses, and connected them by straight segments.

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

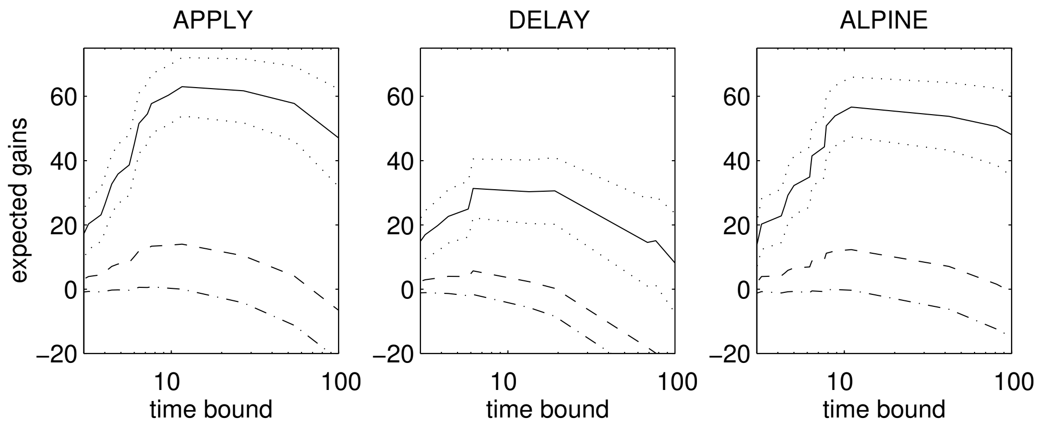


Figure 2: Dependency of the expected gain on the time bound, for the reward of 10.0 (dash-and-dot lines), 30.0 (dashed lines), and 100.0 (solid lines). The dotted lines show the standard deviation of the gain estimate for the 100.0 reward.

gains $R-s_1, R-s_2, ..., R-s_c$, and fail 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 the mean of all these $n+m+k$ gains:

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

For instance, if we use ALPINE with reward 30.0 and time bound 6.0, the expected gain is 6.0.

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

$$\sqrt{\frac{SqrSum - \frac{Sum^2}{n+m+k}}{(n+m+k)\cdot(n+m+k-1)}}, \tag{2}$$

where

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

This expression is an approximation based on the *Central Limit Theorem*, which states that the distribution of sample means is close to the normal; for example, see the textbook by Mendenhall *et al.* [1999]. The approximation accuracy improves with sample size; for thirty or more sample problems, it is near-perfect. For instance, if we use ALPINE with reward 30.0 and time bound 6.0, the standard deviation of the expected gain is 2.9.

In Figure 2, we show the dependency of the expected gain on the time bound, for three different values of the reward R: 10.0, 30.0, and 100.0. The dotted lines show the standard deviation of the gain estimate for the 100.0 reward; the lower line is "one deviation below" the estimate, and the upper line is "one deviation above."

We have so far assumed that $B \leq b_1$. 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 have interrupted

#	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

(a)

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

(b)

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

(c)

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

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 we need to estimate the gain for $B = 6.0$. We cannot use $b_1, b_2, ..., b_e$ directly in the gain estimate since 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 gain using the data on the past problem-solving episodes in which the method has run beyond b_1. We thus remove b_1 from the sample and distribute its chance to occur among the higher-time outcomes. In Table 2(a), b_1 is 4.5, and there are 21 problems with larger times; thus, we remove 4.5 from the sample and increase the weights of the larger-time outcomes from 1 to $1 + \frac{1}{21} = 1.048$ (Table 2b). We next distribute the weight of b_2 among the larger-than-b_2 times. In the 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, denoted by w. The sum of the weights is equal to the number of problems in the original sample; that is, $\sum_{i=1}^{c} u_i + \sum_{j=1}^{d} v_j + (n + m + k - c - d - e) \cdot w = n + m + k$.

We have thus obtained $n+m+k-e$ weighted times, and we use them to estimate the success and failure probabilities. The probability of solving a problem, within the time bound B, is $\frac{u_1+u_2+...+u_c}{n+m+k}$; similarly, the failure probability is $\frac{v_1+v_2+...+v_d}{n+m+k}$. If we use Table 2(c) to determine these probabilities for ALPINE with time bound 6.0, then the success chance is $\frac{11.096}{30} = 0.37$, and the failure chance is $\frac{2.096}{30} = 0.07$.

We next use the weighted times to compute the expected gain:

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

c	number of the already processed success times; the next success time to process will be s_{c+1}
d	number of the already processed failure times
e	number of the already processed interrupt times
h	number of the already processed time bounds
w	weight of the success, failure, and interrupt times larger than the time bound B_{h+1}
S_Num	sum of the weights of the processed successes, $\sum_{i=1}^{c} u_i$
F_Num	sum of the weights of the processed failures, $\sum_{j=1}^{d} v_j$
S_Sum	weighted sum of the gains for the processed successes, $\sum_{i=1}^{c} u_i \cdot (R - s_i)$
F_Sum	weighted sum of the gains for the processed failures, $-\sum_{j=1}^{d} v_j \cdot f_j$
Sum	weighted sum of the gains for all sample problems, for the current time bound B_{h+1}
S_SqrSum	weighted sum of the squared gains for the processed successes, $\sum_{i=1}^{c} u_i \cdot (R - s_i)^2$
F_SqrSum	weighted sum of the squared gains for the processed failures, $\sum_{j=1}^{d} v_j \cdot f_j^2$
SqrSum	weighted sum of the squared gains for all sample problems, for the time bound B_{h+1}

Figure 3: Variables used in the gain-estimate algorithm in Figure 4.

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

$$\sqrt{\frac{SqrSum - \frac{Sum^2}{n+m+k}}{(n+m+k) \cdot (n+m+k-e-1)}}, \tag{4}$$

where

$$Sum = \textstyle\sum_{i=1}^{c} u_i \cdot (R - s_i) - \sum_{j=1}^{d} v_j \cdot f_j - (n+m+k-c-d-e) \cdot w \cdot B,$$
$$SqrSum = \textstyle\sum_{i=1}^{c} u_i \cdot (R - s_i)^2 + \sum_{j=1}^{d} v_j \cdot f_j^2 + (n+m+k-c-d-e) \cdot w \cdot B^2.$$

The application of these expressions to the data in Table 2(c), 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.

If B is larger than the largest of the past bounds (that is, $B > b_k$), and the largest past bound is larger than all past success and failure times (that is, $b_k > s_n$ and $b_k > f_m$), then the re-distribution procedure does not work. We need to distribute the weight of b_k among the larger-time problems, but the sample has no such problems. In this case, the data are insufficient for the statistical analysis because we do not have past experience with large enough time bounds.

We have assumed in the derivation that the execution cost is proportional to the running time; however, we may extend the results to any other monotonic dependency between time and cost, by replacing the terms $(R - s_i)$, f_i, and B with more complex functions [Fink, 2003].

We now present an algorithm for computing the success and failure probabilities, gain estimates, and estimate deviations, for multiple values of the time bound B. We describe the variables used in the computation in Figure 3 and give pseudocode in Figure 4. The algorithm computes the weights and gain estimates in one pass through the sorted list of successes, failures, interrupts, and time-bound values. When processing a success or failure time, it increments the corresponding sums of the weighted gains and weighted squares of the gains. When processing an interrupt time, it modifies the weight value. When processing a time bound, it uses the accumulated sums to compute the gain estimate and standard deviation for this bound.

The input of the algorithm includes the reward R; the sorted list of success times, $s_1, ..., s_n$; the sorted list of failure times, $f_1, ..., f_m$; the sorted list of interrupt times, $b_1, ..., b_k$; and a sorted list of candidate time bounds, $B_1, ..., B_l$. The variables used in the computation are described in Figure 3.

Set the initial values:

$c := 0$; $d := 0$; $e := 0$; $h := 0$
$w := 1$; $S\_Num := 0$; $F\_Num := 0$
$S\_Sum := 0$; $F\_Sum := 0$
$S\_SqrSum := 0$; $F\_SqrSum := 0$

Repeat the computation until finding the gains for all time bounds, that is, until $h = l$:

- Select the smallest among the following four times: s_{c+1}, f_{d+1}, b_{e+1}, and B_{h+1}
- If the success time s_{c+1} is selected, increment the related sums:
  $S\_Num := S\_Num + w$
  $S\_Sum := S\_Sum + w \cdot (R - s_{c+1})$
  $S\_SqrSum := S\_SqrSum + w \cdot (R - s_{c+1})^2$
  $c := c + 1$
- If the failure time f_{d+1} is selected, increment the related sums:
  $F\_Num := F\_Num + w$
  $F\_Sum := F\_Sum - w \cdot f_{d+1}$
  $F\_SqrSum := F\_SqrSum + w \cdot f_{d+1}^2$
  $d := d + 1$
- If the interrupt time b_{e+1} is selected:
  If no success or failure times are left, that is, $c = n$ and $d = m$,
  then terminate (the data are insufficient for estimating the gains for the remaining bounds)
  Else, distribute the interrupt's weight among the remaining times, by incrementing w and e:
  $w := w \cdot \frac{n+m+k-c-d-e}{n+m+k-c-d-e-1}$
  $e := e + 1$
- If the time bound B_{h+1} is selected:
  First, compute the sum of the sample-problem gains and the sum of their squares:
  $Sum := S\_Sum + F\_Sum - (n + m + k - c - d - e) \cdot w \cdot B_{h+1}$
  $SqrSum := S\_SqrSum + F\_SqrSum + (n + m + k - c - d - e) \cdot w \cdot B_{h+1}^2$
  Next, compute the success and failure probability, gain estimate, and deviation, for B_{h+1}:
  Success probability: $\frac{S\_Num}{n+m+k}$ Gain estimate: $\frac{Sum}{n+m+k}$
  Failure probability: $\frac{F\_Num}{n+m+k}$ Estimate deviation: $\sqrt{\frac{SqrSum - Sum^2/(n+m+k)}{(n+m+k)\cdot(n+m+k-e-1)}}$
  Finally, increment the number of processed bounds:
  $h := h + 1$

Figure 4: Computing the success and failure probabilities, gain estimates, and estimate deviations.

The overall time of this computation is linear; that is, for l time bounds and a sample of n successes, m failures, and k interrupts, the algorithm's complexity is $O(l + n + m + k)$. The complexity of pre-sorting the sample is $O((l + n + m + k) \cdot \lg(l + n + m + k))$, but in practice it takes much less time than the rest of the computation. We have implemented the algorithm in Common Lisp and tested it on a Sun Sparc 5, which is the same computer as we have used for solving the transportation problems; the running time is about $(l + n + m + k) \cdot 3 \cdot 10^{-4}$ seconds.

