The Nature of Heuristics

Douglas B. Lenat*
Computer Science Department, Stanford University, Stanford, CA 94305, U.S.A.

Recommended by Pat Hayes

ABSTRACT

Builders of expert rule-based systems attribute the impressive performance of their programs to the corpus of knowledge they embody: a large network of facts to provide breadth of scope, and a large array of informal judgmental rules (heuristics) which guide the system toward plausible paths to follow and away from implausible ones. Yet what is the nature of heuristics? What is the source of their power? How do they originate and evolve? By examining two case studies, the AM *and* EURISKO *programs, we are led to some tentative hypotheses: Heuristics are compiled hindsight, and draw their power from the various kinds of regularity and continuity in the world; they arise through specialization, generalization, and—surprisingly often—analogy. Forty years ago, Polya introduced Heuretics as a separable field worthy of study. Today, we are finally able to carry out the kind of computation-intensive experiments which make such study possible.*

1. Overview

The impressive performance of expert knowledge-based systems [1, 5, 8] leads us to consider anew the field of *Heuretics*: the study of the informal, judgmental 'rules of thumb' which underlie such programs. To understand the successes of the expert systems, and perhaps ultimately to improve such results. Heuretics asks *What is the source of power of heuristics*? Similarly, with an eye toward understanding, facilitating, and perhaps ultimately automating the construction of expert systems, Heuretics asks *How do new heuristics originate*? Experiments with the AM and EURISKO programs provide some initial answers, and some concrete methodological suggestions about how to go about getting better answers.

*The author is an assistant professor of Computer Science at Stanford University, a member of that university's Heuristic Programming Project, and a consultant for CIS at XEROX PARC.

Artificial Intelligence **19** (1982) 189–249

1.1. What is the source of power of heuristics?

By examining the situations in which heuristics *fail* (in Section 2.3 and 3.1), we are led (in Section 3.2) to hypothesize that the underlying source of heuristics' power is a kind of two-dimensional *continuity*. If a heuristic H was (or would have been) useful in situation S, then it is likely that heuristics similar to H will be useful in situations similar to S. In other words, if we could somehow actually compute APPROPRIATENESS(Action,Situation), that function would be continuous in both variables, and would vary very slowly.

One useful exercise (Section 3.3) is to consider the graph of APPROPRIATENESS values for a fixed action, varying over the situations in which it might be applied. The language of graphs of functions is then at our disposal, an attractive metaphor within which to discuss such processes as specializing a heuristic, using multiple heuristics, and measuring attributes of a heuristic's performance.

Of course the world isn't so accomodating. There are many possible measures of APPROPRIATENESS (efficiency, low down-side risk, comprehensibility), and many dimensions along which Situations can vary (difficulty, time, importance, subject matter). Compounding this is the nonlinearity of the Situation space along most of these dimensions. Thus the 'zero-th order theory' espoused in the last two paragraphs is merely a metaphor.

Yet it is too attractive, too close to what human experts actually do, to reject out of hand. It can be extended into a "first order theory": It is frequently useful to *behave* as though the zero-th order theory were true, i.e., to behave as though APPROPRIATENESS(Action,Situation) exists and is continuous. To give an example: the current situation may appear similar to ones in which it was cost-effective to skip to the Conclusions Section of the paper; even though you can't be sure that's an appropriate action to take now, it may be useful for you to behave as though the world is that continuous, to take that action anyway. If you do so, you're following a heuristic. That heuristic guidance is only as good as the generalization process you used in deciding the situation was similar (e.g., would you apply it to all articles?; to all articles written by X?).

As the world changes, a heuristic which was valid and useful may become invalid. Perhaps X's writing style has improved. In the extreme case of a rapidly changing environment, the mean useful lifetime of a heuristic may be too small to make it worth relying on. Consider, for example, the prices of stocks on the New York exchange, or the locations of individual molecules in a gas in a flask. There, *continuity* is not at issue, but *volatility* is.

There is a difference between these last two examples though; we can record the stock prices, but it is impossible to record the positions of all the molecules in the flask of vapor.

We thus have three considerations—continuity, stability, and observability—determining which domains may adequately be modelled as heuristic searches.

Observability: If data cannot be gathered, heuristics cannot be formed and evaluated.

Continuity: If the environment changes abruptly, the heuristics may never be valid.

Stability: If the changes are continuous but rapid, the heuristics may have too short a lifetime before becoming useless—or worse than useless.

At the present time, the most constraining of these requirements is *observability*. Very few fields admit automatic data acquisition. One might build a program which proposed promising new experiments to perform in molecular biology, but it is beyond the capabilities of present technology to automate the carrying out of those experiments to see the results. The most observable fields are those which can be completely formalized within the machine: mathematics, programming, and games. But the behavior of a running program can also be recorded and inspected by the program, in particular a program which employs heuristics might monitor its own performance; therefore, we may add *Heuretics* to that list of observable fields. EURISKO (Section 4) is such a program, and from it we have begun to learn more about Heuretics.

1.2. How do new heuristics originate?

Empirical results from AM (Section 2) suggest that new heuristics arise from three sources:

Specialization of existing, more general heuristics. This often has the form of adapting, binding, matching a template to observed data, producing a more specialized, more efficient offspring. *Compiling* and *structured programming* are two computer science analogues of this process. A second way in which specialization occurs is when an exception to a general heuristic is noted, and a more specialized, higher-precedence heuristic is formulated. *Debugging* and *type-checking* are the computer science analogues of such accommodation.

Generalization of existing, more specialized heuristics. An extreme—but common—form of this is *abstraction* from observed data. In such a case, the heuristic is a prediction about APPROPRIATENESS(Action,Situation) for a whole domain of situations and actions, based on having actually seen one or more elements of that domain. Other types of generalization are also useful: Often, a powerful new theorem or technique will be proven for some domain D; it may then be a useful *heuristic* to apply it outside D as well. For instance, the values of some infinite series were successfully guessed at by pretending they were differentiable; once the series' value is conjectured, *proving* it is made much simpler.

Analogy to existing heuristics and to past, successful acts of creating new heuristics. It is a remarkable thing that analogy works, a sign of an even deeper kind of continuity than was sought in 1.1. Even though two domains may appear disparate, analogous heuristics may be equally powerful in coping with them (e.g., *Look for examples of concept C before you try to prove any theorems about C*). Even if the heuristics for the two domains seem disparate, the paths which were followed in getting the powerful heuristics of the field may be similar (e.g., *Examine successful and unsuccessful attempts at finding proofs, and embody (in new heuristics) any features that discriminate between them*).

Which of these three is most efficacious? Under what circumstances are each of these three methods indicated or contraindicated? These are Heuretics questions, and best answered by performing experiments. Results from such experiments on AM and EURISKO are presented in Section 4, and surprisingly (to us) lead us to favor analogy over the other two methods.

1.3. Other Heuretics issues

One Heuretics question which will not be addressed herein is *What is the impact of an individual heuristic upon a search*? That issue has been well covered elsewhere, both qualitatively and quantitatively, by Michie, Nilsson, and others. See, for example, [6] and the references he cites.

Another issue given only brief consideration is *How should advice from several heuristics be combined*? Results from building expert systems lead to the conclusion that the details of the control structure are not crucial. As Feigenbaum is wont to say, "In the *knowledge* lies the power." One can view the heuristics as production rules, and then this issue becomes: What interpreter should run the rules? This problem can itself be recursively 'solved' by making the interpreter a production system, and so on *ad infinitum*, although when one tries to find such strategic rules there are few to be had, and even fewer of noticeable impact. See, for example, [4].

1.4. Heuretics as a field of knowledge

We spoke earlier of Heuretics as a *field of knowledge*. Polya championed the study of heuristics as a separate scientific discipline forty years ago, and traced its origins back to Liebnitz, Descartes, and even Pappus. A decade ago, Pospelov and Pushkin tried to define the field as "the science which studies the laws governing the design of new actions in new situations." To merit the designation of 'field of knowledge', Heuretics must comprise some more or less well agreed-upon objects of study, some motivation for studying such objects, some central questions about the nature of such objects, and some accepted methods for investigating those questions.

The *object of study* are of course heuristics. Our initial definition of a heuristic is: *a piece of knowledge capable of suggesting plausible actions to follow or implausible ones to avoid*. In Section 2.2 it becomes apparent that this is insufficient; for a body of heuristics to be effective (useful for guiding rather than merely for rationalizing in hindsight) each heuristic must specify a situation or context in which its actions are especially appropriate or inappropriate. In other words, heuristics must have both an if- and a then-part. The theory developed in Section 4 extends this definition: a heuristic is seen as a bundle of attributes (and corresponding values) which includes many kinds of conditions (if- parts), many kinds of actions (than-parts), and also several nonexecutable attributes such as its worth, origin, and average running time. Section 4.2 presents a principled method for automatically generating all the possible 'legal' attributes that a heuristic might possess. As of this writing, over one hundred distinct attributes of a heuristic have proven themselves useful (to the running of EURISKO). The objects of study (heuristics) are plentiful, complex, and interrelated; in short, there is a richness of structure to this field.

