HEURISTIC SEARCH VS. EXHAUSTIVE SEARCH

Laurent Siklossy and Vesko Marinov
Department of Computer Sciences
University of Texas at Austin
Austin, Texas U.S.A.

ABSTRACT

A theorem proving system embodying a systematic search procedure is described. Although the search spaces are usually infinite, and not even locally finite, the asymmetric way in which they are generated results in a speed that is estimated to be one to two orders of magnitude faster than the theorem provers of Quinlan and Hunt, and Chang and the problem-solver of Fikes, to which this system has been compared extensively.

Descriptive terms: search, heuristic search, exhaustive search, search space, local infinite search space, rewriting rule, expansion rule, theorem proving, problem-solving.

1. INTRODUCTION

Heuristic programming techniques have been developed to make it possible to solve problems whose solutions each constitute just one point or a small set of points in a very large, and possibly infinite, space, the search space. Many problems in artificial intelligence are difficult, and require "intelligence" from the problem solver, precisely because of the size of the search space. Although heuristic search does not guarantee a solution, the size of the search space precludes an alternative technique such as exhaustive search. Any method that would attempt to search unselectively the whole search space for a solution would quickly exhaust avaliable resources of time and memory and would be bound to failure, except in some rare, "lucky" cases.

The statements in the above paragraph seem so "obvious" that few would disagree with them. The present study shows that, in fact, they are false in regards to several artificial intelligence systems that have been built. First, the search space is very small in many problems, often less than even a few dozen nodes. Second, although the search spaces may be infinite in principle, in reality only very small parts of this space need to be generated, in a systematic way, until the solution is found. Finally, a simple program, embodying a systematic, blind and "dumb" (?) procedure, was written. This program has solved over sixty problems that are typically considered as requiring heuristic search. The vast majority of these problems have a potentially infinite search space. An estimate of the relative speeds of different computers and programming languages indicates that our program is from one to two orders of magnitude faster, on the average, than the most powerful problem solvers with which it has been compared.

The reader should not conclude, either now or after reading the additional evidence that we present, that heuristic programming has no value. The success of our program, however, has important methodological and practical consequences. Exhaustive search appears as a derided, yet powerful technique to solve many problems that are very hard for humans and that seemed to necessitate heuristics for their solution. In a powerful problem-solver, an exhaustive search component might have an important function in solving efficiently large classes of problems, although it certainly cannot solve all problems.

II. PROBLEM DOMAIN

Points in the problem space are expressions defined recursively as variables (atoms) or zero-, one- and two-place function symbols (operators) having expressions as arguments. Constants are zero-place functions. New points in the problem space can be obtained by applying a rewrite rule to an expression that is already in the problem space. A rewrite rule is of the form "<left-hand-side> :* <right-hand-side>" where both sides are expressions. A rewrite rule can be applied to an expression if the expression, or one of its subexpressions, is a substitution instance of the left-hand-side of the rewrite rule. We say that the rewrite rule operates on the (sub)expression. The (sub)expression that is operated on is transformed by applying the same substitution to the right-hand-side of the rewrite rule. For example, if the binary operator + is used with the variables A, B and C; and if the expression (A+B)+C is in the problem space, a single application of the rewrite rule A+B* = B+A can produce two new expressions: C+(A+B) and (B+A)+C, depending on whether the rule is applied at the top level or not. The rules are seen as having one input and producing one output. This is not general, and in resolution, for example, rewrite rules have two inputs for one output.

A theorem is in the form of a rewrite rule, for example (A+B)+C:=A+(B+C). A proof of the theorem consists in showing that, by a succession of applications of rewrite rules, the left-hand side can be transformed into the right-hand side. After having been proved, theorems can be considered as rewrite rules.

This problem domain is identical to that considered extensively by Quinlan* (7) and Quinlan and Hunt** (8). Q&H show results of their learning programs, FDS1 and FDS 2, on 59 theorems grouped in 5 different areas (see Tables I through V). Q includes much additional information on the programs' performance. FDS is considered the most powerful theorem-prover of its type. Ernst and Newell (1) consider the results of QMF "quite impressive." (p. 27.) The same version of our program has proved all 59 theorems in the tables in Q&H.