4 Selection of a method and time bound

We use the statistical estimate in selecting a problem-solving method and time bound, and provide heuristics for combining exploitation of past experience with exploration of new alternatives. The basic idea is to estimate the gain for a number of time bounds, for each available method, and select the method and 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 2. If the expected gain for all time bounds is negative, then we should not solve the problem at all. For example, if the only available method is DELAY, and the reward is 10.0, then we should skip the problem.

For each method, we use its past success times as candidate bounds, and compute the expected gain only for these bounds. If we used 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 bound from s_i to B would not increase the number of successes on the past problems. In practice, we multiply the success times by 1.001 to obtain candidate bounds, in order to avoid rounding errors. If several candidate bounds are "too close" to each other, we drop some of them, to reduce the amount of computation. In the experiments, we have considered bounds too close if they are within the factor of 1.01 from each other.

We now describe a technique for incremental learning of the performance of available methods. The system may begin with no past experience and accumulate performance data as it solves more problems. For each new problem, it uses the statistical analysis to select a method and time bound; after applying the selected method, it adds the result to the performance data. The incremental learning causes a deviation from rigorous statistics: the resulting success, failure, and interrupt times are not independent because the time bound used for each problem depends on the times of solving the previous problems; however, it gives good results in practice.

Note that the system needs to choose a method and time bound even if it has no past experience. Also, it sometimes needs to deviate from the maximal-expectation selection in order to explore new opportunities. If it always used the selection that maximized the expected gain, it would be stuck with the method that yielded the first success, and it would never set a time bound higher than the first success time. We have not constructed a statistical model for combining exploitation and exploration; instead, we provide a heuristic solution, which gives good empirical results. We first consider the selection of a bound for a fixed method, and then show how to select a method.

Selecting a time bound

If we have no previous data on a method's performance, we set the time bound equal to the reward. This heuristic is based on the observation that, for most artificial-intelligence search algorithms, the probability of solving a problem within the next second usually declines with the passage of search time. For example, if an algorithm has not solved a problem within half a minute, chances are it will not find a solution in the next half minute. Thus, if the reward is 30.0 and the algorithm has already run for 30.0 seconds, it is time to interrupt the search.

Now suppose that we have accumulated some performance data, which allow finding 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. We 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 g and the maximal gain is $g_{\max} - g$. If the estimates are normally distributed, the standard deviation of the expected difference is $\sqrt{\sigma_{\max}^2 + \sigma^2}$; this expression 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 variables, 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 gives good experimental 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 strategy in Figure 5. We have run each of the three methods on the thirty transportation problems, in order. The horizontal axes show the problem's number (from 1 to 30), and 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 incremental learning yielded a near-maximal gain in spite of the initial ignorance. The time bounds (dotted line) converge to the estimated maximal-gain bounds (dashed line) since the deviations of the gain estimates decrease as the system solves more problems. 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 has prevented sufficient exploration.

DELAY's total gain is 115.7, which is 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 have found the optimal bound after solving all problems.

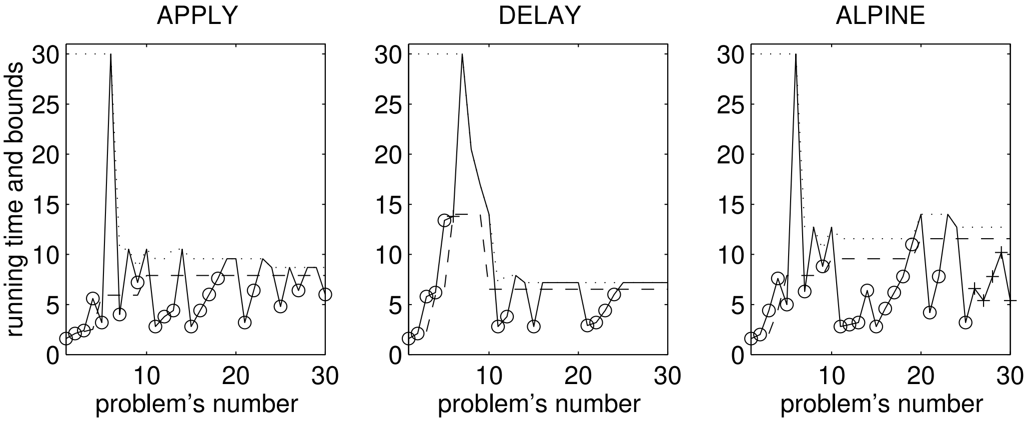


Figure 5: Incremental learning of a time bound: the running times (solid lines), time bounds (dotted lines), and maximal-gain bounds (dashed lines). The successes are marked by circles (o) and the failures by pluses (+).

The main "losses" in the incremental learning occur on the first ten problems, when the past data are insufficient for selecting an appropriate time bound. After this initial period, the choice of a bound becomes close to the optimal. The total time of the statistical computations while solving the thirty problems is 0.26 seconds, which is less than 0.01 per problem. It is negligible in comparison with the problem-solving time, which averages at 6.5 per problem for APPLY, 7.7 per problem for DELAY, and 7.1 per problem for ALPINE.

Selecting a method

We describe incremental selection of a problem-solving method. If we have no past data for some method, we choose this unknown method, thus encouraging exploration. If we have no data for several methods, we choose among them at random. If we have data for all methods, we first select a time bound for each method, and determine the gain estimates and their standard deviations for the selected bounds. Then, for each method and its selected bound, we find the probability that it is the best among the methods. Finally, we make a weighted random selection; the chance of choosing a method is equal to the probability that it is the best. This strategy leads to frequent use of methods that perform well, but it also encourages some exploratory use of poor performers.

We now explain a procedure for estimating the probability that a specific method is the best. Suppose that we have M different methods, and we select one of them, with gain estimate g and deviation σ. Recall that we compute g from the sample of past data, for the selected time bound. We denote the expected problem-solving gain, for the selected bound, by G; then, g is an unbiased statistical estimate of G. Finally, we denote the gain estimates of the other methods by $g_1, ..., g_{M-1}$ and the corresponding deviations by $\sigma_1, ..., \sigma_{M-1}$.

First, suppose that we know the exact value of G, that is, the mean gain for the population of all possible problems. The selected method is the best if G is larger than the expected gains of the other methods. We apply the statistical z-test to determine the probability that G is the largest among the expected gains.

We begin by finding the probability that G is greater than the expected gain of another method i, with gain estimate g_i and deviation σ_i. The expected difference between the two gains is $G - g_i$, and the standard deviation of the difference is σ_i. The z value is the ratio of the expected difference to its standard deviation; that is, $z = \frac{G-g_i}{\sigma_i}$. The z-test converts this value into the probability that the expected gain for the selected method is larger than that for method i; we denote the resulting probability by $p_i(G)$. Note that this z-test uses the value of G, which cannot be found from the sample data.

We next determine the probability $p(G)$ that G is larger than the expected gains of all other methods. If the gain estimates $g_1, ..., g_{M-1}$ are independent, the probabilities $p_1(G), ..., p_{M-1}(G)$ are also independent, and $p(G)$ is their product:

$$p(G) = \prod_{i=1}^{M-1} p_i(G).$$

Since we cannot compute G from the available data, we need to use its unbiased estimate g. The distribution of the possible values of G is approximately normal, with mean g and standard deviation σ, which means that its probability density function is as follows:

$$f(G) = \frac{e^{-(G-g)^2/(2\sigma^2)}}{\sigma \cdot \sqrt{2\pi}}.$$

To determine the probability p that the selected method is the best, we integrate over possible values of G:

$$p = \int_{-\infty}^{\infty} p(G) \cdot f(G)\, dG = \int_{-\infty}^{\infty} \prod_{i=1}^{M-1} p_i(G) \cdot \frac{e^{-(G-g)^2/(2\sigma^2)}}{\sigma \cdot \sqrt{2\pi}}\, dG. \qquad (5)$$

Note that we have made two simplifying assumptions. First, we have assumed that the sample means $g, g_1, ..., g_{M-1}$ are normally distributed; however, if we compute them from small samples, their distributions may not be normal. Second, we have viewed $g, g_1, ..., g_{M-1}$ as independent variables. If we use incremental learning, then the choice of a method depends on the previous choices, and the data collected for different methods are not independent.

Although the derived expression is an approximation, it works well for the learning algorithm. We use the probability p only for the "occasional exploration" heuristic, which does not require high accuracy in determining the exploration frequency. We have implemented the computation of p using numeric integration on the interval from $g - 4\sigma$ to $g + 4\sigma$, with step 0.1σ; that is, we approximate the integral by the following sum:

$$\sum_{j=-40}^{40} \prod_{i=1}^{M-1} p_i(g + 0.1j\sigma) \cdot \frac{e^{-(0.1j\sigma)^2/(2\sigma^2)}}{10 \cdot \sqrt{2\pi}}. \qquad (6)$$

For example, suppose that we are choosing among APPLY, DELAY, and ALPINE based on the data in Table 1. We select bound 13.1 for APPLY, which gives a gain estimate of 13.5 with deviation 3.3; bound 5.3 for DELAY, with a gain estimate of 5.3 and deviation 3.0; and bound 13.2 for ALPINE, with a gain of 11.2 and deviation 3.2. APPLY outperforms the other two methods with probability 0.68; the probability that DELAY is the best is 0.01; and ALPINE's chance of being the best is 0.31. We now choose one of the methods at random; the chance of choosing each method equals its probability of being the best.

In Figure 6, we present the results of this strategy for the reward of 30.0. In this experiment, we first use the thirty problems from Table 1 and then sixty additional transportation problems.

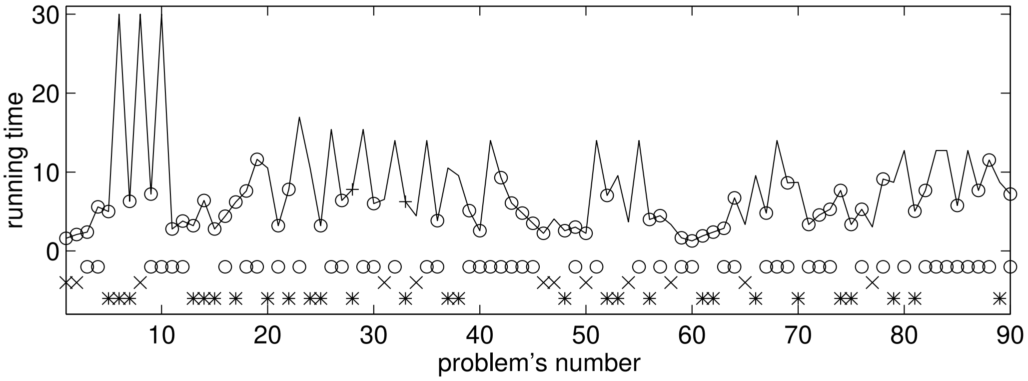


Figure 6: Incremental selection of a method and time bound on ninety transportation problems. The graph shows the running times (solid line), successes (o), and failures (+). The three rows of symbols below the solid line show the choice among APPLY (o), DELAY (x), and ALPINE (∗).

