The Expected Complexity of Problem Solving

Clark Glymour, Kevin Kelly and Peter Spirtes
Department of Philosophy
Carnegie Mellon University

Abstract

Worst case complexity analyses of algorithms are sometimes held to be less informative about the real difficulty of computation than are expected complexity analyses. We show that the two most common representations of problem solving in cognitive science each admit algorithms that have *constant* expected complexity, and for one of these representations we obtain constant expected complexity bounds under a variety of probability measures.

1. Introduction

Newell and Simon's (1972) *Human Problem Solving* provided the framework for much of contemporary cognitive psychology and artificial intelligence. In one representation of problem-solving, they consider "set-predicate" problems. A set-predicate problem instance consists of a set of entities, a subset having a particular property, a method of enumerating or naming the entities, and some method (an oracle) that determines for each particular entity whether or not it has the property in question. The problem solving task is to find an entity in the set that has the property. The interpretation of the representation is that the entities in the set represent sequences of "operators" or actions of some finite number of kinds, and the goal subset is understood to consist of those sequences of operators that produce a desired result. Another representatfon of probjem solving is not explicit in *Human Problem Solving* but is certainly implicit, and is widely used in the artificial intelligence literature. This second, or finite automaton representation, takes an instance of a problem to be given by a finite automaton whose arc labels signify actions available to the agent, whose nodes signify states of the world resulting from actions, whose final states represent goal states, and whose initial state represents the initial world state in which the problem solver is situated. Some mechanism provides the problem solver with a description of the node, or world state, that results from any sequence of actions the problem solver undertakes. The task is to find a path through the automaton from the initial state to a final state.

A key idea of Newell and Simon's approach is that problem solving is computationally difficult and typically requires recourse to *heuristic* search procedures, and, of course, that real human problem solvers do indeed use heuristic problem solving techniques. It is clear that the obvious algorithms for problem solving in either the set-predicate or the finite automaton representations are indeed *worst-case* exponential in natural measures of the size of a problem instance. Although the enumeration algorithm Newell and Simon describe, and other enumeration algorithms, are worst-case exponential, the interesting question of the *expected complexity* of enumeration procedures for problem solving remains open and relevant. Indeed, cognitive scientists sometimes argue that the worst cases are rare, and expected complexity measures are more relevant to questions of feasibility.[1] In discussing the learning of procedures, John Anderson (1976) considers algorithms that identify finite state machines, observes that enumeration algorithms will be worst-case exponential and asks us to consider the possibility

> ...that target machines may not be equally likely and that the learner's hypotheses progress from the more likely machines to the less likely. For instance, suppose the machines were *ordered,* such that the nth machine had probability aO-a)*"^{1} of being the correct machine. That is, there was a geometric probability density across the possible machines. Then the mean number of machines that need to be considered before the correct machine would be 1/a. Even if the probability of a particular machine being correct is very small, the mean time to success would avoid astronomical values, (p. 502)

In the usual analysis of expected complexity, a problem is taken to be a collection of *instances.* Instances are viewed as having *sizes,* and there are at most finitely many instances of any given size. For each possible instance size, a probability measure, usually the uniform distribution, is assumed over all instances of that size. For any algorithm that solves the problem the expected number of steps the algorithm requires to solve a randomly selected instance of size n is then a well defined function of n. This function, the expected complexity function, can have a very different structure from the worst case complexity function for the same algorithm. Recently, for example, Wilf (1986) has shown that for a well-known NP complete problem, the graph 3-coioring problem, there is an algorithm that has an expected complexity function that is bounded, for all n, by 192. Thus every known algorithm that decides whether or not the vertices of a graph can be assigned three colors, in such a way that no two adjacent nodes receive the same color, requires a number of computational steps that in the worst case increases at least exponentially with the number of vertices of the graph, but some algorithm requires an *average* of 192 steps, for each instance size. What happens is that as the size measure-the number of vertices in a graph-increases, the proportion of graphs that are "easy" to decide also increases in such a way as to keep the average number of computational steps constant.

^e have heard this defense, for example, to objections to artificial intelligence programs that implement procedures that are worst-case exponential.

Essentially the same mathematical phenomenon can occur in problem solving. We will show, using any of several natural measures, that an algorithm for the general case of problem solving in Newell and Simon's set-predicate formulation has a *constant expected complexity*. Further, we will show that there is a constant upper bound on the expected number of operators that must be applied in obtaining a solution to a problem instance of any finite size. Similarly, we will show that if a uniform probability measure is used, there is an algorithm that solves the finite automaton representation that also has a constant expected complexity, although the bound is different.

2. The Expected Complexity of Problems in the Set-Predicate Formulation

Newell and Simon present the "set-predicate" formulation of problem solving this way (pp. 95-96):

> ...to be given a problem in this formulation means to be given, somehow, a set, U, and the goal of finding, producing or determining a member of a subset G of that set--this latter identified most generally by a test that can be performed on the elements of U.

In Newell and Simon's problem solving framework, the elements of U can be regarded as finite sequences of operators. If each operator is identified by a letter in a set Σ, then U and G can be regarded as sets of words over the alphabet Σ, or as subsets of Σ^*, the set of all finite strings of elements of the alphabet.

Newell and Simon describe an algorithm that solves all problem instances of this kind. They illustrate it for the case of theorem proving, where it is called the "British Museum Algorithm."

- Measure the length of a word in Σ^* by the number of letters it contains, and enumerate all words, shorter words before longer words.
- Test each word in turn for membership in G.
- Stop when a word in G is found.

The procedure is guaranteed to find a member of G if there is one. Moreover, it is guaranteed to satisfy a side constraint that is often imposed on problem solving tasks, namely a preference for the shortest sequence of operations that solves the problem instance.