Chang (2) has also proved some of the theorems in QMF using a resolution-based system, and a comparison of his results with ours is given. Resolution is best viewed as a system utilizing a single two-input, one-output rewrite rule. Finally, Fikes (4) has documented proofs of four heuristic search problems. Although stated in a programming language, these four problems in Fikes can be easily rewritten as problems in a

*To be abbreviated Q.
**To be abbreviated Q&H.

rewrite rule formulation. Two of them were solved by the same program that proved the theorems in Q&H. The addition of some numerical capabilities to our program would allow it to solve the other two problems. (See Siklossy and Marinov (10) for a hand-simulation and additional information on our system.) Some of the problems that can be easily expressed for Chang's system are awkward to express in our system of rewrite rules. Our system lacks the constraint manipulation techniques and numerical capabilities of Fikes' REFARF. On the other hand, it is doubtful that either Chang's or Fikes' system could prove all the theorems in Table III- In this article, their systerns are compared on problems on an intersection of their respective domains of expertise with the domain of application of our system.

III. SYSTEMATIC SEARCH

A. Non-Expansion Rules

A theorem is of the form L:=R, where L and R are the left and right expressions of the theorem. A systematic search for the proof of the theorem starts with the search space consisting of the expression L. From L, we build the next leve1 of the search tree- To this effect, we apply to L all rewrite rules, one after the other. An expression resulting from the application of a rewrite rule to a node of the search tree is inserted in the tree if and only if it does not already figure in the search tree. Figure 1 shows the search tree for the proof of theorem 1 in Table 1, using only the first two rewrite rules. The third node of the search tree, (B+A)-K;, is called a dead node since no new expressions can be obtained by applying rewrite rules RI or R2 to this node. The search terminates when the expression A+(B+C) is found at the fifth level of the tree, as the 12th node.

B. Expansion Rules

The reader will notice that rule R4, A:= (A+B)-B, in Table I could also be applied to try to prove theorem 1. Any expression or subexpression is a substitution instance of the left-hand-side of R4. Thus, any (sub)expression can be rewritten, using RA, by first adding to it any expression, then subtracting the same expression that had been added. There are clearly an infinity of expressions that can be substituted for B, so that the search space is locally infinite and, consequently, some recent results in heuristic search by Pohl (6) on locally finite search spaces are not applicable to the problem domain considered here.

We call expansion rules those rules (such as R4 in Table 1, or R6 in Table IV) whose repeated application could generate an infinite search space for the problem domain considered. Some ordered generation of the space is needed, especially for those rules (such as R4 in Tablei) which allow the introduction of arbitrary expressions.

Since expansion rules play an important role in our system they deserve a more detailed discussion. The rules are a sample of the different types of expansion rules in the domains that we have considered:

1.	A:=(A+B)-B	R4 in Tables I and II.
2.	A/C:=(A/B)/C	R16 in Table III.
3.	0:=A-A	T4 in Table II.*
4.	A:=A+0	T1 in Table II.
5.	P:= P	R6 in Table IV.
6.	A>B:=(A-B)>0	R1 in Table V.

The first three rules generate a locally infinite search space, while the other three generate an infinite space, which is locally finite. For example, applying TI, Table II to the node X results in the node X+0. Another appl1cation of the same rewrite rule produces the three nodes (X+0)+0, (X+0)+0 and X+(0+0).

We further distinguish between two types of expansion rules. Atomic expansion rules have a single variable as the left-hand-side. Rules 1, 4 and 5 are atomic. The other expansion rules are called non-atomic. The search strategy is slightly different for the two types of expansion rules (see below)

It is undecidable in general whether a rewrite rule system has an infinite search space. However, in the problem domains that we considered, a rule R is an expansion rule if it can be applied to its own right-hand-side (i.e. if the right-hand-side is a substitution instance of the left-hand-side) and if the expression obtained by applying R to its own right-hand-side has more nodes (when considered as a tree) than the right-hand-side of R. It is seen that testing whether a rule is an expansion rule is a simple matter in our problem domains.