What is the *motivation* for studying heuristics? Two are detailed in Section 2.1: (1) The recent successes with heuristic rule based expert systems drive us to investigate their apparent source of power, heuristics. (2) One of the major bottlenecks in constructing such systems is extracting domain-dependent heuristics from human experts, and that could be partially automated by a program whose field of expertise is itself the formulation, discovery, extraction, modification, etc. of heuristics. In order to build such a program, a better understanding of heuristics is necessary.

We have already presented some of the *major heuretics questions*: what is the source of a heuristic's power? the origin of new heuristics? the quantitative impact of a heuristic on a search? the interactions between heuristics working toward the same ends? the useful attributes of a heuristic?

Finally, there must be a *methodology* for answering such questions, an accepted experimental procedure. This paper proposes to use the standard empirical inquiry paradigm which dominates AI research: test hypotheses about heuristics by constructing—and studying—computer programs such as AM and EURISKO, programs which use heuristics and which try to find new ones. For twenty years after Polya's plea to investigate Heuretics, we lacked the ability to automatically manipulate symbols with enough facility to construct large heuristic search programs. For the next twenty years, we lacked the representational know-how and, frankly, the necessary number of machine cycles, to carry out the second-order investigations: what happens as heuristics are added, changed, removed. As these impediments crumble, we can design concrete experiments, we can build a methodology for tacking the various Heuretics questions.

1.5. Heuristics about Heuristics

As with any field of human endeavor, Heuretics is accumulating a corpus of informal judgmental knowledge—heuristics about heuristics. These guide the heuretician in extracting heuristics from experts, in deciding when the existing corpus of heuristics needs to be augmented, in representing heuristics within knowledge bases, in evaluating the worth of a heuristic, in troubleshooting a program built around a large collection of heuristic rules, etc. Some examples are listed below.

(1) The expert, if asked initially to state his informal judgmental rules, usually either denies their existence or provides a very small fraction of them. How does the knowledge engineer typically overcome this block? Each domain object and operation mentioned by the expert probably has some heuristics peculiar to it. Each pair of domain entities may have one or two heuristics about such combinations. Therefore, one extraction technique is to present each domain object or operation, or pair of same, to the expert, and ask him/her to introspect on rules of thumb for dealing with that entity or combination of entities.[1] This technique is itself a heuristic about heuristics. Here is another one: It is rarely cost-effective to carry out that extraction procedure for the co-occurrence of all triples (or larger sets) of domain objects and operations.

(2) Creating new examples of a domain concept can be a straightforward process, but creating new instances of the use of a heuristic is often much more timeconsuming—each usage of a heuristic often demands the creation and investigation of many new domain concepts. The impact of having and using a domain concept for a while that later turns out to be a 'blind alley' is much less serious than having and using a bad heuristic for a while.

(3) If the representation (vocabulary) is well suited to the content of the heuristic, then it is possible for the heuristic to be concisely represented and efficiently used. In the extreme case, a heuristic might simply say " c_1Rc_2", as in "Children CanBeTrainedLike Chickens" and "Children LikeToEat Chickens". Such compaction obviously depends upon the proper relation R being defined. If the vocabulary of relations is large and well chosen, many heuristics can be represented tersely. In cases where one heuristic is used very frequently, or where several heuristics could all be compacted, it is worth defining one or more new relations R to shorten the heuristics.

Just as the study of computational linguistics had to be grounded in particular languages during its maturation, so Heuretics has had to remain grounded in particular task domains. Eventually the theory of formal grammars lifted itself above the details of any individual language, though specific grammars are still used to illustrate the various theorems and relationships. Analogously,

[1]To provide an argument for heuristic (1) above, it is worth mentioning that the author initially drew a blank when composing this subsection of the paper, specifically when trying to think of examples of heuristics for heuristics. The problem vanished after listing several precise roles that such heuristics could fulfill (at the end of the first paragraph of this subsection), and then considering each role in turn: introspecing on heuristics for extracting heuristics, heuristics for deciding whether to try to get new heuristics, heuristics for representing a heuristic in a program, etc. Since the development of EURISKO, 25 additional 'heuristics about heuristics' have been produced by hand, and EURISKO itself has synthesized over 300.

we aim toward eventually studying Heuretics in a domain-independent fashion, but of necessity must ground our examples—and our test programs—in particular domains.

2. AM: The Origin of New Concepts and Conjectures

This section of the paper is a brief detour, a demonstration that new domain facts and conjectures can be discovered by employing a body of heuristics for guidance. Sections 3 and 4 return to the 'main line' by respectively considering the two primary Heuretics questions: the source of a heuristic's power and the origin of new heuristics.

2.1. Motivation

Heuretics is important for both theoretical and practical reasons. As Zavalashina said in [15] "one of the basic conditions for further evolution of heuristic programming, for an increase in the range of problems with which it can deal, is investigation of the qualitative structure of heuristic activity, its 'informal' components". The subsequent successes of programs (e.g., see [1, 5]) built upon a large core of domain knowledge—both facts and heuristics—reinforce this argument for the importance of heuristic reasoning as a phenomenon worthy of study. Artificial Intelligence is constantly seeking and developing new 'power sources' to guide and constrain search; heuristics are one of the most ubiquitous and potent sources of power, and merit further study for that reason alone.

More pragmatically, one current bottleneck in constructing large expert systems is the problem of knowledge acquisition: extracting knowledge from a human expert and representing it for the program. The expert must communicate not merely the 'facts' of his field, but also the heuristics: the informal judgmental rules which guide him in rapid decision-making. These are rarely thought about concretely by the expert, and almost never appear in his field's journal articles, textbooks, or university courses. Techniques for automatically discovering domain knowledge could alleviate this extraction problem.

Can this be done? Since *knowledge* comprises both facts and heuristics, the question divides into two parts: can new domain concepts and relationships be discovered (addressed in Sections 2.2 and 2.3), and can new domain heuristics be discovered (addressed in Sections 3 and 4)?

Is automated knowledge acquisition cost effective? Consider the making of a *human* expert. Having him or her rediscover the knowledge of their field seems at first glance hardly the typical pedagogical practice. That's certainly true for the *facts* of the field, which are readily presented in texts. Yet practitioners of many fields become experts only after a period of apprenticeship to a master, a trying period during which they must induce the *heuristics* of their craft from examples. Witness the crucial role of the internship of medical doctors, counselors, artists, graduate students, and many others.

2.2. The process of discovery

"How was X discovered?" When confronted with such a question, the philosopher or scientist will often retreat behind the mystique of the all-seeing I's: Illumination, Intuition, and Incubation. A different approach would be to provide a rationalization, a scenario in which a researcher proceeds reasonably from one step to the next, and ultimately synthesizes the discovery X. In order for the scenario to be convincing, each step the researcher takes must be justified as a plausible one. Such justifications are provided by citing *heuristics*, more or less general rules of thumb, judgmental guides to what is and is not an appropriate action in some situation.

For example, consider heuristic $H1$, shown in Fig. 1. It says that if a function f takes a pair of A's as arguments, then it's often worth the time and energy to define $g(x) = f(x, x)$, that is, to see what happens when f's arguments coincide. If f is multiplication, this new function turns out to be squaring; if f is addition, g is doubling. If f is union or intersection, g is the identity function; if f is subtraction or exclusive-or, g is identically zero. Thus we see how two useful concepts (squaring, doubling) and four fundamental conjectures might be discovered by a researcher employing this simple heuristic. Application of $H1$ is not limited to mathematics of course; one can think of Compile(x,x) (i.e., optimizing compilers written inefficiently in the language they compile, and then processed by themselves); Kill(x,x) (i.e., suicide); Ponder(x,x) (i.e., self-awareness); and even Apply(x,x) (i.e., the activity we are now engaging in).

$H1$: **if** $f: A \times A \rightarrow B$,
then define $g: A \rightarrow B$ as $g(x) = f(x,x)$
$H2$: **if** $f: A \rightarrow B$, and there is some extremal subset b of B,
then define and study $f^{-1}(b)$

FIG. 1. Two heuristic rules which lead to useful concepts and conjectures.