If G is empty the Newell-Simon algorithm will continue forever. In the cases Newell and Simon consider, however, there is generally[2] a feature of any problem instance that can be used to bound the search and

[2]But not always, for example not in the case of theorem proving.

to modify the algorithm so that the procedure stops if no word in G is found within that bound.

Newell and Simon themselves suggest that problem instances come with size bounds:

...the initial space in which the solver encodes the problem already provides some measure of how big a world he has to consider.

The size bound on a set-predicate problem instance could be taken to be n, where n is an upper bound on the length of words in G. The size n is an upper bound on the number of operations necessary to solve the problem instance. Size bounds of this kind are important because they guarantee that a problem instance is finite and can be solved by an exhaustive search of the words of length less than or equal to the bound. If a bound of this kind is assumed to be implicit in the problems addressed, we can without loss of generality take both U and G always to be finite sets, ignoring any sequences of operators longer than the length given by the value of the size measure.

Let Σ be an alphabet. Let Σ^n denote the set of all strings of length no greater than n drawn from symbols in Δ. Now, we take a set-predicate *problem instance* to be an alphabet Σ, a size limit n, a finite subset G of Σ^n, and an oracle for G that given a word in Σ^n returns "yes" or "no" accordingly as the word is or is not in G. A *problem* is the set of all problem instances for a fixed alphabet.

A *deterministic problem solver* is an effective procedure that given Σ and a size bound effectively determines at each step what word to query the oracle about, and conjectures a word after making queries to the oracle. A deterministic problem solver *solves a problem instance* iff it conjectures a word in G and then immediately halts. A problem solver *solves a problem* iff it solves every instance of the problem.

In this paper we examine two different measures of problem solving complexity. The *oracle call* complexity of a deterministic problem solver on a particular problem instance is the number of calls to the oracle the problem solver makes before halting. The *operator application* complexity of a problem solver on a problem instance is the sum of the lengths of the words queried before the problem solver halts on that instance. Clearly, the operator application complexity of a problem solver is never smaller than its oracle call complexity.

For a given problem and complexity measure, the number of problem instances of any given size is finite and there is a worst value for the complexity of a deterministic problem solver for instances of that size (assuming that the problem solver never queries the oracle about the same hypothesis twice). For a deterministic problem solver and problem, the worst-case complexity is a function of the size of problem instances. It is straightforward to show that there are problems no algorithm can solve with only polynomial^ many oracle calis. (See Propositions 1 and 2 of the Appendix). But does the worst-case analysis imply that problem-solving is infeasible in an intuitive sense? We will show that for a variety of probability distributions over problem instances, the answer is no.

First, we assume for any problem, in the sense of "problem" defined above, and for any value of the size measure for problem instances, a uniform probability distribution on the problem instances of that size. An enumeration algorithm then has, for every instance size, an expected complexity for instances of that size.

Theorem 1. For any set-predicate problem, the expected number of oracle calis of any enumeration algorithm is bounded above by 2.

Each member of U is a finite sequence of operators, and also a word in the language Z\ Theorem 1 says that on the average, for problem instances of any given size, we have to query no more than 2 words before we find a word in G. The result is intuitive on a little reflection. Denote the universe set U for instances of size n by U(n). Let S(n) be the cardinality of U(n); it is equal to

$$\sum_{i=1}^{n} |\Sigma|^{n}$$

where |£| is the cardinality of I.

The problem instances of size n correspond to all ways of choosing solution sets G from U(n). The number of such subsets is 2^{S}W, and under the uniform measure each is as probable as any other. The number of such instances that contain the first word in the enumeration is 2^{s}<^{n}^\ and so 2^{s}<^{n}>-^{1}/2^{s}(^{n}) = 1/2 is the probability that the *very first* conjecture is correct. The number of instances that do not contain the first word conjectured but do contain the second is $2^{S(n}$>*^{2}, and so the probability that the second word in the enumeration is the first word in G to be conjectured is 1/4. In general, the probability that the kth word is the first in G to be conjectured is $1/2^{K}$. The expected number of conjectures is therefore

S(n) .

Yi

Since the sum of this series from 1 to infinity is 2, the result follows, and in tact it follows that 2 is the smallest bound that holds for all n.

Now let us consider the expected operator application complexity of problem solving. That is, how many "actions" must be taken, on the average, by the problem solver before it succeeds.

Theorem 2. For any set-predicate problem with an alphabet of 2 or more letters, the expected number of occurrences of letters in words that the oracle is queried about by any enumeration algorithm that enumerates shorter words before longer words is bounded above by 10.

A proof of this result is given in the Appendix. We do not claim that the bound given here is the best possible.

To illustrate these results, imagine problem instances of the following kind: A door with a combination lock must be opened. In actual safes, one sequence of numbers (the combination) opens the safe. Suppose that in this case *sets* of sequences of numbers open the safe, i.e. it is possible that more than one combination opens the safe. With a problem instance of this kind, we consider a letter to be a dial setting, and a word to be a sequence of dial settings. For a problem instance of size n, we know that a sequence of no more than n dial settings is required to open the door, so we need not consider words on the alphabet of lock settings of length greater than n. In this case U is the set of all such sequences, and G is any subset of U. The first theorem tells us that for *every* positive n, if we consider all problem instances of size n, the average number of sequences tried before the safe is successfully opened is never greater than 2; the second theorem tells us that in the same circumstance the average number of separate dial twists performed before the door opens is no greater than 10.

Problems that consist of compositions of set-predicate problems may be straightforwardly analyzed using these results. For example, consider a safe with n dials that is opened by n-tuples of sequences, where each sequence in an n-tuple corresponds to a sequence entered on a different dial. If for each dial all subsets of combinations are equally likely to be chosen to be the combinations for that dial that contribute to an open-door state, and the probability of contributing to an open-door state is, for each dial, independent of any other dial, then the expected complexity of the enumeration algorithm is bounded by a linear function of n.