Our systematic search procedure proceeds as follows- first non-expansion rules are applied to the left-hand side of the theorem to be proved. If the right-hand side of the theorem is found, the system terminates. Otherwise, all nodes of the search tree are eventually dead, since non-expansion rules, by definition, cannot generate an infinite search space. At that point, nodes of the search tree are generated in leve1 order (9). For each node so generated, we apply all expansion rules *(as* will be described below), adding any new nodes to the search tree. We then shift to uslng on1y non-expansion rules again. It is seen that expansion rules are called in onlv when all else has failed; and when they are ca1led in, they are applied as sparsely as possible. The reason for this is simply that expansion rules tend to increase drastically the size of the search space. It is interesting to notice the similarity of our asymmetrie use of rewrite rules to the asymmetry in the use of the two classes of axioms in the geometry theorem machine (5). In this latter system, axioms of class (ii) are used only when axioms of class *(\)* have failed to prove a theorem

An example of the use of a 1oca1ly finite expansion rule (R6 in Table IV) is given in Figure 2. After the application of RI to the left-hand side of theorem 1, Table IV, both nodes of the search tree are dead. The (unique) expansion rule R6 is applied to the first node

*V0 is a constant

of the tree, namely (S«*rR).R, producing nodes 3,4 and 5. We switch back to non-expanBion rules only, and the answer is quickly found at node 9. Notice that nodes 2,6 and 7 are dead.

The situation is more complex for a locally infinite expansion rule, which introduces arbitrary expressions, as, for example, R4 in Tables I and II. We have chosen to allow for such expressions only those that can be constructed legally from the atoms encountered in the theoien to be proved, which is no real restriction. It should be noted that here is the only case where the right-hand side of the theorem is used for purposes other than just checking for termination. Otherwise, the search is Indeed blind!

Our strategy is to grow the search space as slowly as possible. We attain our goal by limiting both the expressions to which atomic expansion rules are applied, and the expressions that can be substituted for the arbitrary variables in locally infinite expansion rules. The growth of the search space takes place in several passes, and in each successive pass there are fewer limitations on the use of expansion rules.

In a first pass, atomic expansion rules (which could be applied to any (sub)expression of a node) are applied only to variables, constants and unary functions. There is no restriction on the (sub)expreBsions that non-atomic expansion rules are allowed to match. During that same first pass, we limit the expressions that will be substituted for the arbitrary variables in locally infinite rules. (If a certain expression is substituted for the arbitrary variable, wc say that we are expanding with that expression.) In the first pass, locally infinite expansion rules expand only with variables, constants and unary functions.

If the theorem is not proved during the first pass, the restrictions described above would be successively loosened. We did not need to program passes beyond the first since all theorems could be proved during the first pass. Moreover, we are now testing more sophisticated techniques for expansion that render much of the above scheme inefficient.

One might argue that we are using heuristics. We believe though that we are only using a strategy, since the order in which the rules are applied is predetermined and not modified during proofs.

IV. COMPARISONS WITH OTHER SYSTEMS

A. Comparisons with Quinlan and Hunt

The theorem proverj of Q&H (there are two versions of FDS) were written for a FORTRAN compiler on the IBM 7094. Our system is written for a LISP 1.5 interpreter on the CDC 6600. Comparisons between the speeds of the two systems are awkward. A consensus of "experts" indicates that we should expect our system to be about ten times* slower than Q&H. (It was estimated that the FORTRAN compiler is about 100 times faster than our LISP interpreter, while the CDC 6600 may be some eight to ten times faster than the IBM 7094.) To facilitate comparisons, all our times (in seconds) were divided by 10 in the results in Tables I through VI. All rewrite rules, labelled Ri, were taken in exactly the same order as given in Q&H, and all theorems, labelled Ti, proved exactly in the same order. All the proofs mentioned in this article were obtained with the same version of our program. By contrast, the results mentioned by Q&H are for different, locally optimized programs. The only exception is in Table V: the rewrite rules in Q&H are insufficient to prove the theorems. Q&H indicate that they used additional rewrite rules, without specifying which. We added the four rewrite rules R12 through R15. Because of these changes, the results of Table V are not as meaningful as those of the first four tables. (In the tables below, S&M label our results.)