Elsewhere [10], we describe the use for heuristic $H2$ (see Fig. 1), which says to investigate the inverse image of known extrema. If f is *Intersection*, $H2$ says it's worth considering pairs of sets which map into extremal kinds of sets. Well, what's an extremal kind of set? Perhaps we already know about extremely small sets, such as the empty set. Then the heuristic would cause us to define the relationship of two sets having empty intersection—i.e., disjointness. If f is *Employed-as*, then the above heuristic says it's worth defining, naming, and studying the group of people with no jobs (zero is an extremely small number of jobs to hold), the group of people who hold down more than one job (two is an extremely large number of jobs to hold). If f is *Divisors-of*, then the heuristic would suggest defining the set of numbers with no divisors, the set of numbers with one divisor, with two divisors, and with three divisors. The third of these four sets is the concept of prime numbers. Other heuristics might then cause us to gather data, to do that by dumping each number from 1 to 1000 into the appropriate set(s), to reject the first two sets as too small, to notice that every number in the fourth set is (surprisingly) a perfect square, to take their square roots, and finally to notice that they then coincide precisely with the third set of numbers. Now that we have the *definition* of primes, and we have found a surprising *conjecture* involving them, we shall say that we have *discovered* them. Note that we are nowhere near a proof of that conjecture.

Of course the above instances of discoveries are really just *reductions*. We can be said to have reduced the problem "How might Squaring be discovered?" to the somewhat simpler problem "How might Multiplication be discovered?" by citing $H1$. Similarly, we reduced the problem of discovering Primes to the problem of discovering Divisors-of by citing $H2$. Such reductions could be continued, reducing the discovery of Divisors-of to that of Multiplication, thence to Addition or Cartesian-product, and so forth. Eventually, we are down all the way to our conceptual primitives to concepts so basic that we feel it makes no sense to speak of discovering them (see Fig. 2).

```
             PRIMES
               |
               |
               V
          DIVISORS-OF
               |
               |
               V
             TIMES
              /\
             /  \
            /    \
           /      \
      PLUS         CARTESIAN PRODUCT
        |                 |
        V                 V
```

FIG. 2. Reducing each concept's discovery to that of a simpler one. Note that multiplication can be discovered if the researcher knows either addition of numbers or Cartesian products of sets.

Why, then, is the act of creation so cherished? If some significant discoveries are merely one or two 'heuristic applications' away from known concepts, why are even one-step discoveries worth communicating and getting excited about? The answer is that the discoverer is moving upwards in the tree, not downwards. He is not rationalizing, in hindsight, how a given discovery might have been made; rather, he is groping outward into the unknown for some new concept which seems to be useful or interesting. The downward, analytic search is much more constrained than the upward, synthetic one. Discoverers move upwards; colonizers (axiomatizers and pedagogues) move downwards. Even in

```
                 \   ↑   /     \  ↑  /
                  \  |  /       \ | /
                   \ | /         \|/
                  PRIMES  NUMBERS-WITH-MANY-DIVISORS
                  \   ↑   /
                   \  |H2 /H2      \ | /
                    \ |  /          \|/
                  DIVIS-OF  EXPONENTIATION
                 \    ↑   /  /
                  \   |H1 /H1 /
                   \  |  /  /
      PARTITION      TIMES  /
          ↑        /  ↑   /
       \  |H2  /H1    |  /
        \ |  /        | /
         PLUS      CARTESIAN PRODUCT
```

FIG. 3. The more explosive upward search for new concepts. Most heuristics apply in several situations, often in more than one way (such as $H2$ applying to Divisors-of). Some concepts (such as multiplication and exponentiation) are reached from several paths.

the limited situation depicted in Fig. 3, the researcher might apply the 'Repeat' heuristic to multiplication, and go off along the vector containing squaring, exponentiation, hyper-exponentiation, etc. Or he might apply $H2$ to Divisors-of in several ways, for example looking at numbers with very many divisors.

Once a discovery has been made, it is much easier to rationalize it in hindsight, to find some path downward from it to known concepts, than it was to make that discovery initially. That is the explanation of the phenomenon we've all experienced after working for a long time on a problem, the feeling "Why didn't I solve that sooner?" When the reporter is other than ourselves, the feeling is more like "I could have done that, that wasn't so difficult!" It is the phenomenon of wondering how a magic trick ever fooled us, after we're told how it was performed. It enables us to follow mathematical proofs with a false sense of confidence, being quite unable to prove similar theorems. It is the reason why we can use Polya's heuristics [14] to parse a discovery, to explain a plausible route to it, yet feel very little guidance from them when faced with a new problem and a blank piece of paper.

There still is that profusion of upward arrows to contend with. One of the triumphs of AI has been finding the means to muffle a combinatorial explosion of arrows: one must add some heuristic guidance criteria. That is, add some

$H3$: **if** the range of one operation has a large intersection with the domain of a second, and they both have high worth, and either there is a conjecture connecting them or the range of the second operation has a large intersection with the domain of the first,
then compose them and study the result.

$H4$: Compose two operations and study the result.

FIG. 4. Contingent heuristic rule and an explosive one.

additional knowledge to indicate which directions are expected to be the most promising ones to follow, in any situation. So by a *heuristic*, from now on, we shall mean a *contingent* piece of guiding knowledge, such as *H*3 in Fig. 4, rather than an unconstrained Polya-esque maxim like *H*4. The former is a heuristic, the latter is an explosive.

2.3. AM: A computer program that discovers mathematical concepts and conjectures

There is a partial theory of intelligence here, which claims that discovery can be adequately guided by a large collection of such heuristic rules. In particular, mathematical discovery may be so guided. To test this hypothesis, we designed and constructed AM, a LISP program whose task was to explore elementary finite set theory: gathering empirical data, noticing regularities in them, and defining new concepts, AM is described at length elsewhere [9], and a very brief recapitulation here should suffice.

AM began with 115 set theory concepts, including static structures (sets, bags, lists) and active operations (union, composition, canonize). For each concept, we supplied very little information besides its definition. Additionally, AM contained 243 heuristic rules for proposing plausible new concepts, for filling in data about concepts, and for evaluating concepts for 'interestingness'. Among them were *H*1, *H*2, and *H*3.

Each concept was represented as a frame-like data structure, using the property list feature of LISP. Fig. 5 illustrates a typical mathematical function (composition), and Fig. 6 illustrates a typical mathematical object (primes). These show very extensively fleshed-out concepts; the knowledge *initially* provided to AM about Composition was merely its definition (Statement and Coded-Statement), Is-a, View, and Origin slots. The other slots of Compose—and all the slots of Primes—were subsequently filled in by AM.

During the course of its longest run (one PDP KI-10 cpu hour), AM defined two hundred new concepts, about half of which were judged to be reasonable (e.g., well known to humans already, or some interesting regularity involving them found by AM). AM noticed hundreds of simple relationships involving the old and new concepts, most of which were trivial. It synthesized concepts from set theory (disjointness, de Morgan's laws), stumbled across natural numbers, rapidly found arithmetic and redeveloped elementary divisibility theory, and then began to bog down in advanced number theory (after finding the fundamental theorem of arithmetic, Goldbach's conjecture, and a conjecture about highly composite numbers first found earlier in this century by the self-taught Indian mathematician Ramanujan).

The total number of 'micro-discoveries' AM made is roughly (300 old and new concepts) $\times$ (10 new slots filled in for each) $\times$ (10 entries for each slot) = 30 000. Each 'discovery' involved relying on (executing) 20–50 heuristics; the typical heuristic was used in an integral way in the making of several hundred

```
NAME:      Compose
ABBREVIATION:  - o -
STATEMENT (=DEFINITION)
   English: Compose two functions F and G into a new one FoG
   LISP: λ (F,G,H) ...  <an executable LISP predicate testing that F(G(x)) = H(x) here>
   DOMAIN:  F, G are functions
   IF-potentially-relevant: F, G are functions
   IF-truly-relevant:   Domain of F and Range of G have some intersection
   IF-resources-available: at least 2 cpu seconds, at least 200 cells
   THEN-add-task-to-agenda: Fill in entries for some slots of FoG
   THEN-conjecture:  Properties of F hold for FoG
                     Properties of G hold for FoG
   THEN-modify-slots:  Record FoG as an example of Compose
   THEN-print-to-user:   English(Compose)
   THEN-define-new-concepts: Name FoG;
                             ORIGIN Compose F,G;
                              WORTH: Average(Worth(F),Worth(G))
                              DEFN: Append(Defn(G),Defn(F))
                              Avg-cpu-time: Plus(Avg-cpu(F),Avg-cpu(G))
                              IF-potentially-rele: IF-potentially-rele(G)
                              IF-truly-relevant:   IF-truly-relevant(G)
CODED-STATEMENT (=ALGORITHM)
   CODED-IF-PART: λ(F,G) ...  <an executable LISP predicate carrying out the 3 IF- tests >
   CODED-THEN-PART:  λ(F,G) ... <an executable LISP function doing the 5 THEN- actions >
   CODED-IF-THEN-PARTS:  λ(F,G) ...  <a concatenation of the preceding two slots>
   COMPILED-CODED-IF-THEN-PARTS:  <a compiled version of the preceding slot>
SPECIALIZATIONS:  Composition-of-bijections, Composition-of-F-with-itself
GENERALIZATIONS:    Combine-concepts,    Sequential-execute,    Combine-
functions
   Immediate-Generalizations:  Combine-functions
IS-A:  Function, Deterministic-op, Math-op, Op, Math-concept, Anything
EXAMPLES:
   Good-Examples: Compose Count and Divisors
   Bad-Examples:  Compose Count and Count
CONJECTURES:  Composing F and F is sometimes very good and usually bad
ANALOGIES:  Sequence, Append
WORTH: 700
ORIGIN:  Specialization of Append-concepts with slot=Definition
   Defined-using: Specialize        Creation-date:    11/4/75     03:18
HISTORY:
   NGoodExamples:    14            NBadExamples:       19
   NGoodConjectures:  2            NBadConjectures:  1
   NGoodTasks-added: 57           NBadTasksAdded: 34
   AvgCpuTime: 1.4 seconds        AvgListCells:       160
```