Our results show that an arbitrary enumeration can solve the safe dial problem quickly on the average because a preponderance of the problem instances of a given size are solved on the first few guesses.

The difficulty of particular problem instances is, however, sometimes implicitly judged by the number of steps a random hypothesis generator would require to solve the particular instance. Our results do not imply that *each* instance of the safe dial problem is easy in this sense. If a random generator produces only hypotheses that are members of U, then the expected number of steps, and the number of steps required to have any given confidence in finding a word in G, can be calculated from a binomial distribution in which the probability of success on any trial is given by (G/U). Thus if there is a single setting of a safe dial that opens the vault, for a random hypothesis generator it is intuitively a very hard problem. In contrast, expected complexity measures locate the probability in the problem instances, not in the hypothesis generators. Our results mean this: If there are a vast collection of vault doors each with a single dial, and if for any two *sets* of combinations there are as many doors that open with any combination in the one set as with any combination in the other set, then if you select vault doors at random from the collection, on the average no more than 2 guesses will be needed to find a combination that unlocks the vault door.

A more realistic application is to recent work by Ackley (1987). Ackley studied the behavior of a number of algorithms designed to determine any maximum value of real valued functions defined on the space, B^n, of n-place bit vectors. B^n is the space of all strings of length n on the vocabulary {0,1}. Since B^n is finite, every real valued function on this space has a maximum. Ackley's algorithms included familiar hill climbing procedures and a procedure Ackley called Stochastic Iterated Genetic Hillclimbing that integrates connectionist and genetic algorithms (Ackley (1985), Holland (1975)). The procedures were allowed to ask for the value of the target function for any element of B^n. Comparison of the performance of the various algorithms was based on the number of queries of this kind the procedure made before finding the maximum value. The procedures were not required to recognize that a value *was* the maximum (that is, in the terminology of learning theory (Osherson and Weinstein (1986)) they were not self-monitoring), nor were they required to continue to output the maximum value once it was found. His study applied seven algorithms to each of six functions for various values of n. The functions were chosen to differentiate properties of alternative hill climbing procedures.

With three qualifications, Ackley's arrangements form a set-predicate problem of the kind we have described. G is the set elements of B^n for which the target function assumes its maximal value; U(n) is B^n. The measure of complexity is exactly as in Theorem 1. The qualifications are that that there is no test for membership in G, that in this application U(n) is all strings exactly of length n, not all strings of length n or less, and, finally, Ackley's algorithms are not exclusively deterministic.

If we put a uniform probability distribution on the (nonempty) subsets of G, we can ask for the expected number of queries that any enumeration algorithm will make before finding a member of G. The answer is given by Theorem 1. The difference in U(n) affects only the value of S{n) in the proof of that theorem, which is irrelevant to the result.

3. Alternative Probability Distributions

These results quite naturally raise questions about the expected complexity of problem solving when it is not assumed that all instances of a given size are equally probable. We will show that even when non-uniform probability distributions are assumed there are still algorithms with constant expected complexity that solve set-predicate problems.

Suppose that not all instances of size n are equiprobable, but that all instances of the size n having the same cardinality of G are equiprobable, and each subset of instances of size n containing all and only instances with the same cardinality of G is as probable as any other such subset. Then the probability that a randomly selected instance of size n will have any specified value for the cardinality of G is 1/S(n), where S(n) is the cardinality of U.

Given these probability distributions over instances, there is a problem solver whose expected number of oracle calls is bounded above by 2, for all instance sizes. Since the problem solver with this property is stochastic we must first introduce some definitions.

A *stochastic problem solver* is a stochastic procedure that given X and a size bound stochastically determines at each step which word to query the oracle about, and either runs forever or conjectures a word after making queries to the oracle. A stochastic problem solver *solves a problem instance* iff with probability 1 it conjectures a word in G and then halts. A stochastic problem solver *solves a problem* iff it solves every instance of a problem.

We measure the complexity of a stochastic problem solver on a problem instance of a given size by the *expected* number of calls made to the oracle by the problem solver.

We consider a Bernoulli trial algorithm that determines its queries to the oracle by sampling randomly with replacement from a population in which there is a uniform distribution on *IP.*

We note the following well-known result:

If independent Bernoulli trials are performed, each with probability p of success, and X is the random variable representing the number of trials until the first success, then

$$\text{prob(X-k)-p(1-p)}^{M}$$

and

$$\text{Exp(X)-1/p}$$

and

$$\text{Exp}(X^2)\text{«(1-p)/p}^2$$

The Bernoulli trial algorithm will, for each instance of size n, have an expected number of oracle calls determined entirely by the cardinality of G, namely S(n) divided by the cardinality of G. The cardinality of G can range from 1 to n. Let g denote the cardinality of G. The probability of success on any given trial is the probability that some randomly chosen word x in 1° is in G.

$$P(x \varepsilon G \mid |G| = g) = \frac{g}{S(n)}$$

$$\text{FQctG)} \underset{g=1}{\overset{Sin)}{=}} \text{X J-CredlGl-dW-,)} \underset{g=1}{\overset{S(n)}{}} - \frac{\text{£}}{S(n)} \pm \text{^}$$

$$\frac{\sum_{g=1}^{S(n)} g}{S(n)^2} - " \frac{\frac{S(n)(S(n)+1)}{2}}{S(n)^2} = \frac{\text{SfrQ} + 1}{2S(n)}$$

$$\text{ExpQC)}_{\text{vk}\,\}} = \frac{\text{- L}}{P(xtG)} = \frac{2S(n)}{S(n)+1}$$

Hence the expected number of calls to the oracle is bounded above by 2 for all n.