REWRITING RULES

R1	A+B:=B+A
R2	A+(B+C):=(A+B)+C
R3	(A+B)-B:=A
R4	A:=(A+B)-B
R5	(A-B)+C:=(A+C)-B
R6	(A+B)-C:=(A-C)+B

THEOREMS		TIMES (Seconds)		Nodes to Solution
		Q&H	S&M/10	by S&M
T1	(A+B)+C:=A+(B+C)	5	0.062	11
T2	(A-B)+B:=A	0	0.021	4
T3	A:=(A-B)+B	0	0.027	5
T4	A+(B-C):=(A+B)-C	1	0.020	3
T5	(A-B)+C:=A+(C-B)	5	0.046	5
T6	(A-C)-(B-C):=A-B	215	1.799	87
T7	(A+C)-(B+C):=A-B	169	6.165	351
T8	A+(B-C):=(A-C)+B	7	0.037	5
T9	(A+B)-C:=A+(B-C)	4	0.046	5
T10	(A-B)-C:=(A-C)-B	8	1.422	67
T11	A-(B+C):=(A-B)-C	8	2.560	146
T12	A+(B-C):=A-(C-B)	6	6.052	321
T13	A-(B-C):=A+(C-B)	7	1.372	84
T14	(A-B)+C:=A-(B-C)	5	0.090	7
T15	A-(B-C):=(A-B)+C	4	0.036	4
T16	A-(B-C):=(A+C)-B	6	0.033	4
T17	(A-B)-C:=A-(B+C)	7	2.069	145
T18	(A+B)-C:=A-(C-B)	1	0.079	6
Average		25.444	1.219	

TABLE I

*It is not crucial that the reader agree with our estimate.

REWRITING RULES

R1	A+B:=B+A
R2	A+(B+C):=(A+B)+C
R3	(A+B)-B:=A
R4	A:=(A+B)-B
R5	(A-B)+C:=(A+C)-B
R6	(A+B)-C:=(A-C)+B
R7	(A+B)+C:= A+(B+C)
R8	(A-B)+B:=A
R9	A:=(A-B)+B
R10	A+(B-C):=(A+B)-C
R11	(A-B)+C:=A+(C-B)
R12	(A-C)-(B-C):=A-B
R13	(A+C)-(B+C):=A-B
R14	A+(B-C):=(A-C)+B
R15	(A+B)-C:=A+(B-C)
R16	(A-B)-C:=(A-C)-B
R17	A-(B+C):=(A-B)-C
R18	A+(B-C):=A-(C-B)
R19	A-(B-C):=A+(C-B)
R20	(A-B)+C:=A-(B-C)
R21	A-(B-C):=(A-B)+C
R22	A-(B-C):=(A+C)-B
R23	(A-B)-C:=A-(B+C)
R24	(A+B)-C:=A-(C-B)
R25	A+0:=A
R26	A-0:=A
R27	NEG(A):=0-A
R28	0-A:=NEG(A)

THEOREMS		TIMES (Seconds) Q&H	S&M/10	Nodes to Solution by S&M
T1	A:=A+0	5	0.112	11
T2	A:=A-0	8	0.112	11
T3	A-A:=0	21	2.046	58
T4	0:=A-A	6	0.256	13
T5	A+NEG(A):=0	5	0.124	8
T6	0:=A+NEG(A)	58	1.095	44
T7	A+NEG(B):=A-B	5	0.201	10
T8	A-NEG(B):=A+B	6	0.124	6
T9	NEG(A)+NEG(B):=NEG(A+B)	125	2.136	50
T10	NEG(A)-NEG(B):=NEG(A-B)	47	1.712	51
T11	NEG(NEG(A)):=A	105	0.489	13
Average		35.545	0.764	

TABLE II

REWRITING RULES

R1	(A+B)+C:=A+(B+C)
R2	A+(B+C):=(A+B)+C
R3	I+A:=A
R4	A:=I+A
R5	A+I:=A
R6	A:=A+I
R7	Z+A:=Z
R8	A+Z:=Z
R9	Z:=A+Z
R10	Z:=Z+A
R11	I/I:=I
R12	I:=I/I
R13	(A+C)/(B+C):=(A/B)+C
R14	(A/B)+C:=(A+C)/(B+C)
R15	(A/B)/C:=A/C
R16	A/C:=(A/B)/C
R17	A/(B/C):=A/C
R18	A/C:=A/(B/C)
R19	SIG(A):=(A+SIG(A))/I
R20	(A+SIG(A))/I:=SIG(A)
R21	SIG(A/B):=SIG(A)
R22	SIG(A):=SIG(A/B)
R23	SIG(A)+B/C:=SIG(A)+C
R24	SIG(A)+C:=SIG(A)+B/C
R25	SIG(I):=Z/I
R26	Z/I:=SIG(I)