FIG. 5. Frame-like representation for a mathematical function from AM. It is composed of nothing but attribute:value pairs. After each attribute or slot (often heavily hyphenated) is a colon and then a list of the entries or values for that attribute of the Compose concept.

different discoveries. Thus the set of heuristics is not merely 'unwound' to produce the discoveries. In almost all cases, the discoveries made were unexpected (by both program and author), and often were concepts and conjectures unknown to the author. Since AM's heuristics did lead to its discoveries, they must in some sense be an encoding for them, but they were not a conscious or (even in hindsight) obvious encoding.

AM's basic control structure was simple: select some slot of some concept,

```
NAME: Primes
STATEMENT
   English: Numbers with two divisors
   LISP: λ (n) (Apply* (Lisp-Statement Doubleton)
                        (Apply* (Compiled-Coded-If-Then Divisors-Of) n))
SPECIALIZATIONS: Odd-primes, Small-primes, Pair-primes
GENERALIZATIONS: Positive numbers
IS-A: Class-of-numbers
EXAMPLES:
   Extreme-exs: 2,3
   Extreme-non-exs: 0,1
   Typical-exs: 5,7,11,13,17,19
   Typical-non-exs: 34, 100
CONJECTURES:
   Good-conjecs: Unique-factorization,   Formula-for-d(n)
   Good-conjec-units: Times, Divisors-of, Exponentiate, Nos-with-3-divis, Squaring
ANALOGIES: Simple Groups
WORTH: 800
ORIGIN: Application of H2 to Divisors-of
   Defined-using: Divisors-of                    Creation-date: 3/19/76 18:45
HISTORY:
   NGoodExamples: 840                            NBadExamples: 5000
   NGoodConjectures: 3                           NBadConjectures: 7
```

FIG. 6. Frame-like representation for a static mathematical concept from AM.

and work to fill in entries for it. Since AM began with over 100 concepts, and each had about 20 slots to fill in (Examples, Generalizations, Conjectures, Analogies, etc.), there were about 2000 small tasks for AM to perform, initially. This number grew with time, because new concepts would usually be defined long before 20 slots were filled in on old ones. Each task was placed on an agenda, with symbolic reasons justifying why it should be attended to. Those tasks having several good reasons would eventually percolate to the top of the agenda and be worked on. To accomplish the selected task, AM located relevant heuristics and obeyed them. They in turn caused entries to be filled in on hitherto blank slots, defined entirely new concepts, and proposed new tasks to be added to the agenda.

Let us briefly illustrate the three types of actions initiated by heuristics. One task AM worked on was "Fill in Examples of Set-Equality". One relevant heuristic, $H5$ (see Fig. 7) said to look at the domain of Set-Equality (which was pairs of sets), look at the Sets concept, look at its Examples slot, pick (randomly) a pair of sets from there, and feed them as the input to the definition of Set-Equality, thereby producing an output of either T (true) or NIL (false). By this method a few examples of Set-Equality were found, but hundreds of *non-examples* were rejected in the process—after all, very few random sets are equal to each other. This illustrates how a few entries for the Examples slot of Set-equality were recorded.

Another heuristic, $H6$, reacted to the rarity of the Set-Equality predicate returning T: it added a new task to the agenda, namely "Fill in Generalizations

*H*5: **if** task is to find Examples(*f*), *f* Is-a Pred, and Defn(*f*) exists,
then apply Defn(*f*) to random entries on Examples(Domain(*f*))

*H*6: **if** a few Examples(*x*) are found, but over 90% are Non-examples,
then (someday) define and study various Generalizations of *x*

*H*7: **if** task is to generalize *f*, *f* Is-a Op of any kind, and one Defn(*f*) has two or more conjoined recursive calls on *f*,
then define a new Op similar to *f* but with one conjunct excised

FIG. 7. Once a task is selected, heuristics find new entries for a given slot of a given concept, propose new plausible tasks for the agenda, and synthesize whole new promising concepts.

of Set-Equality". This is the second kind of activity which we said heuristics could initiate.

When that task eventually ran, it caused heuristics to fire which defined whole new concepts—predicates similar to Set-Equality but with a definition that was slightly laxer than Set-Equality's. For instance, one heuristic (*H*7 above) accessed a recursive definition of Set-Equality, saw that it recurred in both the CAR and CDR direction, and eliminated one direction of recursion, thereby producing two new, weaker predicates (LISP functions which would return T whenever Set-Equality did, and perhaps more frequently as well). One of these two predicates turned out to be Same-First-Element-As, and the other turned out to be quite powerful, namely Same-Length-As.

There is one more issue about AM that should be discussed in this paper: how it was able to efficiently restrict its attention to a small set of potentially relevant heuristics at all times. Consider for a moment the AM heuristic that says "if a composition f∘g preserves most of the properties that *f* had then it's more interesting". That's useful when evaluating the worth of a composition, but of course is of no help when trying to find examples of Sets. We associated that heuristic with the Composition concept, the most general concept for which it was relevant. Another AM heuristic says "if the domain and range of an operation coincide, then it's more interesting". That one was tacked onto the Operation concept. But note that since Compositions are special kinds of Operations, the heuristic should apply to them as well. The general principle at work here is the following: *If a heuristic is relevant to C, then it's also relevant to all specializations of C.* Examining the AM representation for Composition (Fig. 5), we see a frame-like data structure (schema, property list) one of whose slots is Is-a, and one of the entries therein is Operation. This is AM's way of recording the fact that Composition is an instance of Operation. The obvious algorithm, then, when dealing with some specific concept *C*, is to follow Is-a and Generalization links upward, gathering heuristics tacked onto any concept encountered along the way (see Fig. 8). In general, this means that AM's attention is restricted to $\log(n)$ heuristics, rather than n. AM can completely ignore all the rest, and need only evaluate the if parts of these $\log(n)$

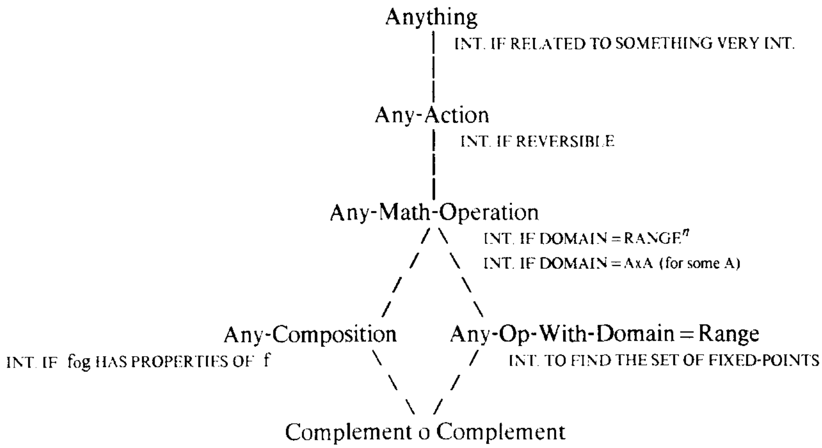

FIG. 8. One branch of the Generalization hierarchy of concepts, with a few of the attached interestingness (INT) heuristics.

potentially relevant ones. In other words, the Generalization/Specialization hierarchy of concepts has *induced* a similar powerful structuring upon the set of heuristics. The power of this technique is dimmed somewhat by the unequal distribution of heuristics in the Generalization/Specialization tree: a large number of heuristics clustered near the few topmost (very general) concepts.

As AM forayed into number theory, it had only heuristics from set theory to guide it. For instance, when dealing with prime pairs (twin primes), there were no specific heuristics relevant to them; they were defined in terms of primes, which were defined in terms of divisors-of, which was defined in terms of multiplication, which was defined in terms of addition, which was defined in terms of set-union, which (finally!) had a few attached heuristics. Because it lacked number-theory heuristics, embodying what we would call common-sense about arithmetic, AM's fraction of useless definitions shot way up: Numbers which are both odd and even; Prime triples; The conjecture that there is only one prime triple (3,5,7) but without understanding why; etc. It was unexpected and gratifying that AM should discover numbers and arithmetic at all, but it was disappointing to see the program begin to thrash. When a few dozen concepts from plane geometry were added to AM, the same type of thrashing soon occurred; the addition of specific geometry heuristics delayed this collapse.