The horizontal axis shows the number of a problem, and the vertical axis is the running time; we mark successes by circles and failures by pluses. The rows of symbols below the curve show the method selection: a circle for APPLY, a cross for DELAY, and an asterisk for ALPINE.

The total gain is 998.3, which is 11.1 per problem. The overall time of the statistical computations is 0.78, which is about 0.01 per problem. The selection converges to the use of APPLY with the time bound 12.7, which is the optimal for this set of ninety problems. If we used the final selection on all problems, we would earn 13.3 per problem. The convergence is slower than in the bound-selection experiments (Figure 5) because we test each method only on about one third of all problems.

5 Empirical examples

We give results of the statistical selection in two other domains. First, we consider an extended transportation domain, which includes airplanes for carrying packages between cities and vans for local deliveries [Veloso, 1994]. In Table 3, we give the performance of APPLY, DELAY, and ALPINE on thirty problems in this domain.

In Figure 7, we present the results of the incremental learning of a time bound for the reward of 400.0. 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 it for all problems, we would earn 135.4 per problem. DELAY earns 131.1 per problem and chooses the 105.3 bound at the end of the learning process. The actual optimal bound is 98.4, which would give 153.5 per problem. Finally, ALPINE gains 243.5 per problem and chooses the bound 127.6. The optimal bound for ALPINE is 430.8, which would give the per-problem gain of 255.8. As a side note, ALPINE outperforms APPLY and DELAY because it uses abstraction, which separates the problem of between-city transportation from within-city deliveries.

Although the bound learned for ALPINE is much smaller than the optimal (127.6 versus 430.8), the resulting gain is close to the optimal. In this experiment, ALPINE's dependency of the expected gain on the time bound has a long plateau, and the choice of a bound within the plateau does not make much difference. Note that ALPINE's optimal bound is larger than the reward (430.8 versus 400.0); this observation shows imperfection of the heuristic for choosing

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

Table 3: Performance in the extended transportation domain.

the initial bound, which assumes that the optimal bound is no larger than the reward.

In Figure 8, we show the results of the incremental selection of a method; we first use the thirty problems from Table 3 and then sixty other problems. The learning process converges to the choice of ALPINE with the bound 300.6, and gives the gain of 207.0 per problem. The optimal choice for these problems is ALPINE with the time bound 517.1, which would yield 255.8 per problem. We have identified this optimal choice in a separate experiment, by running every method on all ninety problems.

We next apply the learning technique to the bound selection when calling a friend on the phone. We determine how many seconds (or rings) a caller should wait for an answer before hanging up. The reward for reaching a friend may be determined by the time that the caller is willing to wait in order to talk now, as opposed to calling later. In Table 4, we give the times for sixty calls, rounded to 0.05 seconds; we have made these calls to sixty different people at their home numbers, and measured the time from the beginning of the first ring, skipping the connection delays. A success occurs when our party answers the phone, whereas a reply by an answering machine is a failure.

The graph in Figure 9(a) shows the dependency of the expected gain on the time bound, for the rewards of 30.0, 90.0, and 300.0. We assume that the caller is not interested in leaving a message, which means that a reply by a machine gets the reward of zero. The optimal bound for the 30.0 and 90.0 rewards is 14.7 (three rings), whereas the optimal bound for the 300.0 reward is 25.5 (five rings).

If the caller plans to leave a message, then the "failure" reward is not zero, although it may be smaller than the success reward. The graph in Figure 9(b) shows the expected gain for the success reward of 90.0 with three different failure rewards, 10.0, 30.0, and 90.0. The optimal bound for the failure reward of 10.0 is 26.7 (five rings); for the other two rewards, it is 32.9 (six rings).

The graph in Figure 10 shows the results of selecting the bounds incrementally, for the 90.0 success reward and zero failure reward. The learned bound converges to the optimal bound,

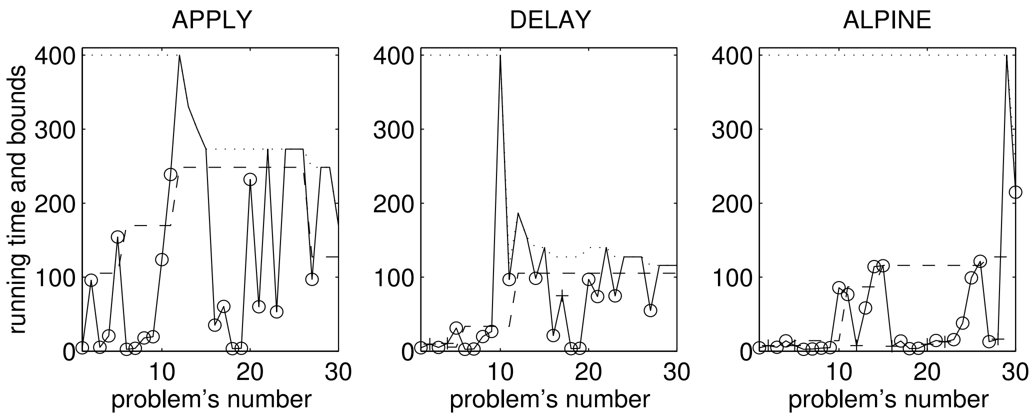


Figure 7: Incremental learning of time bounds in the extended transportation domain: the running times (solid lines), selected bounds (dotted lines), and maximal-gain bounds (dashed lines). The successes are marked by circles (o) and the failures by pluses (+).

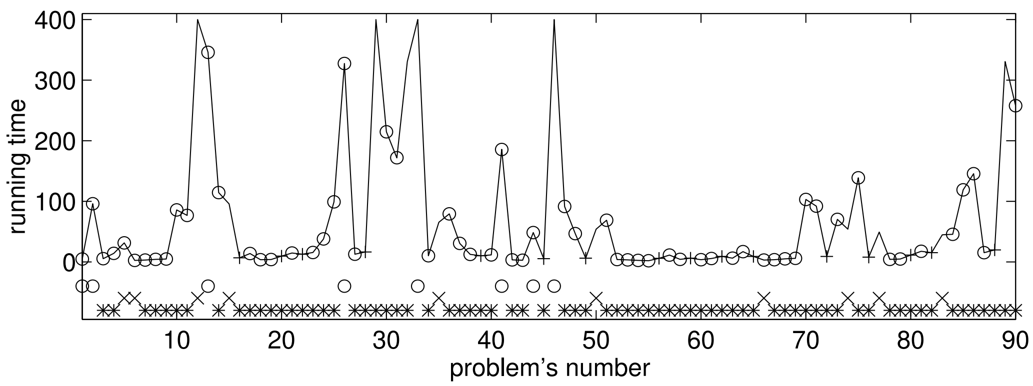


Figure 8: Selection of a method in the extended transportation domain: the running times (solid line), successes (o) and failures (+), and the choice among APPLY (o), DELAY (x), and ALPINE (∗).

which is 14.7. The average gain obtained during the learning process is 38.9 per call. If we used the optimal bound for all calls, we would earn 41.0 per call.

To summarize, the experiments in the two PRODIGY domains and phone-call domain show that the learning procedure usually finds an appropriate bound and yields a near-optimal gain. In Section 8, we will give a series of tests with artificially generated time values, using normal, log-normal, uniform, and log-uniform distributions, and show that the learning gives good results for all four distributions.

6 Problem sizes

We have considered the task of finding a method and time bound that work well for most problems. If we can estimate the sizes of problems, we improve the performance by adjusting the time bound to a 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 usually a difficult task, but many domains allow

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

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

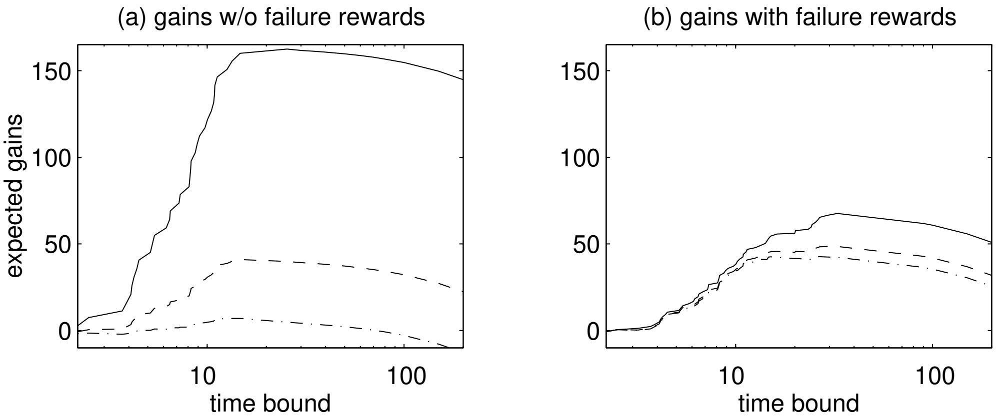


Figure 9: Dependency of the expected gain on the time bound in the phone-call domain: (a) for the rewards of 30.0 (dash-and-dot line), 90.0 (dashed line), and 300.0 (solid line); (b) for the success reward of 90.0 and failure rewards of 10.0 (dash-and-dot line), 30.0 (dashed line), and 90.0 (solid line).

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 Tables 1 and 3, we show the number of packages in each problem. Note that measures of a problem size are usually domain-specific, and the choice of a good measure is the user's responsibility. We allow the user to specify different measures for different problem-solving methods.

We apply regression to find the dependency between the sizes of sample problems and the times to solve them using separate regressions for successes and failures. In PRODIGY, successes usually occur after exploring a small part of the search space, whereas failures require the exploration of the entire space, and the dependency of the success time on the problem size is different from that of the failure time.

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 Figures 11(a) and 11(b), we give the regression expressions for a polynomial dependency between size and time; the regression

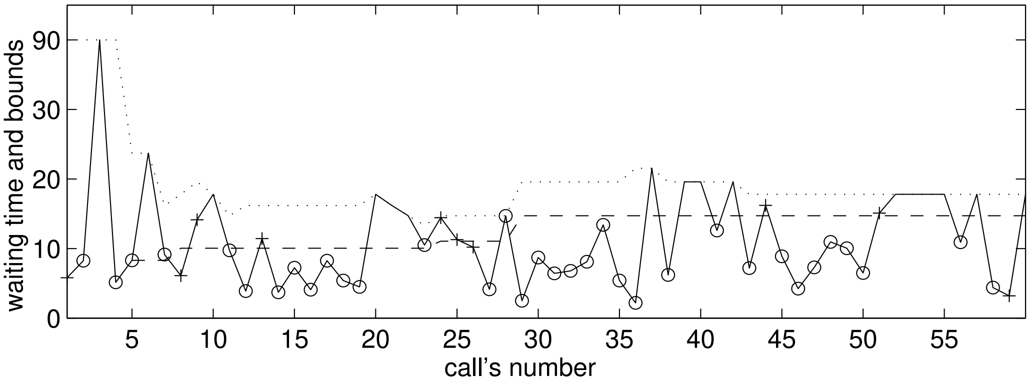


Figure 10: Incremental learning of a time bound in the phone-call domain.