THEOREMS		TIMES (Seconds) Q&H	S&M/10	Nodes to Solution by S&M
T1	A/A:=A	1	0.560	17
T2	A:=A/A	2	0.323	11
T3	((A/B)+C)/D:=(A+C)/D	2	0.058	2
T4	A/((B/C)+D):=A/(C+D)	2	0.061	2
T5	(A+B)/C:=((A/D)+B)/C	10	12.830	192
T6	A/(B+C):=A/((D/B)+C)	21	12.714	193
T7	SIG(Z):=Z/I	1	0.186	7
T8	Z/I:=SIG(Z)	1	0.410	14
T9	SIG(A)/I:=SIG(A)	13	0.352	7
T10	SIG(A):=SIG(A)/I	11	4.457	73
T11	SIG(A)+SIG(B):=SIG(A)	36	0.713	14
T12	SIG(A):=SIG(A)+SIG(B)	596	4.832	74
T13	SIG(A)/SIG(B):=SIG(A)	2	0.724	13
T14	SIG(A):=SIG(A)/SIG(B)	1334	13.885	198
T15	SIG(SIG(A)):=SIG(A)	108	0.957	18
Average		142.667	3.680	

TABLE III

REWRITING RULES

R1 P·Q:=Q·P
R2 P·(Q·R):=(P·Q)·R
R3 P·(P→Q):=Q
R4 ⌐Q·(P→Q):= P
R5 ⌐⌐P:=P
R6 P:=⌐⌐P

THEOREMS	TIMES(Seconds)		Nodes to Solution
	Q&H	S&M/10	by S&M
T1 (S→⌐R)·R:= S	1	0.085	8
T2 B·(⌐A→⌐B):=A	1	0.034	3
T3 ⌐B·(A→B)·(⌐A→C):=C	2	0.090	7
T4 ⌐C·(B→C)·(⌐B→⌐⌐A):=A	2	0.184	12
T5 (P→⌐Q)·Q·(⌐P→(R·S):=R·S	6	0.171	11
Average	2.400	0.113	

TABLE IV

REWRITING RULES

R1 A>B:=(A-B)>0
R2 (A-B)>0:=A>B
R3 A>0:=0>NEG(A)
R4 0>NEG(A):=A>0
R5 A IS REAL:=(A>0) | (A=0) | (NEG(A)>0)
R6 (A IS REAL)·(B IS REAL):=(A-B) IS REAL
R7 (A IS REAL)·(B IS REAL):=(A+B) IS REAL
R8 (A-B)=0:=A=B
R9 A=B:=(A-B)=0
R10 (A>B) (B>0):=(A+B)>0
R11 (A>0) (B>0):=(A*B)>0
R12 NEG(A):=0-A
R13 0-A:=NEG(A)
R14 A-B:=NEG(B-A)
R15 NEG(B-A):=B-A

THEOREMS	TIMES(Seconds)		Nodes to Solution
	Q&H	S&M/10	by S&M
T1 0 A:=NEG(A)>0	10	0.049	3
T2 NEG(A)>0:=0>A	10	0.055	6
T3 0 (A-B):=B>A	55	0.137	5
T4 A B:=0>(B-A)	163	0.072	4
T5 (A-B)>0:=0>(B-A)	32	0.061	4
T6 0 (B-A):=(A-B)>0	50	0.055	5
T7 NEG(A-B)>0:=B>A	15	0.136	10
T8 B>A:=NEG(A-B)>0	11	0.078	4
T9 (A IS REAL)·(B IS REAL):= ((A-B)>0) | ((A-B)=0) | (NEG(A-B)>0)	10	0.250	14
T10 (A IS REAL)·(B IS REAL):= (A>B) | (A=B) | (B>A)	143	8.112	179
Average	49.900	0.901	

TABLE V

Actually, the eomparison between Q&I and our system is misleading. Not only are their results obtained with two different versions of FDS, but their results for <u>each</u> Table arc obtained after opttimizat ion of the average t ime to prove the theorems of a particular table. In other words, the particular version of FDS that proves most efficiently the theorems of Table III performs miserably when proving the theorems of Table I. In Table VI, we list, when ava i lable, the bes t and worst average times to prove theorems in the various tables (as collected from (7)) as wo 11 as the results in Q&H and our own.