There are two relevant conclusions from the AM research: (i) It is possible for a body of heuristics to effectively guide a program in searching for new concepts and conjectures involving them. (ii) As new domains of knowledge emerge, the old corpus of heuristics may not be adequate to serve as a guide in those new domains; rather, new specific heuristics are necessary.

One feature of Heuretics' being a 'field of knowledge' is that there can be—nay, must be—hypotheses about heuristics, experiments to test them out, and eventually a developing *theory* of heuristics. Toward that end, we can begin collecting elements of such a theory based on our experiences with AM. See Fig. 9. One remark, besides the two mentioned in the last paragraph, is that heuristics can be used both to suggest promising actions and to discourage poor ones. AM's search space is never explicitly described; there is no clear notion of a set of legal operators which defines some immense space of syntactic mathematical concepts and conjectures, etc. Any such attempt would probably produce a search space of such size as to be useless (100^{20} in AM's domain of elementary finite set theory, where definitions were about twenty nontrivial words long, and there were about 100 concepts to choose from to fill each of those blanks). Rather, AM's set of heuristics implicitly defines its search space. If you remove a heuristic from AM, it has less to do; this is exactly the opposite of the case with most heuristic search programs, where heuristics are used exclusively to prune away implausible paths. The fraction of the legal concepts that would rank as interesting, recognizable, or important is negligible; contrast that with the almost 50% hit rate of concepts proposed by AM's heuristics.

(I) A SET OF HEURISTICS CAN GUIDE CONCEPT DISCOVERY

(II) A NEW FIELD WILL DEVELOP SLOWLY IF NO SPECIFIC NEW HEURISTICS FOR IT ARE CONCOMITTANTLY DEVELOPED

(III) HEURISTICS CAN BE USED AS PLAUSIBLE MOVE GENERATORS OR AS IMPLAUSIBLE MOVE ELIMINATORS

(IV) THE GENERALIZATION/SPECIALIZATION HIERARCHY OF CONCEPTS INDUCES A SIMILAR STRUCTURE UPON THE SET OF HEURISTICS

FIG. 9. Elements of a theory of heuristics, learned from work on AM.

The final remark noted in Fig. 9 is that the heuristics can be organized into a hierarchy, induced by the Generalization/Specialization hierarchy between domain concepts (like Fig. 8). In other words, each heuristic has a domain of relevance: the most general concept to which it's relevant and all the specializations of that concept. This organization enables the interpreter, through simple inheritance, to focus on the log of the number of all heuristics in the system, rather than that entire set of heuristics, at each moment. This may not matter much for systems with a dozen or two rules, but is currently becoming crucial as we build systems with on the order of a thousand rules.

2.4. Controlling the use of heuristic knowledge

There is an implied 'control structure' for the processes of using and acquiring knowledge (solving and proposing problems, using and discovering heuristics, choosing and changing representations, etc.) In fact, it's a nontrivial assumption that a *single* control loop is powerful enough to manage both types of processes. Our experiences with expert systems in the past [5] have taught us repeatedly that the power lies in the knowledge, *not* in the inference engine.

What is that topmost control loop? It assumes that there is a large corpus of heuristics for choosing (and shifting between) representations. From time to time, some of these heuristics evaluate how well the current representations are performing (e.g., is there now some operation which is performed very frequently, but which is notoriously slow in the current representation?). At any moment, if the representations used seem to be performing sub-optimally, some attention will be focused on the problem of shifting to other ones, maintaining the same knowledge simultaneously in multiple representations, devising whole new systems of representation, etc. Similarly, we assume there are several heuristics which monitor the adequacy of the existing stock of heuristics, and as need arises formulate (and eventually work on and solve) tasks of the form "Diagonalization is used heavily, but has no heuristics associated with it; so try to find some new specific heuristics for dealing with Diagonalization". A typical heuristic rule for working on such a task might say "To find heuristics specific to *C*, try to analogize heuristics specific to concepts which were discovered the same way that *C* was discovered".

It is assumed that these representation heuristics and heuristic heuristics have run for a while, and the system is in a kind of equilibrium. The representations employed are well suited to the tasks being performed, and the heuristics being followed serve as quite effective guides for 'plausible move generation' and 'implausible move elimination.' The system now proceeds for a while along its object-level pursuits, whatever they may be (proving theorems in plane geometry, discovering new concepts in programming, etc.). Gradually, the object level evolves: new concepts are uncovered and focused upon, new laboratory techniques are discovered, long-standing open questions are answered, etc. As this occurs, the old representations for knowledge, and the old set of guiding heuristics, become less ideal, less effective. This in turn is detected by some of the heuristic heuristics discussed in the last paragraph. They cause the system to attempt to recover its equilibrium, to spend some time searching for new representations and new heuristics to deal effectively once again with the objects and operations at the object level see Fig. 10).

So new concepts, conjectures, theorems, etc. emerge all the time; as they are investigated, some turn out to be useful and some turn out to be dead-ends; using a fixed set of guiding heuristics, the rate at which useful new discoveries are made will decline gradually over time; eventually it's worth pausing in the search for domain-specific knowledge, and turning instead to the problem of finding new heuristics (perhaps by abstracting recent experiences in the task domain). The discoverer later returns to his original task, armed with new and hopefully more powerful heuristics. He keeps his eye on the new ones, trying

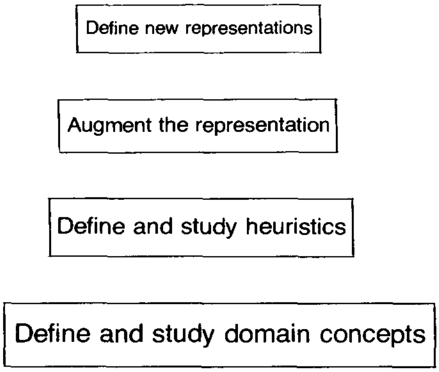

FIG. 10. Implied control structure of discovery systems. As activities at one level decline in efficacy, the system is forced to spend a little time at the next higher level before proceeding.

to gain enough experience with them to evaluate just how useful they really are.

This cycle of looking for domain concepts, occasionally punctuated by an effort to find new heuristics, continues until, gradually, it becomes harder and harder to find new heuristics. At that point it becomes worthwhile to look for new and different representations for knowledge.

The top-level control structure is thus homeostatic: detecting and correcting for any inappropriateness of representations employed or heuristics employed. For these purposes, we hypothesize that it suffices to have (and use) a corpus of heuristics for guidance. Of course that top level loop could itself be implicitly defined by a set of heuristic rules, and we would expect such rules to change from time to time, albeit very slowly. If, for example, no new concepts or operations were defined at the object level for a long period of time, then the need for close monitoring of the adequacy of the representations being employed would evaporate.

In EURISKO, meta-heuristics are in no way distinguished from object-heuristics. For example, the very general recursive rule "To specialize a complex construct, find the component using the most resources, and replace it by several alternate specializations" applies to specializing laboratory procedures, mathematical functions, heuristics (including itself!), and representational schemes.

3. The Source of Heuristics' Power

3.1. AM's need to acquire new heuristics

AM was armed with a powerful set of heuristics and concepts for its initial domain (finite set theory), and it progressed as best it could without ever abstracting its experiences into new heuristics. Earlier we claimed that the thrashing which ultimately ensued was due to the absence of such compiled bits of hindsight. By examining that claim more carefully, we hope to justify the *necessity* of periodic learning of new heuristics, at least for open-ended domains such as empirical scientific theory formation.

During the period in which AM defines its first 200 concepts (beyond the 115 it began with), 125 are judged to be 'acceptable' (i.e., well-known mathematical concepts which humans have given names to, and about which AM finds some nontrivial conjectures). This 'hit rate' of 62.5% falls off rapidly, however, if the program continues to run. Of the next 300 concepts AM defines, only twenty-nine (less than 10%) satisfy the above criterion for meaningfulness.

By adding heuristics manually, this degradation can be delayed. For example, after the 200th new concept is defined, the human observer notes that all of the conjectures involving Primes and Addition have turned out to be useless; indeed, most of them have turned out to be false. Forming this into a heuristic, and supplying it to AM, causes many poor paths to be avoided. When AM is restarted, the same 29 useful concepts emerge at the expense of 260 poor ones, rather than 271.