Suppose again that we do not assume that all problem instances of the same size are equally probable. Suppose instead it is known that the cardinality of G is always a fixed (non-zero) proportion of the cardinality of U, but otherwise make no assumption about the probability distribution on instances of a given size. Then there is a p > 0 such that for any value of the size measure, all problem instances (with non-zero probability of occurring) have 10Op % of their members in G. Consider again a problem solving procedure that randomly selects (with replacement) a word from U and conjectures it, until a word in G is found. Whatever the instance considered, application of the procedure produces a sequence of independent Bernoulli trials each with probability p of success, and so it follows from the results just cited that the expected number of steps in the procedure is bounded by 1/p for all n, and that the variance of the number of steps required is $(1-p)/p^2$. Clearly this is not the "best" problem solving algorithm, and it could be improved by sampling from U *without* replacement. Equally clearly, the result can provide an upper bound on the expected complexity in cases in which it is cnly known that for every instance the ratio of the cardinality of G to the cardinality of U is *at least* p.

On the other hand, suppose the cardinality of G has a fixed upper bound. In that case the algorithm that samples with replacement can for any instance of any size still be viewed as a sequence of independent Bernoulli trials, but p, the probability of success on any trial, approaches zero as the size measure, n, increases without bound. Thus the expected complexity function is in these cases unbounded for this algorithm, and indeed one expects for any enumeration algorithm.

One can challenge the convention that defines a distinct probability measure for each instance size but no single probability measure over the set of all instances. There are many ways to define a measure on the set of all instances of a problem solving task. For example, many artificial intelligence system designers seem to believe that in the environments in which their systems operate, small problem instances are much more probable than large problem instances. One can put measures on all of the problem instances over an alphabet Z that satisfy this restriction; for example, one can give the set of all instances of size n the probability $(i/2)^n$, and divide the probability for the class of instances of size n equally among the instances of that size. With this measure the expected number of oracle calls for a deterministic enumeration algorithm diverges. On the other hand, if one imposes a geometric distribution, as Anderson suggests, the expected number of oracle calls required of a deterministic enumeration algorithm is finite.

4. The Complexity of the Finite Automaton Formulation of Problem Solving

Following Newell and Simon, most problem solving research assumes problem instances to have more structure than the set-predicate formulation presents. It assumes that the world can assume a set of discrete states describable by a finite collection of unary predicates. The application of an operator may (or may not) change the state of the world, but an operator always gives the same resulting state from the same prior state. Some state is designated as the *initial* or *starting* state. Other states, possibly including the initial state, are designated as *goal* or *accepting* states. So construed, a problem instance is determined by a finite state automaton whose number of states is known to the problem solver, but whose graph and set of accepting states are not known. The task is to halt after producing an accepting path in the automaton, i.e., a path from the initial state to an accepting state. The problem solver is not told which states are accepting states, so to recognize when to halt, it must query the oracle. Some of the standard illustrations of problem solving such as safe cracking have this form. The state of the combination lock as the cracker finds it is the initial state, and any state that opens the door is an accepting state. The opening of the door or its failure to open counts as the response from the oracle. Theorem proving, natural language parsing, and many other interesting tasks do not fit into the finite automaton framework since the number of possible world states is infinite, and hence cannot be described with a finite set of unary predicates.

Whenever problems can be represented by finite state automata whose nodes are state descriptions, the number of predicates, n, required for a state description determines the number of automaton states, 2^n, and we are guaranteed that if there is an accepting path, i.e., a word in G, there is one containing no more than 2^n-1 operators. Hence when the automaton assumption is made, it is natural to measure the size of a problem instance by the number of states of the problem instance. This size bound does *not* correspond to the size bound we considered above, for the following reason. Not every set of words containing words of at most length n-1 letters over an alphabet S is exactly the set accepted by some finite state automaton with n states. The smallest automaton accepting all and only the words in a set G of words, all of length n or shorter, has a set of states that may be exponential in n. Thus the bank vault door problem, for example, has in the previous measure a size given by the maximum number of dial twists required to open the door, and all problem instances-all sets of door opening values-with the same maximum number of dial twists have the same size. But if the size of a problem instance is measured by the number of states of the smallest automaton that accepts all and only the words in G for that instance, then different specifications of the combinations of dial settings that open the bank vault will give problem instances of different sizes, and some of these sizes will be much larger than the maximum number of dial twists required to open the safe.

If it is assumed that the problem instances are all generated by finite automata, then in many cases it is quite natural to take the number of states of the automaton to measure the size of the problem. In that case we can put a uniform measure on all labeled finite automata with a common initial state, arc labels from an alphabet I, and having n states.

What is the expected complexity of problem solving with the Newell-Simon enumeration algorithm for problem instances generated by finite state automata?

Theorem 3: The expected complexity function of the Newell-Simon enumeration algorithm for the finite state automaton problem over any nonempty alphabet, £, has a finite bound determined by the cardinality of I.

The proof is given in the Appendix. The bound in this case may be much larger than 2, but in typical cases is not -astronomical." In Theorem 3 all automata count, including those with inaccesible states; in many cases this is realistic, since the states of the automaton are descriptions of *logically possible* situations, and some logically possible situations are nonetheless impossible in the sense that no actions will bring about their realization. We conjecture, however, that an analogue of Theorem 3 holds even if one counts only automata all of whose states are accessible.

5. Conclusion

In this paper, we have examined a class of problems which could not be effectively or stochastically solved without queries to an oracle. We have measured the complexity of an algorithm by the number of queries it made to an oracle. However, results similar to those we have described can be produced for other problems and other measures of complexity.