A proof terminates when the (right-hand side of a) theorem is found. The level at which the theorem is encountered is fixed, but the number of nodes searched at that level can vary if we change the order in which the rewrite rules are applied. 1t would be straightforward to implement a sys tem tha t would change the order of application of rewrite rules to minimize the average time per proof in a given table. We cannot estimate the gains in speed that would result. As we mentioned previously, we have kept the order of the rules and t lie or oms that we found in Q&H.

	Best in Q	Worst in Q	Q&H	S&M/10
Table I	25(FDS2)	136(FDS2)	25	1.22
Table 1I	36(FDS2)	1221(FDS1)	36	0.76
Table III	137(FDS1)	600(FDS1)	143	3.68

TABLE IV Average times (in s) per theorem.

B. Compari son with F ikes

Fikes (4) describes four heuristic search problems: monkey and bananas; water]ug; a programming problem; and the mi ssionaries and cann i-bals . The sta tements of these problems are not reproduced here. These problems in *V*Ikes are written in a programming language, but each of them can be translated easily into a rewrite rule system. Such a translation for the flrst problem is given in Figure 3. In each triple, the first element Indicates the monkey's position the second element the monkey's height and the last element the position of the box. Each of the rewrite rules corresponds to an a 1lowable ac tivity of the monkey, in exac tly the same order as found in Fikes. The translation of the programming problem is shown in Figure 4. The sextuple represents the two regi sters put side by side. The rewrite rules correspond to the allowed shi ft operations.

Fikes[1] system was programmed on an IBM 360/67 for an IPL-V interpreter. His computer may be slightly s1 owe r than the CDC 6600, wh11e we expec t 1 PL to have a speed comparable to our LISP. The real times to solution, also listed in Figures 3 and 4, are therefore comparable as given.

The water jug problem and the missionaries and cannibals problem necessi tate some ari thmetic capabilities that our system lacks. Both problems

can be easily translated into a system for rewrite rules (see (10)). For example, in the waterjug problem the rewrite rule corresponding to pouring from the large jug into the small Jug would be:

(a,b) := (a=-min(5,a+b), b'=b-(a'-a))

where the first and second element of the ordered pair represent the amounts of water in the small and large jugs, respectively.

The missionaries and cannibals problem has a total search space of 16 nodes (1) while our hand simulation (10) shows that our system would solve the waterjug problem in a total of seven nodes. We would estimate our system to use less than 2 seconds for each of these problems. These (hypothesized) times should be compared to 491 seconds and 83 seconds for Fikes' REF-ARF. Moreover, the particular way in which the missionaries and cannibals problem is stated in REF has a strong influence on the time needed to solve it Fikes mentions that a particular statement of the problem would require more than 1800 seconds of 360/67 time.

C. Comparisons with Chang.

Chang (2) reports on a very efficient theorem prover based on the resolution principle. He has proved some of the theorems from Q&H. Table VII compares his times with ours. Chang's program is written in LISP 1.6 for the FDP-10. We do not know whether a compiled or interpreted version of LISP 1.6 was used.

REWRITE RULES

6 Rules as in Table I.

THEOREMS		TIMES (Seconds)	
		Chang	S&M
T1	(A+B)+C:=A+(B+C)	24.55	0.632
T2	(A-B)+C:=A+(C-B)	9.37	0.330
T3	A+(B-C):=(A-C)+B	6.92	0.335
T4	(A+B)-C:=A+(B-C)	7.32	0.275
Average		12.04	0.353

TABLE VII. Comparative Results of Chang and S&M.

The results described in this section could be improved substantially, especially those of the long proofs. A newly generated node is added to the search space only if it is not already in the space. At present, the space is searched linearly for the existence of the node. A hash-coding scheme would cut down dramatically on the time needed for such a search.

In spite of such improvements, our system would perform poorly in potentially large search spaces which could be reduced significantly by the manipulation of constraints.* We have reported results on problems for which few or no constraints can be used.