(a) Approximate dependency of the running time on the problem size:

$\ln time = \alpha + \beta \cdot \ln size$;

that is, $time = e^{\alpha} \cdot size^{\beta}$.

(b) Regression coefficients:

$\beta = \frac{\sum_{i=1}^{n} \ln size_i \cdot \ln time_i - SizeSum \cdot TimeSum/n}{SizeSqrSum - SizeSum^2/n}$,

$\alpha = (TimeSum - \beta \cdot SizeSum)/n$,

where

$TimeSum = \sum_{i=1}^{n} \ln time_i$,

$SizeSum = \sum_{i=1}^{n} \ln size_i$,

$SizeSqrSum = \sum_{i=1}^{n} (\ln size_i)^2$.

(c) The t value, for evaluating the regression accuracy:

$t = \frac{\beta}{TimeDev} \cdot \sqrt{SizeSqrSum - SizeSum^2/n}$,

where

$TimeDev = \sqrt{\frac{1}{n-2} \cdot \left(\sum_{i=1}^{n} (\ln time_i)^2 - TimeSum^2/n - \beta \cdot \left(\sum_{i=1}^{n} \ln size_i \cdot \ln time_i - SizeSum \cdot TimeSum/n\right)\right)}$.

Figure 11: Regression coefficients and the t value for the polynomial dependency of time on size.

for an exponential dependency is similar. We denote the number of sample problems by n, the problem sizes by $size_1, ..., size_n$, and the corresponding running times by $time_1, ..., time_n$.

In Figure 12, we give the results of regressing the success times for the transportation problems from Table 1. The top three graphs show the polynomial dependency, whereas the bottom graphs are for the exponential dependency. The horizontal axes show the problem sizes, and the vertical axes are the running times. The circles mark the sizes and times of the problem instances, and the solid lines are the regression results.

We evaluate the regression using the t-test, where the t value is the ratio of the estimated slope of the regression line to the standard deviation of the slope estimate. We give the expression for t in Figure 11(c), where *TimeDev* is the standard deviation of time logarithms. 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; it is a function of the t value and the number n of sample problems. When the regression gives

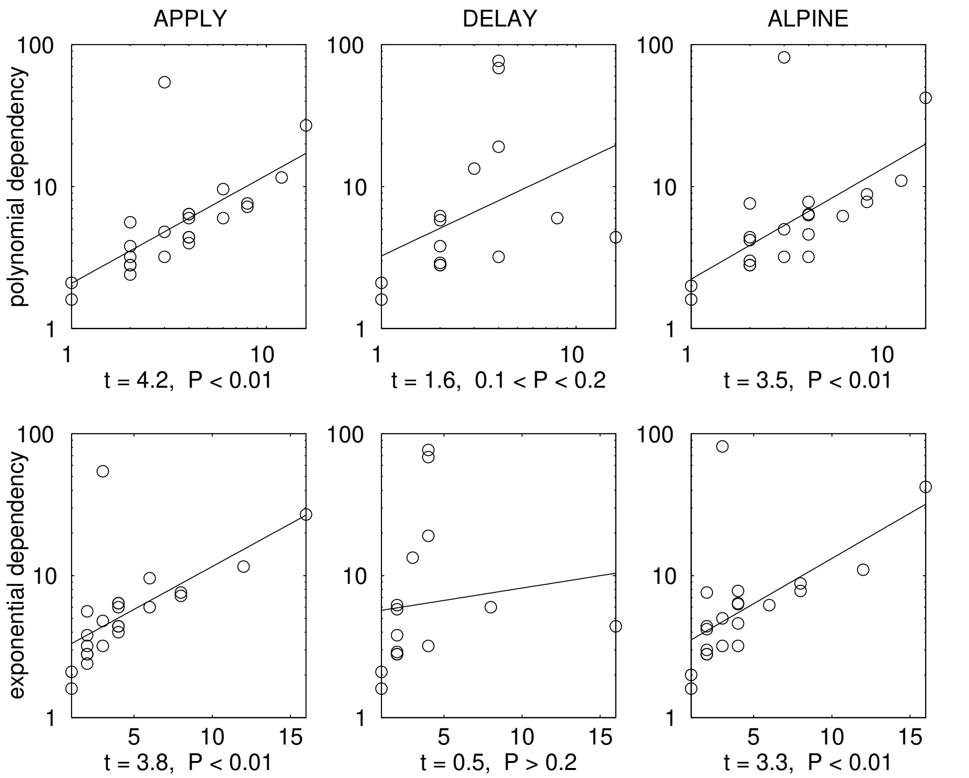


Figure 12: Dependency of the success time on the problem size. The top graphs show the regression for a polynomial dependency, and the bottom graphs are for an exponential dependency.

a good fit, t is large and P is small. In Figure 12, we give the t values and the corresponding intervals of the P value.

We use the regression only if P is smaller than a certain threshold; in the experiments, we have set this threshold to 0.2; that is, we have used sizes when $P < 0.2$. We have chosen 0.2 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. For example, all three polynomial regressions in the top row of Figure 12 pass the $P < 0.2$ test, and the exponential regressions for APPLY and ALPINE also satisfy this condition, whereas the exponential regression for DELAY fails the test.

The choice between the polynomial and exponential regression is based on the t value; specifically, we prefer the regression with the larger t. In Figure 12, the polynomial regression wins for all three methods. The user has an option to select between the two regressions herself; for example, she may insist on the exponential regression. We also allow the user to set a regression slope, which is useful when the past data are insufficient for an accurate estimate. If the user specifies a slope, the system uses her value in the regression; however, it compares the user's value with the regression estimate of Figure 11, determines the statistical significance of the difference, and gives a warning if the user's estimate is off with high probability.

Note that the least-square regression and related t-test make quite strong assumptions about the distribution. First, for problems of fixed size, the distribution of the time logarithms must be normal; second, for all problem sizes, the standard deviation of the distribution must be the same. In practice, however, the regression provides a good approximation even when these

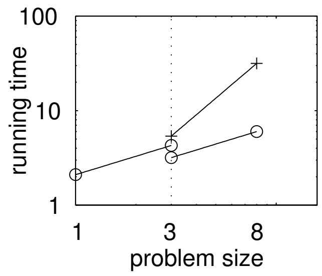


Figure 13: Scaling two success times (o) and a failure time (+) of DELAY to a 3-package problem.

assumptions are not satisfied.

The use of the problem size in estimating the gain is based on "scaling" the times of sample problems to a given size. We illustrate it in Figure 13, 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. To scale a problem's time to a given size, we draw the line with the regression slope through the point representing the problem (solid lines in Figure 13), to the intersection with the vertical line through the given size (dotted line); the ordinate of the intersection is the scaled time.

If the size of a problem is $size_{\mathrm{old}}$, the running time is $time_{\mathrm{old}}$, and we need to scale it to a size $size_{\mathrm{new}}$, using a regression slope β, then we compute the scaled time $time_{\mathrm{new}}$ as follows:

Polynomial regression:
$\ln time_{\mathrm{new}} = \ln time_{\mathrm{old}} + \beta \cdot (\ln size_{\mathrm{new}} - \ln size_{\mathrm{old}})$;
that is, $time_{\mathrm{new}} = time_{\mathrm{old}} \cdot (size_{\mathrm{new}}/size_{\mathrm{old}})^{\beta}$.

Exponential regression:
$\ln time_{\mathrm{new}} = \ln time_{\mathrm{old}} + \beta \cdot (size_{\mathrm{new}} - size_{\mathrm{old}})$;
that is, $time_{\mathrm{new}} = time_{\mathrm{old}} \cdot \exp(\beta \cdot (size_{\mathrm{new}} - size_{\mathrm{old}}))$.

We use the slope of the success regression in scaling success times, and the slope of the failure regression in scaling failures. The slope for scaling an interrupt should depend on whether the method 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. We have also experimented with "distributing" each interrupt point between success and failure slopes, similar to the distribution of small interrupt times described in Section 3; however, it did not provide higher accuracy than the simple heuristic.

For a sample of n successes, m failures, and k interrupts, the overall time of computing the polynomial and exponential regression, selecting between the two regressions, and scaling the sample times is $(n+m+k)\cdot 9\cdot 10^{-4}$ seconds. For the incremental learning, we have implemented a procedure that updates the slope and t value after adding each new problem to the sample; its amortized time is $((n+m+k)\cdot 2+7)\cdot 10^{-4}$ seconds per problem.

After scaling the sample times to a given size, we apply 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 by 2 because the success and failure regressions

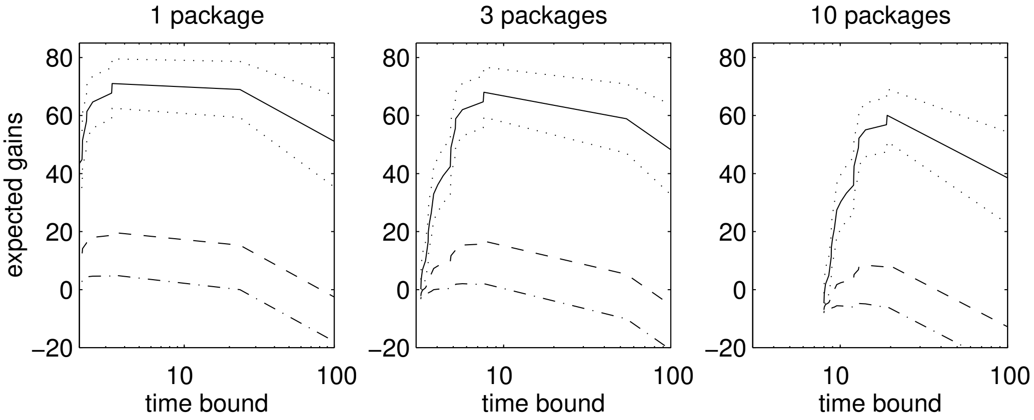


Figure 14: Dependency of APPLY's expected gain on the time bound, for rewards of 10.0 (dash-and-dot lines), 30.0 (dashed lines), and 100.0 (solid lines). The dotted lines show the standard deviation for the 100.0 reward.

	w/o sizes	with sizes
transportation by vans (Section 4)		
APPLY's bound selection	12.0	12.2
DELAY's bound selection	3.9	4.7
ALPINE's bound selection	11.3	11.9
method selection	11.1	11.8
transportation by vans and airplanes (Section 5)		
APPLY's bound selection	110.1	121.6
DELAY's bound selection	131.1	137.4
ALPINE's bound selection	243.5	248.3
method selection	207.0	215.6

Table 5: Per-problem gains in the learning experiments, without and with the use of sizes.

reduce the degrees of freedom of the sample, which means that the deviation is as follows:

$$\sqrt{\frac{SqrSum - \frac{Sum^2}{n+m+k}}{(n+m+k) \cdot (n+m+k-e-3)}}.$$

In Figure 14, 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 sizes in the experiments of Sections 4 and 5, we get larger gains in all eight experiments, as shown in Table 5.