For example, consider the class of problems which require finding an accepting path through a finite state automaton, but in which the accepting states are known. Some of the standard illustrations in the problem solving literature, such as Christians and Cannibals or Towers of Hanoi are problems of this kind. If such a problem has a solution, then it is trivial that there are algorithms that solve the problem with 0 queries to an oracle. Some other measure of complexity is needed. One way to solve these problems is through a -testing" enumeration algorithm that rather than using oracle calls to determine the accepting states, uses an effective test of some kind. One natural measure of the complexity of a "testing" enumeration algorithm is the number of tests it makes. For example, the Towers of Hanoi problem consists of three pegs A, B, and C, and n rings of varying size, initially stacked on peg A in order of decreasing size. The problem is to move all of the rings one at a time from A onto B (using C as temporary storage) in such a way that no ring is ever placed on a smaller ring. An accepting state can be effectively tested for, since it is simply one in which all of the rings are stacked on peg B. (Of course, what counts as a "testing" enumeration algorithm, and what counts as a test in such an algorithm cannot always be determined, but is perfectly clear in some particular cases.) Under this measure of complexity, a proof parallel to the one given for Theorem 3 shows that there are "testing" enumeration algorithms that have a finite upper bound determined by the cardinality of Σ, regardless of the order of enumeration and the instance size.

Similarly, it is natural to consider "testing" enumeration algorithms that solve set-predicate problems in which an effective test for membership in G can be constructed. A proof analogous to Theorem 1 shows that there are "testing" enumeration algorithms for which the expected number of tests performed is bounded above by 2, regardless of the order of enumeration and the instance size.

Our results also parallel, for a setting of psychological interest, results about expected complexity that have been obtained independently for NP complete problems. A celebrated NP complete problem concerns the determination of the simultaneous satisfiability of a set of clauses. Goldberg (1979) has shown that an algorithm for this problem, the Davis-Putnam Procedure, requires a number of steps which with probability approaching one is approximately quadratic in the number of clauses, provided a reasonably natural probability distribution is assumed for the instances of any size. More recently, Franco and Pauli (1983) showed that a procedure that simply enumerates truth assignments and checks whether they satisfy a set of clauses has, with probability approaching 1, a constant time bound if the same probability distribution for instances is used. They also show that under a different distribution, the Davis-Putnam procedure will with probability 1 require a number of computational steps that is exponential in the instance size. Hofri (1987) concludes from these results that

> The moral is that we need robust procedures. Or that we need to know how to tailor an algorithm to a given situation. (p. 4)

Two other conclusions may be broached. One is that "stupid" enumeration procedures, which are often compared unfavorably to "intelligent" heuristic procedures, are not always so stupid. If the computational tractability of a procedure is any indication of intelligence, then enumeration sometimes does quite well. These expected complexity results complement other work that shows, for example, that in a worst case the "version space" learning heuristic requires more computational resources than does simple enumeration (see Mitchell, (1982) and Kelly, (1988)). Algorithms that seem "clever," as do the Davis-Putnam procedure for satisfiability, or version space learning algorithms, or various problem solving heuristic procedures, may in various circumstances be less efficient and less powerful than enumeration algorithms that seem "inane."

A second conclusion is that whether a problem solving or learning situation requires recourse to heuristic procedures depends on what is known, or believed, about the situation, and in a variety of cases knowledge about the situation can straightforwardly be turned into complexity analyses that are directly relevant to the design of intelligent systems, or that show the computational admissability of various hypotheses about how natural intelligent systems are able to do what they do. The demonstration that a problem solving procedure is reliable and efficient on a few selected instances of a problem is no evidence at all that it is more reliable or more efficient than other procedures on other instances of that same problem. Judgements of the optimality or even "satisficity" of a procedure require some knowledge of the distribution of instances that procedure will meet.

The analyses given here are rather simple, and assume that the probability of occurrence of instances of any kind is unaffected by the evidence obtained in attempting to solve a particular problem instance. A more thoroughly Bayesian analysis would relax this assumption and interpret the probability distribution over instances of a given size as degrees of belief to be changed by forming a sequence of conditional probability distributions as evidence accrues. Indeed, the sorts of problems considered here are rather close to the problem originally treated by Reverend Bayes.[3]

[3]We thank Jay Kadane, Teddy Seidenfeld and Bas Van Fraassen for very helpful suggestions.

References

D. Ackley, "A Connectionist Genetic Algorithm," in J. Grefenstette, ed., *Proceedings of an International Conference on Genetic Algorithms and Their Applications,* 121-135. Carnegie-Mellon University, Pittsburgh, Pa., 1985.

D. Ackley, *Stochastic Iterated Genetic Hillclimbing,* Ph.D Thesis, Carnegie Mellon University, Department of Computer Science, 1987.

J. Holland, *Adaptation in Natural and Artificial Systems,* University of Michigan Press, 1975.

D. Angluin, "A Note on the Number of Queries Required to Identify Regular Languages,* *Information and Control,* 51:76-87, 1981.

A. Newell and H. Simon, *Human Problem Solving,* Prentice-Hall, 1972.

J. Anderson, *Language, Memory, and Thought,* Eribaum, 1976.

H. Wilf, *Algorithms and Complexity,* Prentice-Hall, 1986.

K. Kelly, "Version Spaces, Structural Descriptions and NP-Completeness," Technical Report, Laboratory for Computational Linguistics, Carnegie Mellon University, 1988.

T. Mitchell, "Generalization as Search," *Artificial Intelligence,* 18, 1982, 203-226.

A. Goldberg, -Average Case Complexity of the Satisfiability Problem" *Proceedings of the Fourth Workshop on Automatic Deduction,* Austin, Texas, 1979.

J. Franco and M. Paull, "Probabalistic Analysis of the Davis-Putnam Proceddure for Solving the Satisfiability Problem), Discr. Appl. Math., 5, 77-87 (1983).