*Fikes' system has solved several constraint satisfaction problems.

V. CONCLUSIONS

The research reported here is part of a larger, continuing investigation into the cost of overhead in a theorem prover. It appears intuitive (but it may not be true) that, as a theorem prover applies more sophisticated techniques to reduce its search space, it will expend more time for computation per node considered. The total number of nodes considered is an inadequate measure of the efficiency of a theorem prover, since this number does not take into account the cost per node. Total time to solution still appears as the only reasonable criterion for comparing different systems with similar domains.

It may come as a surprise to many that our systematic, blind, search procedure performs at least one to two orders of magnitude faster than the most powerful existent theorem provers to which it has been compared. The result is even more surprising when it is stressed that the problem domains considered have infinite search spaces that are often not even locally finite The results can be explained by noticing that the proofs of the theorems are very short. The deepest proofs in Q&H (tied by the theorem in Figure 1) have a depth of only 5. Many of the proofs in Q&H, and all four problems in Fikes, are found after generating at most a few dozen nodes.* Consequently, the problems considered should not tax the capabilities of a sophisticated heuristic program (although they certainly seem to have done so.[1]). These problems do not represent adequate tests for such a program.

An informal investigation into the problems solved by artificial intelligence systems using heuristic search indicates that many of the tasks have, in fact, very small search spaces. If these problems are representative of many of the future problems to be solved by these systems, then it is apparent that a very powerful problem solver can be built by combining heuristic and systematic search procedures that would time-share available resources. The systematic search procedure could solve very rapidly a large number of the easy problems. The cooperation, inside a large system, of subsystems of various degrees of sophistication will be a fascinating area of research.

VI. REFERENCES

(1) Amarel, S. On Representations of Problems of Reasoning about Actions, in: Michie, D. (Ed.), Machine Intelligence 3. American Elsevier, N. Y., 1968, 131-171.

(2) Chang, C.L. The Unit Proof and the Input Proof in Theorem Proving, J.ACM. 17, 4, 1970, 698-707.

(3) Ernst, G.W. and Newell, A. GPS: a Case Study in Generality and Problem Solving. Academic Press, N.Y., 1969.

(4) Fikes, R.E. REF-ARF: A System for Solving Problems Stated as Procedures, Artificial Intelligence, i, 1970, 27-120.

*A more thorough discussion of the size of search spaces will be reported subsequently.

(5) Gilmore, P.C. An Examination of Che Geometry Theorem Machine, Artificial Intelligence, J_, 3, 1970, 171-187.
(6) Pohl, I. Heuristic Search Viewed as Path Finding in a Graph, Artificial Intelligence. J., 3, 1970, 193-204.
(7) Quinlan, J.R. FORTRAN Deductive System: Experiments with two Implementations, Technical Report -68-5-03, Computer Science Group, University of Washington, Seattle, 1968.
(8) Quinlan, J.R. and Hunt, E.B. A Formal Deductive Problem-Solving System, J.ACM, 15, 4, 1968, 625-646.
(9) Salton, G. Manipulation of trees in information retrieval, C.A.C.M. , 5_, 2, 103-114, 1962.
(10) Siklossy, L. and Marinov, V. Experiments in search, Technical Report TSN-20, Computation Center, University of Texas at Austin, 1971.