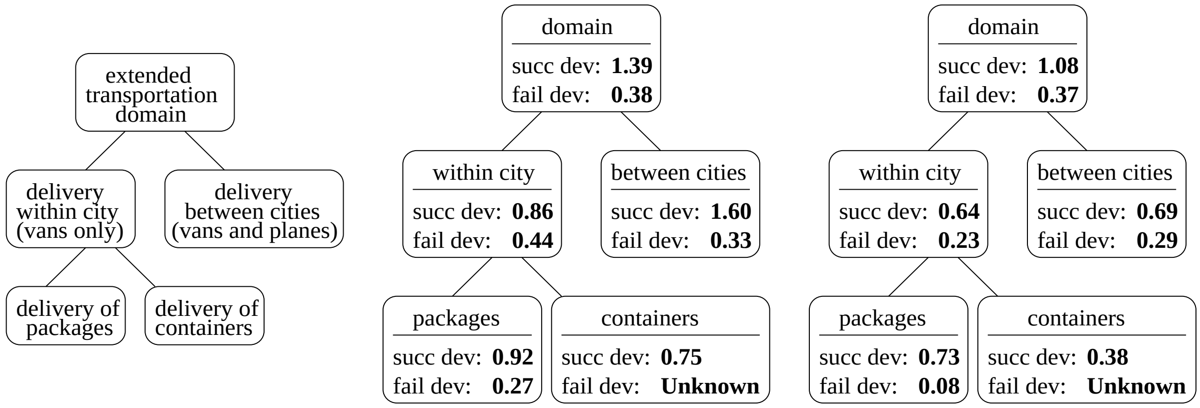
7 Similarity hierarchy

We have estimated the expected gain by averaging the gains for *all* sample problems. If some old problems are especially similar to a new problem, we may improve the estimate by averaging only the gains for these similar problems.

We represent similarity among problems by a tree-structured *similarity hierarchy*. The leaf nodes of the hierarchy are groups of similar problems, and the other nodes represent weaker similarity among groups; we assume that each problem belongs to exactly one group. For instance,

#	time (sec) and outcome			# of
	APPLY	DELAY	ALPINE	conts
1	2.3 s	2.3 s	2.1 s	1
2	3.1 s	5.1 s	4.1 s	2
3	5.0 s	20.2 s	4.8 s	3
4	3.3 s	8.9 s	3.2 s	2
5	6.7 s	36.8 s	6.4 s	4
6	200.0 b	200.0 b	10.1 f	8
7	3.2 s	3.2 s	3.2 s	2
8	24.0 s	200.0 b	26.3 s	8
9	4.8 s	86.2 s	3.4 s	4
10	8.0 s	200.0 b	9.4 s	6

Table 6: Performance on ten container-delivery problems.



(a) Similarity hierarchy. (b) ALPINE's deviations w/o regression. (c) ALPINE's deviations with regression.

Figure 15: Similarity hierarchy (a) and the deviations of ALPINE's success and failure logarithms (b, c); we give the deviations computed without the regression (b), and the deviations with the regression (c).

we may divide transportation problems into within-city and between-city deliveries. We extend this example with a new type of problems, which involves transportation of containers within a city. A van can carry only one container at a time, which makes container delivery harder than package delivery. In Table 6, we give the performance of APPLY, DELAY, and ALPINE on ten problems involving containers. We subdivide within-city problems into package and container deliveries, and show the resulting hierarchy in Figure 15(a).

The construction of a hierarchy is the user's responsibility. We allow the user to construct a separate hierarchy for each problem-solving method or a common hierarchy for all methods. We also allow different problem-size measures for different groups of problems.

We may estimate the similarity of problems in a group by the standard deviation of the logarithms of running times, computed for the sample problems in the group:

$$TimeDev = \sqrt{\frac{1}{n-1} \cdot \left(\sum_{i=1}^{n} (\ln time_i)^2 - (\sum_{i=1}^{n} \ln time_i)^2 / n \right)}.$$

We compute the deviations separately for successes and failures, and use them as a heuristic measure of the hierarchy's quality; the smaller the deviations for the leaf groups, the better the hierarchy. If we use the regression, we apply it separately to each group in the hierarchy. If the regression confirms the dependency between sizes and times, we compute the deviation of time logarithms by a different expression, given in the last line of Figure 11. For example, the deviation values for ALPINE are as shown in Figure 15. Note that the deviations do not change if we multiply all times by the same factor, which means that they do not depend on the speed of a specific computer.

For example, the deviation values for ALPINE are as shown in Figure 15. We give the deviations computed without the regression in Figure 15(b), and the deviations with the regression in Figure 15(c).

We may estimate the expected gain for a new problem by averaging the gains of the sample problems in the same leaf group. Alternatively, we may use a larger sample from one of its ancestors. The leaf group has less data than its ancestors, but the deviation of these data is smaller, and we need to analyze this trade-off when selecting between the leaf group and its ancestors.

We present a heuristic for selecting between a group and its parent based on two tests. The first test shows the difference between the distribution of the group's problems and the distribution of the other problems in the parent's sample. If the two distributions are different, we use the group rather than its parent. If not, we perform the second test, to determine whether the group's sample provides a more accurate estimate than the parent's sample. We now describe the two tests in detail.

If we do not use the regression, the first test is the statistical t-test that shows whether the mean of the group's time logarithms differs from the mean of the other time logarithms in the parent's sample. We perform the test separately for successes and failures. In the experiments, we have considered the means different when we can reject the null-hypothesis that they are equal with the 0.75 confidence. If we use the regression, then we apply a different t-test; specifically, we determine whether the regression lines are different with the 0.75 confidence.

A statistically significant difference for either successes or failures shows that the distribution of the group's running times differs from the distribution of the other problems in the group's parent, which means that we should use the group rather than its parent.

For example, suppose that we use the data in Tables 1, 3, and 6 with the hierarchy in Figure 15(a), and we need to estimate ALPINE's gain on a new problem that involves the delivery of packages within a city. We consider the choice between the corresponding leaf group and its parent; in this example, we do not use the regression. The estimated mean of the success-time logarithms for the package-delivery problems is 4.07, and its standard deviation is 0.20. The estimated mean for the other problems in the parent group is 4.03, and its deviation is 0.16. The difference between the two means is not statistically significant. Since the container-transportation sample has only one failure, we cannot estimate the deviation of its failure logarithms; thus, the difference between the failure-logarithm means is also insignificant.

The second test is the comparison of the standard deviations of the mean estimates for the group and its parent. The deviation of the mean estimate is equal to the deviation of the time logarithms divided by the square root of the sample size, $\frac{TimeDev}{\sqrt{n}}$. We compute it separately for successes and failures, and use it as an indicator of the sample's accuracy in estimating the gain; the smaller the value, the greater the accuracy.

If the group's deviation of the mean estimate is smaller than that of the group's parent, for either successes or failures, then the group's sample is likely to provide a more accurate estimate; thus, we prefer the group to its parent. On the other hand, if the parent's deviation is smaller for both successes and failures, then we use the parent.

Suppose that we apply the second test for estimating ALPINE's gain on within-city package delivery. The standard deviation of the success-time estimate for the leaf group is 0.20, and the deviation for its parent is 0.16. The deviation of the failure-time estimate is also smaller

	using leaf groups	using the top group	heuristic group selection
	without problem sizes		
APPLY's bound selection	11.8	10.5	12.1
DELAY's bound selection	7.0	4.7	7.5
ALPINE's bound selection	19.5	18.1	19.5
method selection	13.1	11.1	13.4
	with problem sizes		
APPLY's bound selection	16.3	11.1	16.8
DELAY's bound selection	12.1	5.2	12.0
ALPINE's bound selection	22.6	18.4	22.6
method selection	19.4	13.7	21.0

Table 7: Per-problem gains in learning experiments, for different group-selection techniques.

for the parent; thus, we prefer the use of the parent.

After selecting between the leaf group and its parent, we apply the same two tests to choose between the resulting "winner" and the group's grandparent; then, we compare the new winner with the great-grandparent, and so on. In the example, we compare the selected parent group with the top-level node (Figure 15a). After applying the first test, we find out that the mean of the group's success logarithms is 4.03, whereas the corresponding mean for the other problems in the top node's sample is 5.39. The difference is statistically significant; thus, we prefer the group of within-city problems to the top-level group.

The running time of the statistical computations is proportional to the height of a hierarchy. For n successes, m failures, and k interrupts, the amortized time of performing the regressions, selecting a group, and scaling the times to the size of the new problem is $((n+m+k)\cdot 4+20)\cdot \mathit{height}\cdot 10^{-4}$ seconds, which is still small compared to PRODIGY's search time.

In Table 7, we present the results of using the hierarchy of Figure 15. We ran the bound-selection experiments on a sequence of seventy problems, constructed by interleaving the problem sets of Tables 1, 3, and 6. We used a three-times longer sequence of transportation problems for the method-selection experiments. The first column includes the results of using only leaf groups in estimating the gains. The second column shows the results of using the top-level group for all estimates, which means that the algorithm does not distinguish among the three problem types. The third column contains the results of using the hierarchy with the heuristic for group selection. The complete hierarchy gives larger gains than either the leaf groups or the top-level group; however, the improvement is not large.

We next use a similarity hierarchy in selecting a time bound for phone calls. We consider the outcomes of sixty-three calls to six different people. We have called two of them, say A and B, at their office phones; we have called the other four, C, D, E, and F, at their homes. We show the similarity hierarchy and call outcomes in Figure 16.