M. Hofri, *Probabilistic Analysis of Algorithms,* Springer-Verlag, 1987.

D. Osherson, M. Stob and S. Weinstein, *Systems That Learn,* M.I.T. Press, 1986.

Appendix

Consider the following problem. Let £ be a finite alphabet with cardinality s > 1. Let n > 0, and let S be an arbitrary subset of W - $U_n L^n$, the set of all strings from the alphabet of length y, 0 ≤ y < n. You are given n, and an oracle for S. The oracle will respond "yes" to the question "Is *n* in S?" if *n* is in S, "no" if *n* is not in S, and will make no other responses. After asking as many questions as you please of the oracle, you are to output a string in S if there is one, and the words "no solution" if there is not one. Call the triple <X, n, S> a *set predicate problem instance.* A *set predicate problem* is the set of all set predicate problem instances with a common alphabet. A *problem solver* is a procedure that, given £ and n, makes a finite number of queries to the oracle and outputs a string. The sequence of queries, and the output of the procedure, must be a function of X, n, and the sequence of oracle responses. A problem solver succeeds on a problem instance <Z,n,S> if it ouputs a string in S if there is one when the input to the procedure is <S,n> and the oracle is for S, and outputs "no solution" if S is empty. A problem solver succeeds on a problem if it succeeds on every instance of the problem.

There are some easy observations to be made about the computational difficulty of this problem. We will measure complexity by the number of calls made to the oracle.

Proposition 1: There is a procedure that solves the set predicate problem with at most $I_{k\ =\ 1}^{k=n}$ s^k queries.

Proof:. The cardinality of W is $I_{k\ „\ 1}^{kssf1}$ s^k. Consider the following simple enumeration procedure where JI is an enumeration of $£^n$ such that no longer string occurs before any shorter string, and strings of the same length are in lexical order:

```
SIMP(n):
begin
        until either the oracle answers "yes" concerning string K
            or each string in ụ. has been queried
        do
            query concerning the first unexamined string in ụ. and
            if the oracle answers "yes" for string n then output this
            string;
        if the oracle answers "no" for the last string in ụ, then output
        "no solution"
end.
```

Clearly this algorithm expends k queries iff the first string in the target language is in position k of µ,. So in the worst case the entire sequence is traversed and $I_{k\ m\ 1}^{k=n}$ s^k queries are made. QED.

It is easy to see that the worst case upper bound on the set predicate problem given by SIMP cannot be improved.

Proposition 2: For each n, in the worst case any solution to the set predicate problem makes at least $X_{k=}$ -$j^{k=n}$ s^k queries.

Proof: Suppose that for some n > m, learner f can solve each instance of size n with fewer than $X_{k\,m}$ -$|^{k=sn}$ s^k queries. Then if the target language is the empty language f will stop after some k < 1^ _ -$j^{k=n}$ s^k queries. Let ***n*** be some string not queried by f. Then when the target language is *{n},* the sequence of queries made by f will be the same as for the empty language, and the series of responses by the oracle will be the same as for the empty language, and f will halt after k queries with the same output as for the empty language. Hence f will not succeed on {A}, which is a contradiction. QED.

Clearly a result completely analogous to Proposition 2 holds if the set of strings is required to be regular.If an n state automaton accepts a string, it accepts a string of length no longer than n-1. So suppose we are to guess a string in a regular set given the number of states in an acceptor of that set. We know we needn't check any strings longer than n to find one if it exists. Viewed this way, our upper bound on the set predicate problem is an upper bound for finding a string in a regular set using an oracle.

In applications, the number of states, n, in the finite state acceptor of a regular target language is usually some quantity exponential in another large parameter. For example, if automaton states are constructed as state descriptions in a language with p monadic predicates, then n = 2^P, so the overall number of queries by SIMP is in the worst case $X_{k\,m}$ -j^{k}"2(p) $_s$k^ where 2(p) denotes 2^P.

Theorem 1: For all I, the expected complexity of the set predicate problem is bounded by 2.

Proof: Let P be a uniform probability measure on the power set of W. That is, for each subset S of W, P assigns {S} probability $(1/2)^{r}$, where r = $Z_{kss1}^{k!sn}$ s^{k}. Recall that SIMP asks k questions of the oracle when the first string in ụ, that is in the target language occurs in position k in JI. Therefore the probability that SIMP uses k queries is the probability that the kth string to occur in ụ, is the first string in p. that is a member of the target language. In other words, the probability P(k) that SIMP uses k queries is the sum of all P(S) such that S is a subset of W and string *n* is in S, and no string prior to *n* in ụ, occurs in S. Each subset S of W that contains *n* but no K^{1} prior to *n* in ụ. can be

expressed as the union *{n}* U S', where S' is in the power set of K = W-faV = *%* or *%'* is prior to *%* in.u,}. Since *%* is in position k, the cardinality of K is $Z_{k\,m\,1}^{k\sim\bar{n}}$ s^{k} - k = r - k. Hence there are 2($^{r"k}$) subsets of W that contain *n* but do not contain any *it'* prior to *n* in JI. Hence P(k) = $2^{(r-k)}/2^{r} = 2^{-k}$.

The expected number E(n) of queries of SIMP for all instances of size n and alphabet I is the sum from one to $I_{k\,\_\,1}^{ks=n}$ s^{k} = r of k times P(k). So E(n) - $£k{=}1^{k=r}$ (^)• ^{Tn}ė ^{infin}'tė series $I_{k=1}$~ $(k/2^{k})$ is greater than E(n), for every n, and converges to 2, since $I_{kss1}^{\circ\circ}$ $k/2^{k}$ - 1/2 +

$\Sigma_{k=1}^{\infty}(k+1)/2^{(k+1)}$ = 1/2 + $I_{k=1}^{\circ\circ}k/2(^{k+1})$ + $Zk.!^{00\wedge\wedge*1})$ = 1/2 + $1/2I_{k=1}^{\circ\circ}$ $k/2^{k}$ + 1/2, and hence $1/2I_{k=1}^{\circ\circ}$ $k/2^{k}$ = 1. QED.