An experiment was performed in which, instead of the specific heuristic mentioned in the last paragraph, the new heuristic added by the user is the following more general one: "conjectures involving C and f are more likely to be useful if f has some relationship to the terms out of which C was defined". In particular, conjectures involving Primes and Multiplication (or Division) are more likely to be valuable than conjectures involving Primes with Addition, Subtraction, Composition, or Printing. Adding this heuristic to AM prevents many blind alleys from being explored, at the expense of a few genuine conjectures being missed. 27 of the useful concepts are found, and only 220 of the poor ones.

Just by adding this one heuristic, AM's hit rate rises from 9% to 11%. We conclude that augmenting AM by a few tens of new heuristics (based on its experiences in working with concepts 1–300) would be necessary if it were to maintain its initial high 62.5% hit rate while developing the next few hundred concepts. More generally, we conclude that periodic learning of new heuristics is necessary to sustain high performance at the task of developing a scientific theory. Heuristics formed during the initial theory formation experiences are potent guides to subsequent attempts to extend that theory.

3.2. The zero-th order theory of heuristics

Heuristics are compiled hindsight; they are judgmental rules which, if only we'd had them earlier, would have enabled us to reach our present state of achievement more rapidly. Why, then, is there any reason to rely on such rules to guide future behavior? It must be because of continuity in the world: Rules which were useful will continue to be useful. Rules useful in situation *S* will be useful in situations similar to *S*. Of course the actions taken in the future will be slightly different than the ones taken in the past, and there must be a presumption that small differences along that dimension also are tolerable.

The basic 0th order theory, the central assumption underlying heuristics, appears to be the following: "APPROPRIATENESS(Action,Situation) is continuous and time-invariant." That is, APPROPRIATENESS, viewed as a function of actions and of situations, is a continuous function of both variables. Moreover, of all the features of the situation which might be relevant, time is (we assume) far from the most critical variable (Fig. 11).

0th: APPROPRIATENESS(Action,Situation) is a *continuous time-invariant* function

Corollary 1: *Analogize*:
If action A is appropriate in situation S,
Then A is appropriate in most situations which are very similar to S.

Corollary 2: *Satisfice*:
If action A is appropriate in situation S,
Then so are most actions which are very similar to A.

Corollary 3: *Remember*:
If action A would have been appropriate in the past situation S,
Then the rule "If similar to S, then try A" may be useful in the future.

FIG. 11. The Central Assumption underlying heuristics, and three special cases.

Of course we can't compute the APPROPRIATENESS function precisely; we can't even sample more than a few variables from Actions and Situations. Nevertheless, this abstraction implies several interesting corollaries, and serves as a theoretical base which can be examined, criticized, and (in Section 3.3) improved. Indeed, simply by considering Appropriateness as a function, we open up the possibilities of visualizing graphs of it, a technique which proves to be a useful metaphor below.

Corollary 1. *For a given action, its appropriateness is a continuous function of the situation.*

Heuristics specify which actions are appropriate (or inappropriate) in a given situation. One corollary of the central assumption is that if the situation changes only slightly, then the judgment of which actions are appropriate also changes only slightly. Thus compiled hindsight is useful, because even though the world changes, what was useful in situation X will be useful again sometime in situations similar to X.

Corollary 1 says, in effect, that if the current task appears to be similar to one you've seen elsewhere, then many of the features of the *task environment* will probably be very similar as well: i.e., the kinds of conjectures which might be found, the solvability and difficulty anticipated with a task, the nature of blind alleys in which one might be trapped, etc. may all be the same as they were in that earlier case. For instance, suppose that a certain theorem, UFT, was useful in proving a result in number theory. Now another task appears, again proving some number theory result. Because the tasks are similar, Corollary 1 suggests that UFT be used to try to prove this new result. Corollary 1 is the basic justification for using analogy as a reasoning mechanism. A sentiment similar to this was voiced by Poincaré during the last century: *The whole idea of analogy is that 'Effects', viewed as a function of situation, is a continuous function.* Corollary 1 is the basis for employing 'generalization of stimuli' as a mechanism for coping with the world.

Corollary 2. *For a given situation, appropriateness is a continuous function of actions.*

This means that if a particular action was very useful (or harmful) in some situation, it's likely that any very similar action would have had similar consequences. Corollary 2 justifies the use of inexact reasoning, of allocating resources toward finding an approximate answer, of satisficing.

Corollary 3. *It is cost-effective to form and use situation/action rules which would have helped in the past.*

The 'time-invariant' condition in the statement of the Central Assumption (the zero-th order theory) means that the world doesn't change much over time, and is the foundation for the utility of *memory*. In a world changing radically enough, rapidly enough, memory would be a useless frill; consider the plight of an individual atom in a gas. Corollary 3 therefore states that the world is assumed to be a stable, nonvolatile place, that any rule which we know (via hindsight) would have been useful to obey in the past, will probably be of use in the future. We are presumed to be inhabiting a world in which McCarthy's Frame Problem really is a problem, where most valid assertions remain valid as situations evolve.

If the Central Assumption holds, then the ideal interpreter for heuristics is the one shown in Fig. 12. Note that this is very similar to a pure production system interpreter. In any given situation, some rules will be expected to be relevant (because they were truly relevant in situations very similar to the present one). One or more of them are chosen and applied (obeyed, evaluated, executed, fired, etc.). This action will change the situation, and the cycle begins anew. Of course one can replace the 'locate relevant heuristics' subtask by a copy of this whole diagram: that is, it can be performed under the guidance of a body of heuristics specially suited to the task of finding heuristics. Similarly,

```
                          New  Situation
                           /    →    \
                          /           \
                         /             \
Changes to the situation                \
  (hopefully for the better)  ↑          ↓  Locate relevant heuristics
  (hopefully quickly)                   /
                         \             /
                          \           /
                           \    ←    /
                      Apply chosen heuristic(s)
```

FIG. 12. The 0th order interpreter for a body of heuristic rules.

the task of selecting which rule(s) to fire, and in what order, and with how much of each resource available, can also be implemented as an entire heuristic rule system procedure. EURISKO has this self-representing architecture, and overcomes the apparent inefficiencies by employing software caching (compiling the rule sets).

By examining the loop in Fig. 12 we can quickly 'read off' the possible bugs in heuristics, the list of ways in which a heuristic can be 'bad':

- It might not be interpretable at all.
- It might be interpretable but it might never even be potentially relevant.
- It might be potentially relevant but its if- part might never be satisfied.
- It might trigger, but never be the rule actually selected for execution (firing).
- It might fire, but its then- part might not produce any effect on the situation.
- It might produce a bad effect on the situation.
- It might produce a good effect, but take so long that it's not cost-effective.

This is reminiscent of John Seely Brown's and Kurt VanLehn's [2] work on a generative theory of bugs, and is meant to be. Perhaps by viewing heuristics as performers, this approach can lead to an effective method for diagnosing buggy heuristics, hence improving or eliminating them.

3.3. The power of each individual heuristic

What is the nature of a single heuristic, the source of its power? One way of interpreting Corollary 1, above, is that each heuristic has its own particular domain of relevance, outside of which it is useless or perhaps worse than useless. Consider the following very special situation: you are asked to guess whether a conjecture is true or false. What heuristics are useful in guiding you to a decision rapidly? If the conjecture is in the field of plane geometry, one very powerful technique is to draw a diagram; see heuristic rule $H8$ in Fig. 13.

But if the conjecture is in the field of point-set topology, or real analysis, $H8$ is a terrible heuristic which will often lead you into error. For instance, if the conjecture mentions a function, then any diagram you draw will probably portray a function which is everywhere infinitely differentiable, even if such is

$H8$: **if** you are guessing the truth of a conjecture.
then draw a diagram and see if it holds in that analogic model

$H9$: **if** one quantity is spoken of as a function of another,
then graph it, and visually inspect the graph

FIG. 13. Two general heuristics for using analogic models.

never stated in the conjecture's premises. As a result, many properties will hold in your diagram that can never be proven from the conjecture's premises. The appropriate technique in topology or analysis is to pull out your book of 101 favorite counterexamples, and see whether any of them violate the conjecture. If it passes all of them, then you may guess it's probably true.

This example dramatizes the idea that the power or utility of a heuristic changes from domain to domain. Thus, as we move from one domain to another, the set of heuristics which we should use for guidance changes. Many of them have higher or lower utility, some entirely new heuristics may exist, and some of the old ones may be actually *detrimental* if followed in the new domain. For instance, the 'if object is falling then catch it' rule is useful for most situations, but each year many people are burned needlessly when they try to catch falling clothes irons and soldering irons.

According to the fundamental assumption of heuristics (the 0th-order theory of Fig. 11), the power of a heuristic is a continuous function of the task it is being applied to. But consider $H9$, above, one of the most powerful heuristics for theory formation. Let's follow its advice in our present situation, that of grappling with the development of the theory of heuristics. $H9$ says to take a heuristic H_0, and plot the graph of its power as a function of task domain, i.e., imagine graphing the utility of applying H_0 as a function of situation.