For each group in the hierarchy, we give the estimated mean of success and failure time logarithms ("mean"), the deviation of the time logarithms ("deviation"), and the deviation of the mean estimate ("mean's dev"). The mean of success-time logarithms for office calls is significantly different from that for home calls, which implies that the distribution of office-call times differs from home-call times. The mean success logarithms for persons A and B do not differ significantly from each other. Similarly, the success means of C, D, and E are not

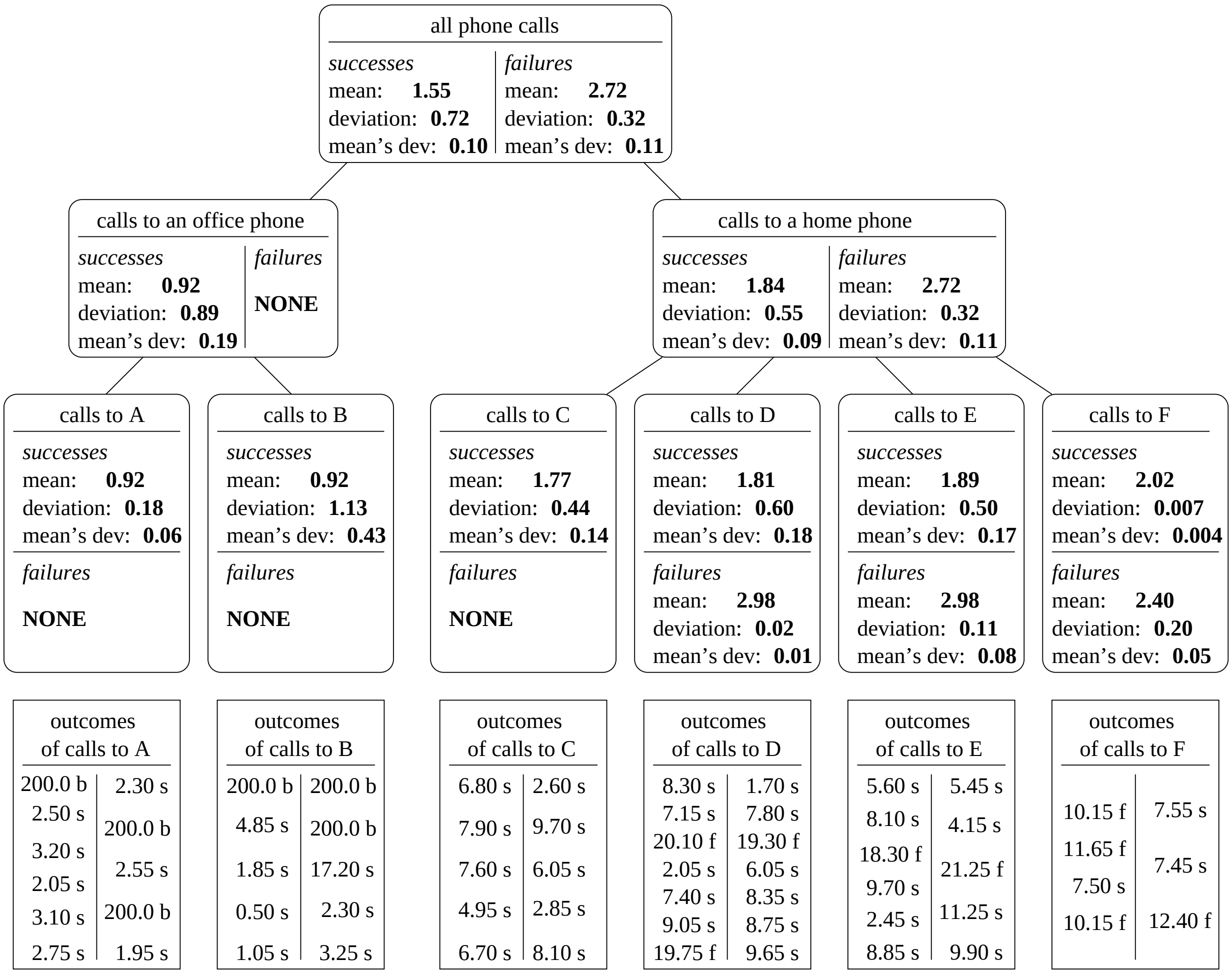


Figure 16: Similarity hierarchy and call outcomes in the phone-call domain.

significantly different from the mean of the home-call group. On the other hand, the success mean of F is significantly different from the other people in the home-call group. Finally, the failure-logarithm means of D, E, and F are all significantly different from each other.

We have run incremental-learning experiments with the reward of 90.0. An experiment with the use of leaf groups for all estimates has yielded the gain of 57.8 per call. We have then run an experiment using the home-call and office-call groups for all estimates, without distinguishing among different people within these groups, and obtained the average gain of 56.3. We have next used the top-level group for all estimates, which has yielded 55.9 per call. Finally, we have experimented with the heuristic choice between the leaf groups and their ancestors, and earned 59.8 per call. If we knew all time distributions in advance, determined the optimal time bound for each leaf, and used these optimal bounds for all calls, then the average gain would be 61.9.

The phone-call experiments have confirmed that a similarity hierarchy improves the performance; note that the gain obtained with the hierarchy is much closer to the optimal than the gain from the use of the leaf groups or the top-level group.

8 Artificial tests

We give the results of testing the selection mechanism on artificially generated performance data. The "running times" in these tests are values produced by a random-number generator, which allows controlled experiments with known distributions. The learning mechanism has proved effective for all tested distributions, and we have not found a significant difference in performance for different distributions. The experiments have shown that the gain obtained in the incremental learning is usually close to the optimal. They have also shown that the regression improves the performance when there is a correlation between size and time, and does not worsen the results when there is no correlation.

We have considered four distribution types:

Normal: The normal distribution of success and failure times corresponds to the situation when the running time for most problems is close to some "typical" value, and problems with much smaller or much larger times are rare.

Log-Normal: The distribution is called log-normal if time logarithms are distributed normally; intuitively, it occurs when the "complexity" of most problems is close to some typical complexity, and the problem-solving time grows exponentially with the complexity.

Uniform: The running times belong to some fixed interval, and all values in this interval are equally likely.

Log-Uniform: The logarithms of running times are distributed uniformly; intuitively, the complexity of problems is within some fixed interval, and the running time grows exponentially with the complexity.

For each of the four distribution types, we have run multiple tests, varying the values of the following parameters:

Success and failure probabilities: We have varied the probabilities of success, failure, and infinite looping.

Mean and deviation: We have experimented with different values of the mean and standard deviation of success-time and failure-time distributions.

Reward: We have set the reward to 100.0 in all experiments.

Length of the problem sequence: We have tested the incremental learning on sequences of 50, 150, and 500 problems.

Correlation between sizes and times: We have run tests without and with problem sizes, and experimented with three different correlations between size logarithms and time logarithms: 0, 0.6, and 0.9.

We have run fifty independent experiments for each setting of the parameters and averaged their results; thus, every graph shows the average of fifty experiments.

Since the learning technique has proved effective in all tests, we conjecture that it also works well for most other distributions. We plan to experiment with a wider variety of distributions and identify situations in which the technique does not give good results.

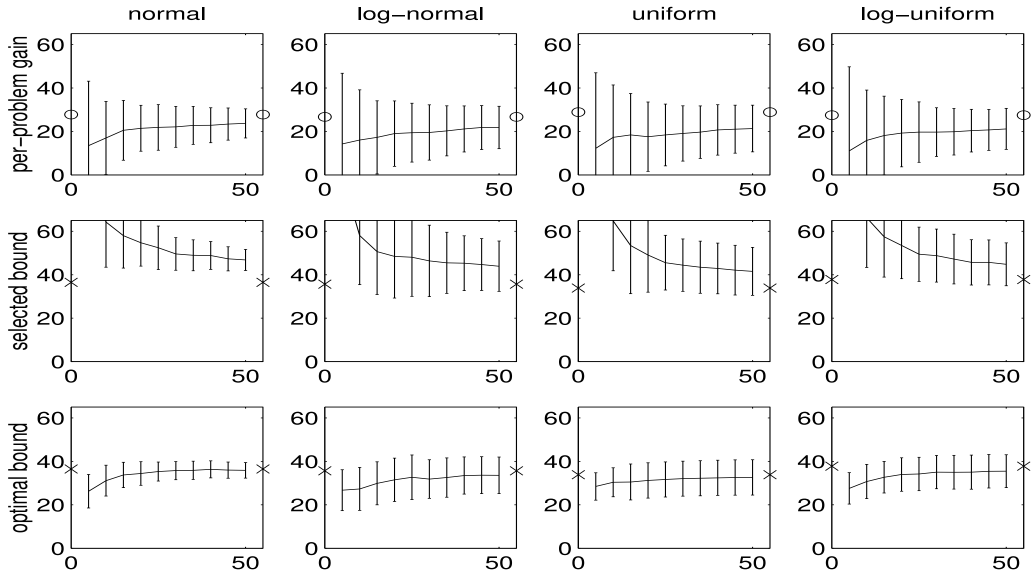


Figure 17: Per-problem gains (top row), time bounds (middle row), and estimates of the optimal bounds (bottom row) in the incremental learning on 50-problem sequences. The crosses mark the optimal time bounds, and the circles show the expected gains for the optimal bounds.

Short and long problem sequences

We first give the results of learning a time bound on sequences of 50 and 500 problems, without the use of problem sizes. The success probability in these experiments is 1/2, the failure probability is 1/4, and the probability of infinite looping is also 1/4. The mean of success times is 20.0 and their standard deviation is 8.0; the failure-time mean is 10.0 and standard deviation is 4.0. We have experimented with all four distribution types; for each distribution, we have run fifty experiments and averaged their results.

In Figure 17, we summarize the results for 50-problem sequences. The horizontal axes in all graphs show the problem's number in a sequence. The top row of graphs gives the average per-problem gain obtained up to the current problem; for example, the left end of each curve shows the average gain for the first five problems, and the right end gives the average for all fifty problems. The circles mark the gain that the system would obtain if it knew the distribution in advance and used the optimal time bound for all problems. The vertical bars show the width of the distribution of gain values obtained in different experiments. Each bar covers two standard deviations up and down, which means that about 95% of the experiments fall within it. The middle row of graphs shows the selected time bounds, and the bottom row gives the system's estimates of the optimal bounds; recall that the selected bounds are larger than the optimal, to encourage exploration. The crosses mark the values of the optimal bounds; note that the system's estimates of the optimal bounds converge to their real values.

In Figure 18, we give similar results for 500-problem sequences. In these experiments, per-problem gains come closer to the optimal values, but still do not reach them. The difference between the obtained and optimal gains comes from losses during early stages of learning and from the use of larger-than-optimal bounds.