Theorem 2: The expected number of letters in queries in any set predicate problem is bounded by 10.

Proof: Let Pki be the probability that the first word in the enumeration SIMP that is in the target language is the ith word of length k. s is the cardinality of £. There are ik letters in the first i words of length k and there are $I_{u=i}{}^{u=\wedge"1}\wedge s^u u$ letters in all strings of length less than k. Therefore $[ik + I_{u\_i}{}^{Us=}(^{k*1} \wedge s^u u]$ gives the number of letters in the enumeration u, of SIMP(n) up to and including letters in the ith word of length k. There are s^k (denoted by s(k) in upper limits of summation) words of length k. Hence the expected number of letters in queries by SIMP(n) is

(1) $E(n) = I_{ks=1}{}^{n} I_{i=1}{}^{i=s}<^{k}>$ Pki $[ik + I_{u=1}{}^{L'}=(^{k}-^{1})s^u u]$.

Reasoning as in the proof of Theorem 1, there are 2^r equiprobable subsets of W. The number of such sets containing the ith word of length k > 0 and no preceeding word in JI is $2t^{r"}('^{+l}-'^{\wedge})$ (where U - $X_{u\_}i^{u=k"1}s^u$, and we understand that if k=1, U is zero), because the ith word of length k is the (i+U)th word in JI. Hence Pki =$1/2f^{i+u}3$ and substituting in (1) we have:

(2) $E(n) = I_{k=1}{}^{kssn} I_{i=1}{}^{i=s(k)}[ik+\Sigma_{u=1}{}^{u=k-1}s^u u]/2^{[i+U]}$.

The right hand side of (2) is the sum of two terms:

$T1 = \Sigma_{k=1}{}^{k=n}\Sigma_{i=1}{}^{i=s(k)}\ ik/2^{[i+U]}$

$T2 = I_{k=1}{}^{k=n} I_{i=1}{}^{i=s}\wedge[I_{u=1}{}^{u=k*1}s^u u]/2 0^{+}U]$

We first derive a bound on T1 for all n.

$T1 = I_{k=1}{}^{k=n} I_{i=1}{}^{i=s}(^{k})\ ik/2t^{i+U}I = I_{k=1}{}^{kssn}(I_{ias1}{}^{i!=S(k)\ I,2i})\wedge 2^U < I_{k=1}{}^{k=n} 2k/2^U$,

where the inequality follows from the argument of Theorem 1. Rearranging the right hand side of the inequality we have Tt < 2 I$_{k=1}$$^{k=n}$ k/2^{u}- Now for

all s > 1 and all k, U = I$_{U=1}$^{Uas}($^{k"1}$)s^{u} is greater than or equal to k, and hence

2^{U} is greater than or equal to 2^{k}. Therefore:

T1 < 2I$_{k=1}$$^{k=sn}$ k/2^{U} S 2I$_{kss1}$$^{k=n}$ k/2^{k} < 2X$_{k=1}$~ k/2^{k} = 4.

Finally, we derive a bound on T2. We define Uu - I$_{u=1}$^{Uss}^^{k,1}^s^{u}u. With that abbreviation:

T2 = I$_{ksa1}$$^{k=sn}$i:$_{i=1}$$^{issS(k)}$ Uu/2t$^{'+U}$) = I$_{k=1}$$^{k=n}$(X$_{iss1}$$^{i8=s(k}$> 1/2') Uu/2^{U} <

I$_{k=1}$$^{k=n}$Uu/2^{U}

Now U^{2} ≥ Z$_{u-1}$$^{Uss(k"1}$> s^{2u} > Z$_{u-1}$$^{u=}$(^{k,1}) s^{u}u = Uu because s^{u} > u for all values of u. Hence T2 < £$_{kss1}$$^{k=n}$Uu/2^{u} < I$_{kss}$-|$^{k=n}$ U^{2}/2^{U}- The quantity U = X$_{USB1}$^{u}=(^{k*1}) s^{u} is a distinct number for every value of k. Therefore T2 <

$\Sigma_{j=1}^{\infty} j^2/2^j = 1/2 + \Sigma_{j=1}^{\infty} (j+1)^2/2^{(j+1)}$.

Expanding the righthand side of the second equality we obtain

Zj.i°° J^{2}/2^{j} = (1/2)1^^j2$_{/2}$0*+1) +$_{2}$ £$_{jV|}$~ J/20+^{1}) + Zj$_{aBi}$$^{\infty}$1/2$^{(j+1)}$

+ 1/2. Now Zj.!^{00} J^{2}/2C)$^{+1}$) (1/2) Ij$_{=1}$°°j^{2}/2i; 2 Zj$_{_1}$^{oe}j/20"$^{+1}$) -

2j.i°°i/2i = 2; and I$_{J-=1}$^{OC}1/20'$^{+1}$) = 1/2. Hence Ij$_{=1}$°° j^{2}/2J = 1/2 + (1/2)

£,vTJ$^{2/2j}$ +2 +1/2 and therefore T2< Ij$_{=1}$°°j^{2}/2J =6.QED.

Take an automaton problem instance to be quadruple <Z, L, A, n> where L is the set of strings from alphabet I accepted by A, and A has n states. Assume as before an oracle for L, and understand automaton problem and automaton problem solver by analogy with the set predicate case. The number of states, n, of A will measure the size of any problem instance, and a problem solver is given I, n and of course the responses of the oracle to any queries the problem solver makes. As in the set predicate case, we can investigate bounds on the average, over all problem instances of size n for a fixed alphabet, of the number of queries a problem solver, and SIMP(n) in particular, makes before ouputting a word in L if there is one, or "no solution" if L is empty. Since in applications the automaton states are usually given as state descriptions in a monadic predicate language, we take the states of the automaton to be labeled, and distinguish otherwise isomorphic automata with distinct labelings.