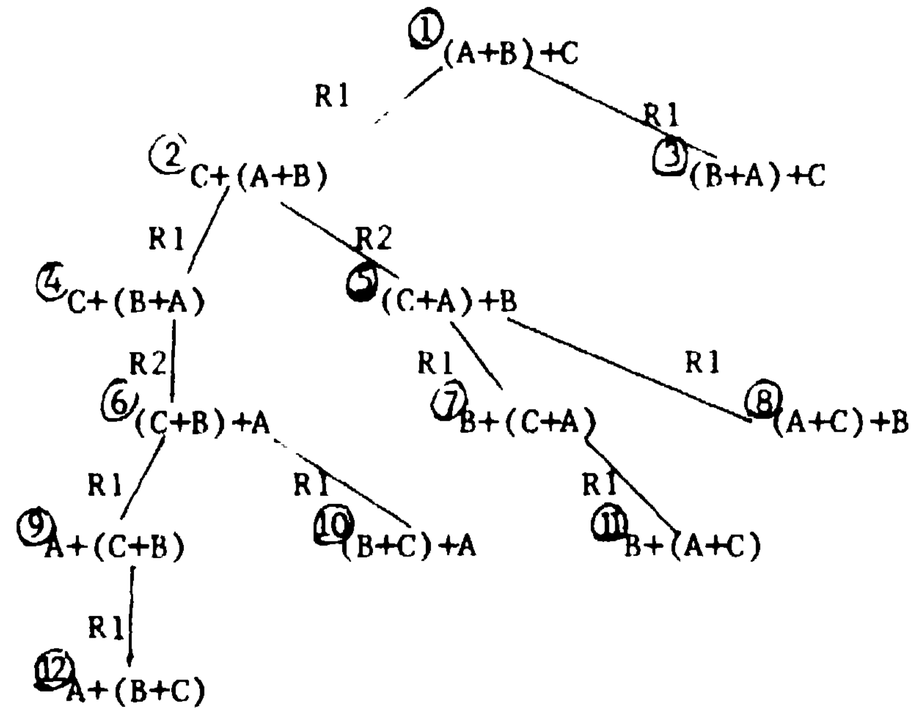

FIGURE 1. Proof of (A+B)+C:=A+(B+C) in TABLE I.

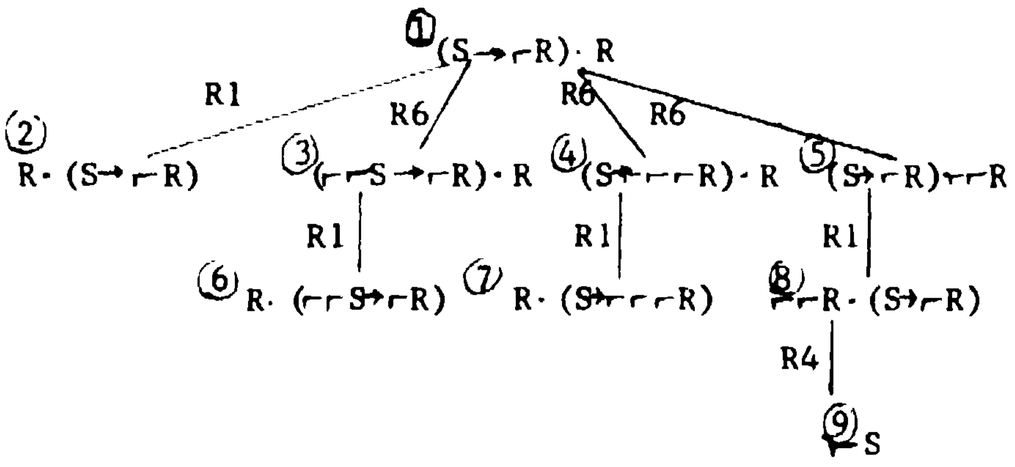

FIGURE 2. Proof of (S→⌐R)·R:=⌐⌐S in Table IV.

```
R1   (A, onfloor, B):=(C, onfloor, B)
R2   (A, onfloor, A):=(C, onfloor, C)
R3   (A, onfloor, A):=(A, onbox, A)
R4   (A, onbox, A):=(A, onfloor, A)
R5   (underbananas, onbox, underbananas):=
       (withbananas, onbox, underbananas)

Variables: A,B,C. Range: X1,X2, underbananas.

T1   (X1, onfloor, X2):=
       (withbananas, onbox, underbananas)

Times for proof: Fikes: 36s;  S&M 2.027s.
```

FIGURE 3. Monkey and Bananas Problem.

```
R1   (a,b,c,d,e,f):=(0,a,b,d,e,f)
R2   (a,b,c,d,e,f):=(b,c,0,d,e,f)
R3   (a,b,c,d,e,f):=(0,a,b,c,d,e)
R4   (a,b,c,d,e,f):=(b,c,d,e,f,0)
R5   (a,b,c,d,e,f):=(a,b,c,a,b,c)

T1   (2,3,4,0,0,0):=(2,3,0,0,0,0)

Times for proof: Fikes 143s; S&M 0.931s.
```

FIGURE 4. Programming Problem

1. One of the investigators (VM) was supported by the National Institute of Health Grant GM15769-04.