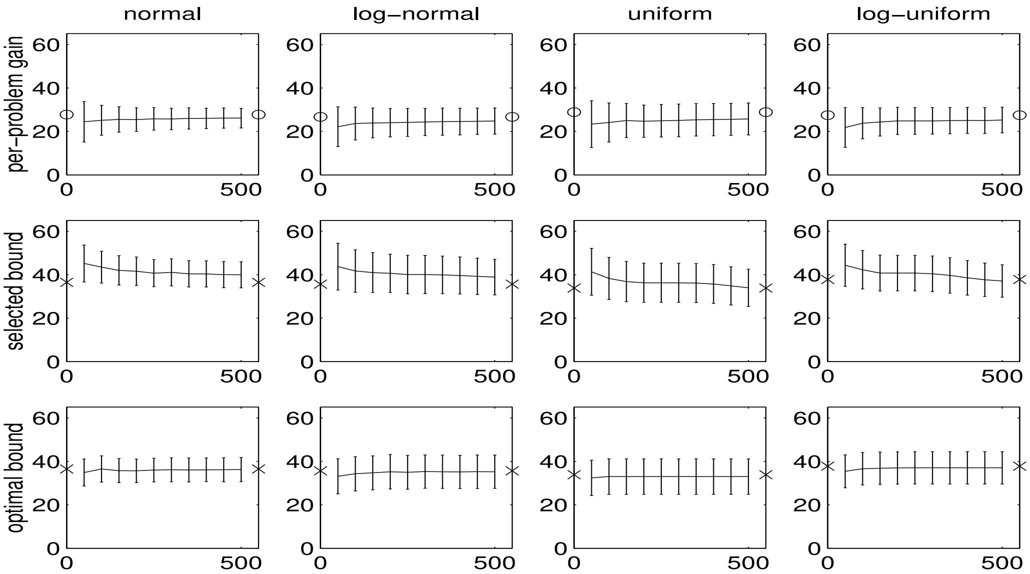


Figure 18: Per-problem gains (top row), time bounds (middle row), and estimates of the optimal bounds (bottom row) in the incremental learning on 500-problem sequences.

Varying success and failure probabilities

We give the results of learning a time bound for different probabilities of successes and failures. The means and standard deviations of the success and failure times are the same as in the previous experiments.

We summarize the results in Figure 19. The top row of graphs is for a method that succeeds, fails, and goes into an infinite loop equally often; that is, the probability of each outcome is 1/3. The middle row gives the results for a method that succeeds half of the time, fails half of the time, and never goes into an infinite loop. Finally, the bottom row is for a method that succeeds half of the time and loops forever otherwise. The solid lines show the average per-problem gain up to the current problem; the dotted lines are the selected time bounds; and the dashed lines are the estimates of the optimal bounds. The crosses mark the optimal bounds, and the circles are the expected gains for the optimal bounds.

Note that, when the probability of infinite looping is zero (middle row), any large time bound gives near-optimal results because the system never needs to interrupt a method. Thus, the system never changes the initial time bound and gets near-optimal gains from the very beginning.

Varying the mean of time distributions

We now vary the mean value of failure times. We keep the mean success time equal to 20.0 with standard deviation 8.0, and experiment with failure means of 10.0 with deviation 4.0, 20.0 with deviation 8.0, and 40.0 with deviation 16.0; we give the results in Figure 20. The gains for normal and log-normal distributions come closer to the optimal values than for uniform and log-uniform distributions; however, the difference is not statistically significant.

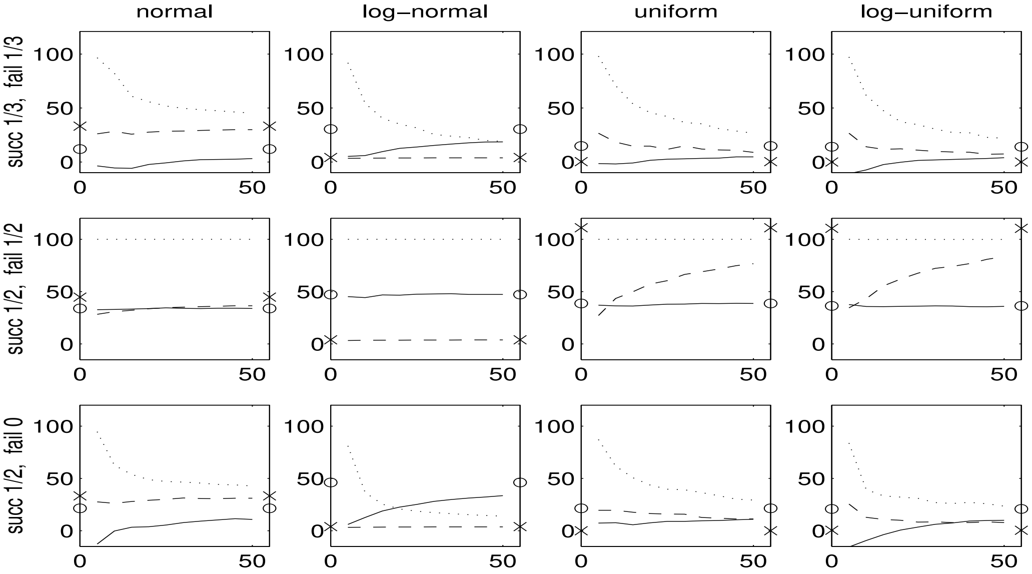


Figure 19: Per-problem gains (solid lines), time bounds (dotted lines), and estimates of the optimal bounds (dashed lines) for different success and failure probabilities. The crosses mark the optimal time bounds, and the circles show the expected gains for the optimal bounds. We give the values of success probability ("succ") and failure probability ("fail") to the left of each row.

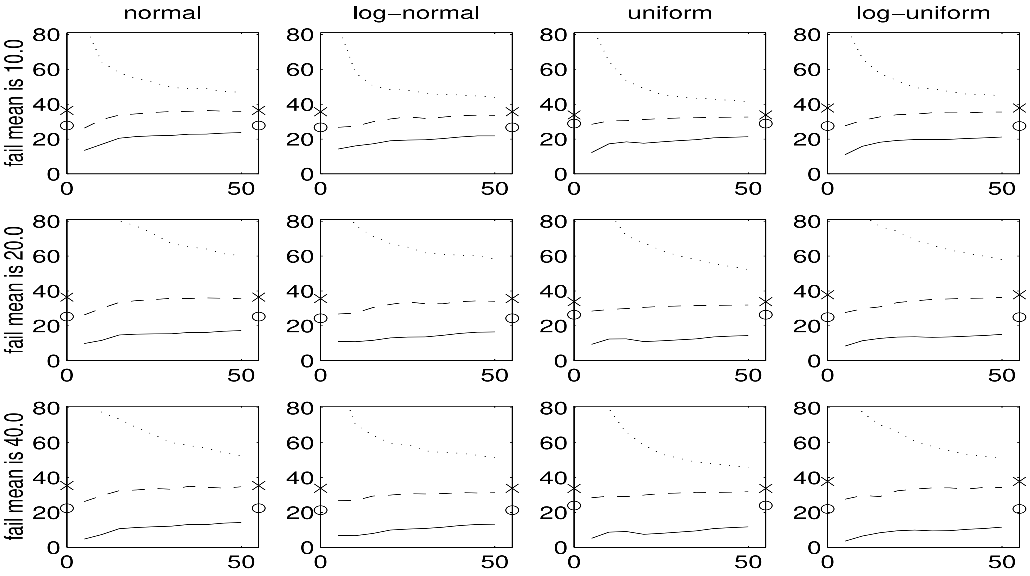


Figure 20: Per-problem gains (solid lines), time bounds (dotted lines), and estimates of the optimal bounds (dashed lines) for different mean values of failure times. The mean of success times is 20.0 in all experiments.

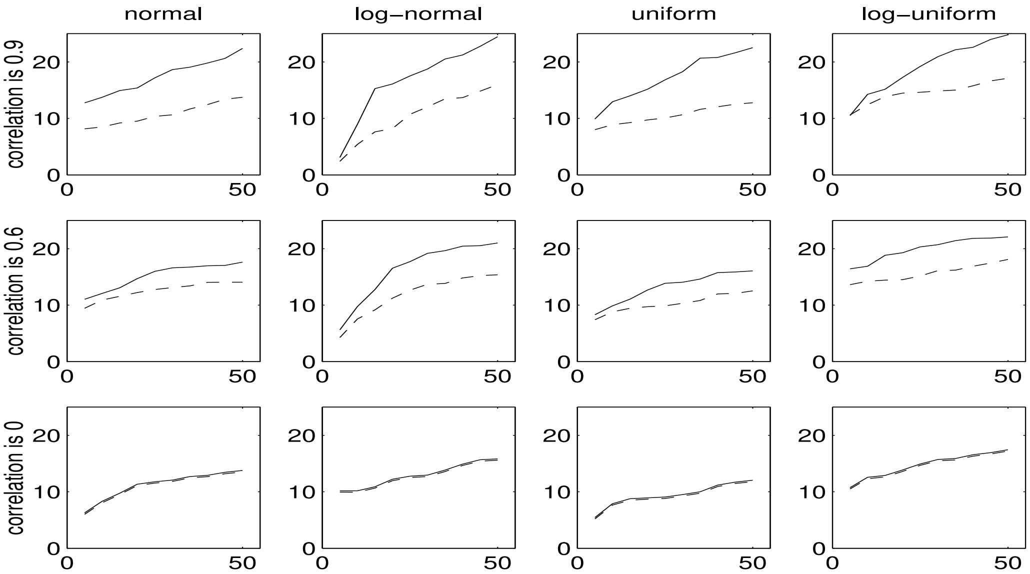


Figure 21: Per-problem gains without sizes (dashed lines) and with sizes (solid lines), for different correlations between size logarithms and time logarithms.

Problem sizes

We compare the gains obtained without and with the regression. Problem sizes in this experiment are natural numbers between 1 and 10, and the logarithms of mean success and failure times are proportional to the problem-size logarithms. We have adjusted the deviation values to obtain desired correlations between time logarithms and size logarithms. We have used the correlation of 0.9 in the first series of experiments and 0.6 in the second series. Finally, we have run experiments with zero correlation; the mean times in this series were the same for all problem sizes.

We give the results in Figure 21, where the dashed lines show the average per-problem gains without the regression, and the solid lines give the gains with the regression. The use of the regression improves the performance, and the improvement is greater for a larger correlation. If there is no correlation, the system disregards the results of the regression and performs identically without and with sizes.

Method selection

Finally, we show the results of the incremental selection among three problem-solving methods on 150-problem sequences. In the first series of experiments, we have adjusted mean success and failure times in such a way that the optimal per-problem gain for the first method is 10% larger than that for the second method and 20% larger than that for the third method.