Counting finite automata meeting conditions on the existence of an accepting path of a specified length but having no shorter path is difficult. The following theorem avoids the difficulty by finding easily counted properties that bound the expected number of queries for automaton problem instances of a given size.

Theorem 3: For every finite I there exists a number N such that for all n, the expected number of queries for the SIMP algorithm over all instances of automata problems based on I and of size n is less than N.

Proof: We assume that the initial state is fixed. Any subset of the n labeled states may be chosen to be the set of accepting states, so there are 2^n specifications of accepting states. Each of the n states has s (the cardinality of *J.)* arcs out of it, and for each arc there are n choices of sink. So there are

nxn...xn x n x n...x n x..................x nxnx...n
[s factors] [s factors].......................[s factors}
{...............................n factors..}

or $(n^s)^n$ graphs. Hence there are $2^n n^{ns}$ automata for Σ having n states. (If the initial state is not fixed, the number is $2^n n^{(ns+1)}$, which would make no significant difference in the argument that follows.)

It is easy to show that the number of finite automata with n states accepting word i of length l(i) on a path without circuits is $2^{(n-1)}n^{(ns-l(i)}$. For let p be the sequence of arcs whose labels in sequence form the string i. The terminus of p must be an accepting state. Any subset of the remaining n-1 states may be accepting, hence there are $2^{(n-1)}$ choices of sets of accepting states. All of the arcs out of any state not touched by p may be freely chosen, and so may the arcs out of the terminal node of p. There are l(i) + 1 states touched by p. Hence there are $(n^s)^{(n-l(i))}$ choices for the collections of arcs. All but one of the arcs out of each of the states touched by p (save the terminal state of p) may be chosen freely. There are $n^{(s-1)l(i)}$ such choices. Hence there are $n^{(s-1)l(i)}2^{(n-1)}n^{s(n-l(i))} = 2^{(n-1)}n^{(ns-l(i))}$ automata. that accept word i without circuits. If a word is accepted by a path with a circuit then the word cannot be the shortest accepted by the automaton. Thus the number of automata accepting a given word by a path without a circuit is at least as large as the number of automata for which that word is the first word in the enumeration μ that is accepted. Hence the probability, Pi, that i is the first word of μ accepted by a randomly chosen n state automaton is less than $2^{(n-1)}n^{(ns-l(i))}/2^n n^{ns} = 1/(2n^{l(i)})$.

Now the expected number of queries for instances of size n, E(n), satisfies

(1) $E(n) < \Sigma_{i=1}{}^{i=r}\, i/2n^{l(i)}$ where $r = \Sigma_{j=1}{}^{j=(n-1)}\, s^j$.

Note that

(2) $\Sigma_{i=m}{}^{i=(m+k)} \leq (k+m+mk+k^2/2)$.

The expression on the right hand side of (1) can be expressed as a sum of "chunks":

$\Sigma_{l=1}^{l(i)=(n-1)}[1/(2n^{l(i)}] \Sigma_{j=L(i)}^{j=M(i)} j$

where L(i) abbreviates the number of words in the enumeration of length less than l(i), and M(i) abbreviates $s^{l(i)}$. Applying (2) to this expression we obtain

(3) $E(n) < \Sigma_{l(i)=1}^{l(i)=(n-1)}[1/(2n^{l(i)})] [(s^{l(i)} - 1) + L(i) +1 + (L(i)+1)(s^{l(i)}-1) + (s^{l(i)}-1)^2/2]$.

On simplifying (3) by multiplying through, the largest terms obtained are $L(i)s^{l(i)}/(2n^{l(i)})$ and $s^{2l(i)}/(2n^{l(i)})$. We show by induction that for all $s > 1$ and all $l(i) > 0$, $L(i)s^{l(i)}/(2n^{l(i)}) < s^{2l(i)}/(2n^{l(i)})$ because $s^{l(i)} > L(i)$. The base case is immediate since $1 > 0$. Assume for $l(i) = k$ that $L(i) = \Sigma_{j=1}^{j=(k-1)}s^j < s^k$. We must show that $\Sigma_{j=1}^{j=(k}s^j < s^{(k+1)}$. Now $\Sigma_{j=1}^{j=k}s^j = \Sigma_{j=1}^{j=(k-1)} s^j + s^k < s^k + s^k$ by the assumption. Hence $\Sigma_{j=1}^{j=k}s^j < 2s^k \leq s^{(k+1)}$, which proves the claim.

Returning to (3), we know that the largest term on the right hand side is $\Sigma_{l(i)=1}^{l(i)=(n-1)} (s^2/n)^{l(i)}$. For every $n > s^2$, $\Sigma_{l(i)=1}^{l(i)=(n-1)} (s^2/n)^{l(i)} < \Sigma_{k=1}^{\infty} (s^2/n)^k$. Since $s^2/n < 1$ for $n > s^2$, this series converges with some limit R. For every $n \leq s^2$, $\Sigma_{l(i)=1}^{l(i)=(n-1)}(s^2/n)^{l(i)}$ is finite, and since s is fixed, there is a finite set of such quantities. Let Q be the largest of them. Then for every n, $\Sigma_{l(i)=1}^{l(i)=(n-1)}(s^2/n)^{l(i)} \leq \max\{R,Q\}$. So there is a bound on the largest term in the right hand side of (3), for all n. Hence there is a K such that for all n, $E(n) < K\max\{R,Q\}$. QED.