$H10$: **if** you are stumped for a solution,
then ask a human expert for the answer

$H11$: **if** you are stumped for a solution,
then do a realtime simulation to obtain an approximate answer

$H11$: **if** you are stumped for a solution,
then axiomatize the problem and apply the Resolution method

FIG. 14. Three techniques with very different domains of applicability.

Suppose our problem is to process a knowledge base about aircraft carriers, answering database queries from nonexperts. Some of the (albeit more extreme) heuristics available are listed in Fig. 14. Consider drawing the graph of power vs. situation for $H10$, the heuristic which advocates querying a human expert when you're stuck. The task or situation axis (x-axis) will be arranged by the difficulty of the problem being worked on, and the power or utility axis

(y-axis) will correspond to the difference between the time required to get an answer from the human expert and the time required to get an answer using other methods (resolution theorem proving, simulation, exhaustive search, etc.). See Fig. 15 for a glimpse of what such a graph reveals. In the case of very trivial problems, it is not worth bothering the user—it would be better for the program to work on its own for a fraction of a second and derive the answer itself, via a few database lookups and some simple inference. Thus the leftmost portion of the graph is below zero—the utility of employing the 'ask an expert' heuristic on a trivial problem is negative. For problems somewhat more difficult, asking an appropriate expert might very well be the most cost-effective method of obtaining a solution. $H10$ becomes a reasonable rule to follow. So in the middle, the graph of the heuristic's utility rises to a high value. For very difficult problems, superhuman amounts of computation may be required, and a detailed simulation may far surpass the ability of any human to provide an answer. Thus the utility declines and becomes negative. If the problem is extraordinarily difficult, then it may be insoluble no matter what methods are tired, and therefore using this heuristic is no worse than using any others—so the utility eventually rises again to zero.

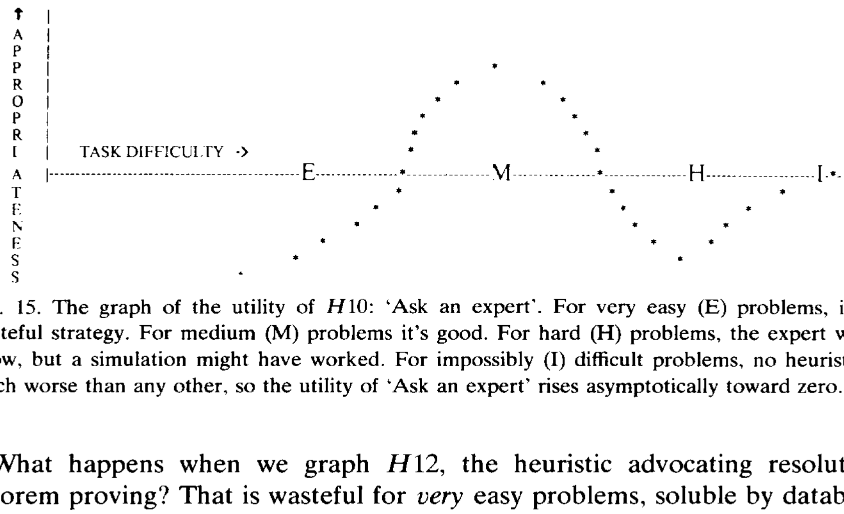

FIG. 15. The graph of the utility of $H10$: 'Ask an expert'. For very easy (E) problems, it's a wasteful strategy. For medium (M) problems it's good. For hard (H) problems, the expert won't know, but a simulation might have worked. For impossibly (I) difficult problems, no heuristic is much worse than any other, so the utility of 'Ask an expert' rises asymptotically toward zero.

What happens when we graph $H12$, the heuristic advocating resolution theorem proving? That is wasteful for *very* easy problems, soluble by database lookup, not too bad a technique for easy (E) problems, progressively worse for middling (M) and hard (H) problems, and no worse than anything else for impossibly difficult (I) ones. In short, we get a curve similar to that of Fig. 15, but skewed further to the left. Similarly, $H11$ yields a graph like Fig. 15 but skewed to the right.

As a second example, consider $H8$, the heuristic that advised drawing a diagram to help guess the truth of a conjecture. This time, the y-axis (utility, appropriateness, power) can correspond to the chance of such a technique yielding the right answer. The task or situation axis can correspond to the ordering of domains of mathematics in curricula; thus we use set theory to define arithmetic, so set theory is located to the left of arithmetic on this axis (see Fig. 16). In the case of logic, very few diagrams of any use can be drawn, so the heuristic is a slight waste of time. In set theory, Venn diagrams may be useful for easy problems, but otherwise tend to clutter up the situation rather than relieving it. In geometry, however, diagrams are in their glory; Gelernter's geometry theorem prover demonstrated vividly the power of drawing even a single diagram for a problem. Advancing to topology and real analysis, the situation reverses, and diagrams which appear to capture the situation are in fact often misleading. Diagrams are gradually rehabilitated in the etherial heights of category theory, though even there they play only an auxilliary role.

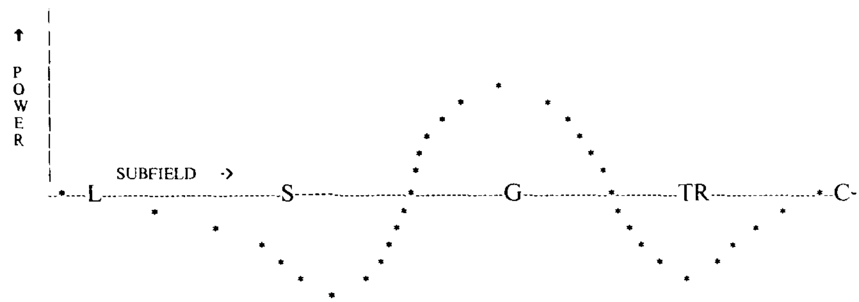

FIG. 16. The graph of the power of drawing a diagram, as a function of the mathematical subfield in which it is tried. Neutral for logic (L), misleading for set theory (S), useful for geometry (G), misleading for topology and real analysis (TR), and neutral for category theory (C).

The diagram above resembles the potential well around a particle, and for that reason a heuristic such as "Assume that two point masses repel each other" would have a utility graph similar to Fig. 16. Namely, at far distances, gravity makes that statement slightly wrong. As the objects get closer, the statement is more and more wrong, until they are so close that nuclear interaction forces overbalance gravitational ones. At nuclear distances, it's a fine heuristic to employ. Analogues of this in various situations abound (e.g., "Avoid getting too close in personal relationships").

As a fourth example, consider the task of planning for company coming to your house to visit. There are many subtasks to schedule: shopping for food, planning menus, cleaning, cooking, talking, etc. There are several heuristics (planning techniques) you might apply to deal with the problem: Pert charts, Noah-like symbolic evaluations, dynamic replanning, counterplanning, setting up of agendas, etc. Each of these methods has some situations in which it works, some in which it fails, and some in which it can't even be tried. For instance, Pert charts demand a full knowledge of dependencies, and the absence of 'cycles' among such dependencies. If the dependencies are not known, the method is not directly applicable; if the dependencies change over time, the Pert charts will be worse than useless. Thus the graph of the utility of the Pert chart method would peak in a certain region, become negative further out, and eventually become zero (as the task got far away from planning).

In general, then, graphing the utility or power of a heuristic H_0 as a function of task domain, if it can be done at all, produces a curve or histogram resembling that of Fig. 16. Typically, there is some range of tasks for which the heuristic has positive value. Outside of this, it is often counterproductive to use the heuristic. For tasks sufficiently far away, the utility approaches zero, because the heuristic is never even considered potentially relevant, hence never fires. E.g., recall $H6$, which said in effect: "if a predicate rarely returns True, then (someday) define new generalizations of it". This heuristic is useful in set theory, worse than useless in number theory, and useless in domains where 'predicate' is undefined.

Sometimes, one (or both) sides of the negative region simply keep getting more negative (as in Fig. 15) rather than reapproaching zero. Sometimes one side drops precisely to zero and stays there (e.g., if the heuristic has a very crisp condition under which it is applicable, then considering using it anywhere else has zero utility because the heuristic will never 'fire'). Of course the shape of the curve depends on how the tasks are ordered on the x-axis, and on what the utility measure is along the y-axis. Indeed, as we have mentioned, the whole notion of graphing this function is primarily a metaphorical device to aid us in further thinking about the theory of heuristics.