We give the results in Figure 22. The top row shows the average per-problem gain without the regression (dashed lines) and with the regression (solid lines). The circles mark the expected gains for the optimal time bounds without the regression. The other two rows give the probability of choosing each method, for the experiments without and with problem sizes. The distance from the bottom of the graph to the lower curve is the probability of selecting the first method, the distance between the two curves is the chance of selecting the second

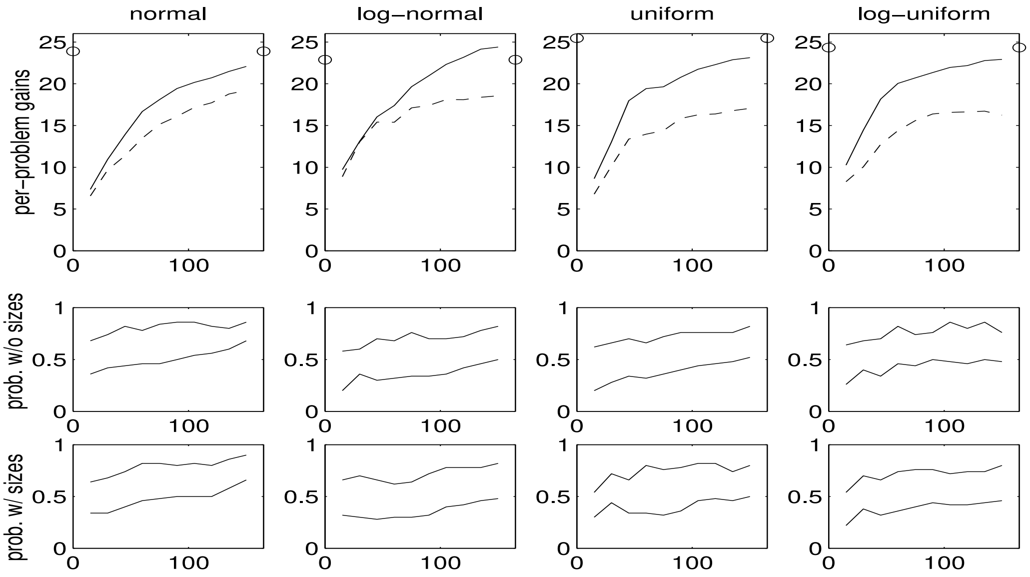


Figure 22: Incremental selection among three methods, where the average gain for the first method is 10% larger than that for the second method and 20% larger than that for the third method. We show the average per-problem gains in the experiments without and with the regression (the top row of graphs), and the probability of selecting each method (the other two rows).

method, and the distance from the upper curve to the top is the third method's chance. The graphs show that the probability of selecting the first method, which gives the highest gain, increases in the process of learning. The probability of selecting the third method, which is the worst-performing, decreases faster than that of the second method.

In the second series of experiments, the optimal gain of the first method is 30% larger than that of the second method and 60% larger than that of the third method. We give the results in Figure 23; note that the probability of selecting the first method grows much faster, due to the larger difference in the expected gains.

9 Conclusions and extensions

We have stated the task of selecting among available problem-solving methods as a statistical problem, derived an approximate solution, and built a system for choosing the most effective method. The system combines exploitation of past experience with exploration of new alternatives. It can use an approximate measure of problem sizes and information about similarity between problems. The selection technique has proved effective for all tested distributions of running times; it gives good results even when the distributions do not satisfy the assumptions of the statistical analysis.

We have implemented heuristics that enhance the statistical technique [Fink, 2003], although we have not used them in the described experiments. In particular, the system allows the user to provide a prediction of the gains for different methods, and then combines the user's prediction with the statistical estimate. If the selected method has failed to solve a problem, the system can choose another method for a second attempt to find a solution; it re-evaluates the gain estimates to incorporate the knowledge that the first attempt has failed. Finally, we provide

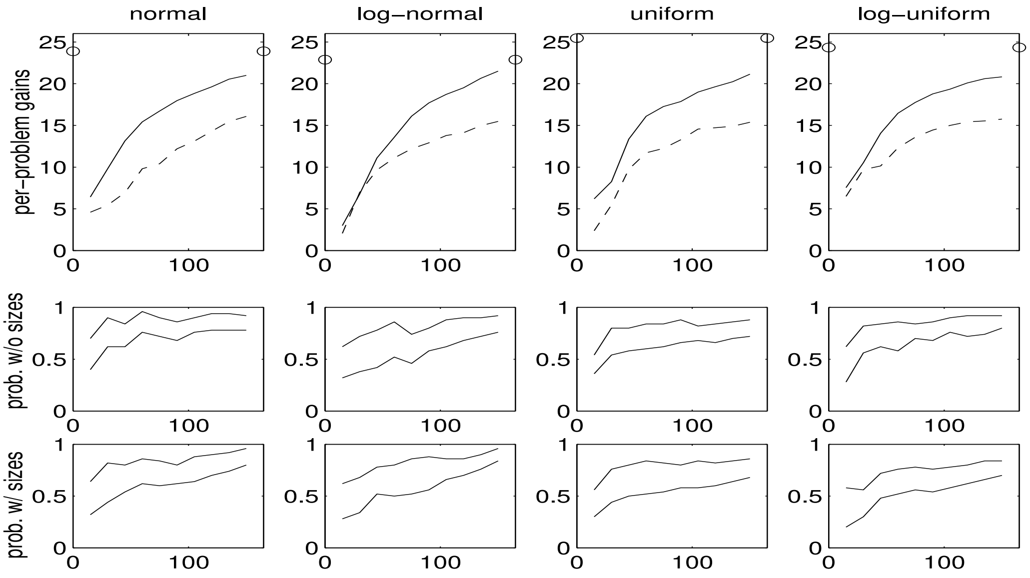


Figure 23: Incremental selection among three methods, where the average gain for the first method is 30% larger than that for the second method and 60% larger than that for the third method.

a mechanism for combining *if-then* preference rules for method selection with the numeric estimates.

The statistical model raises many open problems, which include relaxing the simplifying assumptions, extending the model to account for more features of real-world situations, and improving the heuristics used with statistical estimates. To make the model more flexible, we need to provide a mechanism for switching the method and revising the time bound during the search for a solution. We should also allow interleaving of several promising methods, which is often more effective than sticking to one method [Howe *et al.*, 1999]. Finally, we need to develop a means for learning a similarity hierarchy automatically, to minimize the deviation of time logarithms within similarity groups.

Acknowledgments

I am grateful to Svetlana Vayner, who helped to construct the statistical model and provided a thorough feedback on all ideas. I owe thanks to Herbert Simon, Jaime Carbonell, Manuela Veloso, Martha Pollack, Karen Haigh, Henry Rowley, Josh Johnson, and Hong Tang for their valuable comments and suggestions.

The research has been sponsored in part 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, and by Research and Creative Grant Scholarship program at the University of South Florida.

References

[Allen and Minton, 1996] John A. Allen and Steven Minton. Selecting the right heuristic algorithm: Runtime performance predictors. In Gordon McCalla, editor, *Advances in Artificial*

Intelligence: The Eleventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence, pages 41–53. Springer-Verlag, Berlin, Germany, 1996.

[Bacchus and Yang, 1992] Fahiem Bacchus and Qiang Yang. The expected value of hierarchical problem-solving. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 369–374, 1992.

[Blumer *et al.*, 1987] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24(6):377–380, 1987.

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

[Cohen, 1992] William W. Cohen. Using distribution-free learning theory to analyze solution-path caching mechanisms. *Computational Intelligence*, 8(2):336–375, 1992.

[Cohen, 1995] Paul R. Cohen. *Empirical Methods for Artificial Intelligence.* MIT Press, Cambridge, MA, 1995.

[Fink, 2003] Eugene Fink. *Changes of Problem Representation: Theory and Experiments.* Springer Verlag, Berlin, Germany, 2003.

[Gentner and Stevens, 1983] Dedre Gentner and Albert L. Stevens, editors. *Mental Models.* Lawrence Erlbaum Associates, Mahwah, NJ, 1983.

[Hansen and Zilberstein, 1996] Eric A. Hansen and Shlomo Zilberstein. Monitoring the progress of anytime problem-solving. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1229–1234, 1996.

[Hansson and Mayer, 1989] Othar Hansson and Andrew Mayer. Heuristic search as evidential reasoning. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, pages 152–161, 1989.

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

[Howe *et al.*, 1999] Adele E. Howe, Eric Dahlman, Christopher Hansen, Michael Scheetz, and Anneliese von Mayrhauser. Exploiting competitive planner performance. In *Proceedings of the Fifth European Conference on Planning*, pages 62–72, 1999.

[Knoblock, 1993] Craig A. Knoblock. *Generating Abstraction Hierarchies: An Automated Approach to Reducing Search in Planning.* Kluwer Academic Publishers, Boston, MA, 1993.

[Knoblock, 1994] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.

[Mendenhall *et al.*, 1999] William Mendenhall, Robert J. Beaver, and Barbara M. Beaver. *Introduction to Probability and Statistics.* Duxbury Press, Boston, MA, tenth edition, 1999.

[Minton, 1996] Steven Minton. Automatically configuring constraint satisfaction programs: A case study. *Constraints: An International Journal*, 1(1–2):7–43, 1996.

[Mouaddib and Zilberstein, 1995] Abdelillah Mouaddib and Shlomo Zilberstein. Knowledge-based anytime computation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 775–781, 1995.

[Newell and Simon, 1972] Allen Newell and Herbert A. Simon. *Human Problem Solving.* Prentice Hall, Upper Saddle River, NJ, 1972.

[Pérez, 1995] M. Alicia Pérez. *Learning Search Control Knowledge to Improve Plan Quality.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1995. Technical Report CMU-CS-95-175.

[Polya, 1957] George Polya. *How to Solve It.* Doubleday, Garden City, NY, second edition, 1957.

[Russell *et al.*, 1993] Stuart J. Russell, Devika Subramanian, and Ronald Parr. Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 338–344, 1993.

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

[Simon, 1989] Herbert A. Simon. *Models of Thought*, volume II. Yale University Press, New Haven, CT, 1989.

[Stone *et al.*, 1994] Peter Stone, Manuela M. Veloso, and Jim Blythe. The need for different domain-independent heuristics. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 164–169, 1994.

[Tabachneck-Schijf *et al.*, 1997] Hermina J. M. Tabachneck-Schijf, Anthony M. Leonardo, and Herbert A. Simon. CaMeRa: A computational model of multiple representations. *Cognitive Science*, 21(3):305–350, 1997.

[Valiant, 1984] Leslie G. Valiant. A theory of the learnable. *Communications of the Association for Computing Machinery*, 27(11):1134–1142, 1984.

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

[Veloso *et al.*, 1995] Manuela M. Veloso, Jaime G. Carbonell, M. Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

[Veloso, 1994] Manuela M. Veloso. *Planning and Learning by Analogical Reasoning.* Springer-Verlag, Berlin, Germany, 1994.