If we specialize the then- part of a heuristic, it will typically have higher utility but only be relevant over a narrower domain (see Fig. 17). Notice the area under the curve appears to remain roughly constant; this is a geometric interpretation of the tradeoff between generality and power of heuristic rules. Since the graphs are metaphorical, this notion of conservation of area under a

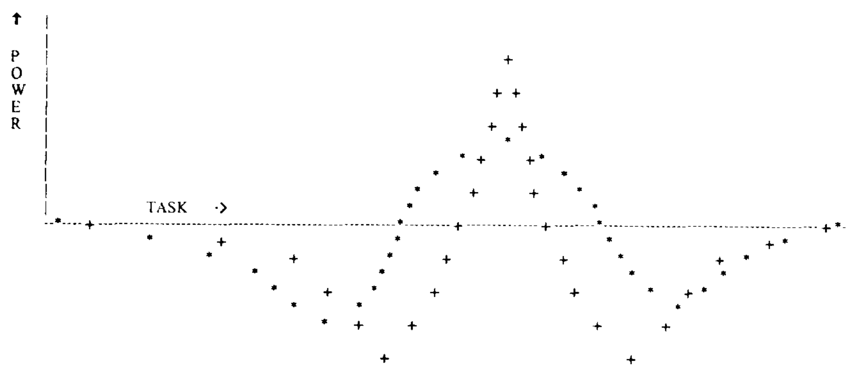

FIG. 17. The change in power when a heuristic (*) has its then-part specialized (+).

curve is likewise a '0th-order' idealization. It is also worth noticing that the new specialized heuristic may have negative utility in regions where the old general one was still positive, and it will be meaningless over a larger region as well. Consider for example the case where "Generalize a predicate" is specialized into "Generalize a predicate by eliminating one conjunct from its definition". The latter is more powerful, but only applies to predicates defined conjunctively; EURISKO found a domain where this heuristic has negative worth (namely, situations in which the if- parts of heuristics are being modified, see *H*17 in Fig. 23).

By examining Fig. 17, it is possible to generate a list of possible bugs that may occur when the actions (then- part) of a heuristic are specialized. First, the domain of the new one may be so narrow that it is merely a spike, a delta function. This is what happens when a general heuristic is replaced by a table of specific values. Another bug is if the domain is not narrowed at all; in such a case, one of the heuristics is probably completely dominated by the other. A third type of bug appears when the new heuristic has no greater power than the old one did. For example, "Smack a vu-graph projector if it makes noise" has much narrower domain, but no higher utility, than the more general heuristic "Smack a device if it's acting up". Thus, the area under the curve is greatly diminished, but no benefit accrues.

While the last paragraph warned of some extreme bad cases of specializing the then- part of a heuristic, there are some extreme good cases which frequently occur. The utility (power) axis may have some absolute desirable point along it (e.g., some guarantee of correctness or efficiency), and by specializing the heuristic it may exceed that threshold (albeit over a narrow range of tasks). In such a case, the way we *qualitatively* value that heuristic may alter; e.g., we may term it 'algorithmic' or 'real-time'. One way to rephrase this is to say that algorithms are merely heuristics which are so powerful that guarantees can be made about their use. Conversely, one can try to apply an algorithm outside its region of applicability, in which case the result may be useful and that algorithm is then being used as a heuristic. The latter is frequently done in mathematics (e.g., pretending one can differentiate a complicated expression, to aid in guessing its value). Another pathologically extreme specialization of a heuristic is turning it into one which applies only on a set of measure zero. This is not necessarily a bad thing: tables of values *do* have their uses.

Specializing the if- part of a heuristic rule results in its having a smaller region of non-zero utility. That is, it triggers less frequently. As Fig. 18 shows, this is like placing a filter or window along the x-axis, outside of which the power curve will be absolutely zero. In the best of cases, this removes the negative-utility regions of the curve, and leaves the positive regions untouched. For example, we might preface the "Draw a diagram" heuristic with a new premise clause, "If you are asked to test a geometry conjecture". This will

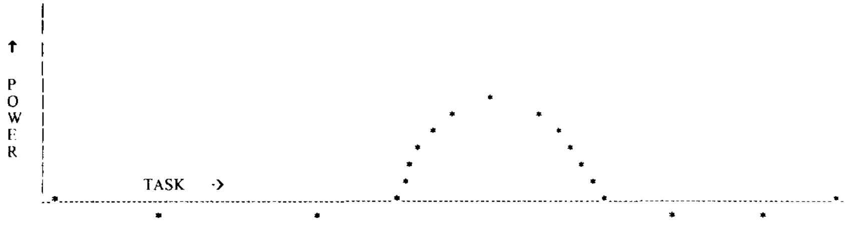

FIG. 18. The graph of a heuristic's power, after its if- part has been optimally specialized.

cause us to use the rule only in geometry situations, a domain where we know it has high utility.

By examining Fig. 18, we can generate a list of possible bugs arising from specializing the conditions (if- part) of a heuristic rule. The new window may be narrowed to a spike, thus preventing the rule from almost ever firing. There may be no narrowing whatsoever: in that case, it typically would add a little to the time required to test the if- part of the rule, while not raising the power at all. Of course the most serious error is if it clips away some—or all!—of the positive region. Thus, we would not want to replace a general diagram-drawing recommendation with one which advised us to do so only for real analysis conjectures. Empirical results from experiments on specializing and generalizing heuristics are presented in Section 4.1.

What are the implications of this simple 'theory of heuristics'? One effect is to determine in what order heuristics should be chosen for execution; this is discussed two paragraphs down. A second effect is to indicate some very useful slots that each heuristic can and should have, attributes of a heuristic that can be of crucial importance: the peak power of the rule, its average power, the sizes of the positive and negative regions (both projections along the task axis (x-axis) and the areas under the curves), the steepness with which the power curve approaches the x-axis, etc. Let us take the last attribute to illustrate. Why is it useful to know how steeply the power curve approaches Utility $=0$ (the x-axis)? If this is very steep, then it is worth investing a great amount of resources determining whether the rule is truly relevant in any situation (for if it is slightly irrelevant, then it may have a huge negative effect if used). Conversely, if the slope is very gentle, then very little harm will result from slightly-inappropriate applications of the rule, hence not much time need ever be spent worrying about whether or not it's truly relevant to the situation at hand.

The whole process of drawing the power curves for heuristics is still conjectural. While a few such graphs have been sketched, there is no algorithm for plotting them, no library of thousands of catalogued and plotted heuristics, not even any agreement on what the various power and task axes should be. Nevertheless, it has already proven to be a useful metaphor, and has suggested some important properties of heuristics which should be estimated (such as the just-mentioned downside risk of applying a heuristic in a slightly inappropriate situation). It is a qualitative, empirical theory [12], and predicts the form that a quantitative theory might assume.

How should heuristics be chosen for execution? In any given situation, we will be at a point along the x-axis, and can draw a vertical line (in case of multi-dimensional task axes, we can imagine a hyperplane). Any heuristics which have positive power (utility) along that line are then useful ones to apply (according to our theory of heuristics), and the ones with high power should be applied before the ones with low power. Of course, it is unlikely we would know the power of a heuristic precisely, in each possible situation; while diagrams such as Figs. 15–18 may be suggestive, the data almost never is available to draw them quantitatively for a given heuristic. It is more likely that we would have some measure of the average power of each heuristic, and would use that as a guess of how useful each one would be in the current situation. Since there is usually a tradeoff between generality and power, a gross simplification of the preceding strategy is simply to apply the most specific heuristic first, and so on. This is the scheme AM used, with very few serious problems. If all heuristics had precisely the same multiple integral of their power curves, this would coincide with the previous scheme. Of course, there are always some heuristics which, while being very general, really are the most important ones to listen to if they ever trigger ("If a conflagration breaks out, then escape it"), and some so important that natural selection has 'wired them in' as reflexes ("If there is a sudden bright light, then close your eyes quickly").

Notice that the 'generality vs. power' tradeoff has turned into a statement about the conservation of volumes in $n \times m$-dimensional space, when one takes the multiple integral of all the power curves of a heuristic. In particular, there are tradeoffs among all the dimensions: a gain along some utility dimension (say Convincingness) can be paid for by a decrease along another (say Efficiency) or by a decrease along a task dimension (a reduction of breadth of applicability of the heuristics). One historically common bug has been over-reliance upon (and glorification of) heuristics which are pathologically extreme along some dimension: tables, algorithms, weak methods, etc.

Heuristics are often spoken of as if they were incomplete, uncertain knowledge, much like mathematical conjectures or scientific hypotheses. This is not necessarily so. The epistemological status of a heuristic, its justification, can be arbitrarily sound. For example, by analyzing the optimal play of Blackjack, a rather complex table of appropriate actions (as a function of situation) is built up. One can simplify this into a 'Basic Strategy' of just a few rules, and know quite precisely just how well those rules should perform. That is, heuristics may be built up from systematic, exhaustive search, from 'complete' hindsight. Another example of the formal, complete analysis of heuristic methods is familiar from physics, where Newtonian mechanics is known to be only an approximation to the world we inhabit. Relativistic theories quantify that deviation precisely. But rather than supplanting Newtonian physics, they *bolster* its use in everyday situations, where its inadequacies can be quantitatively shown to be too small to make worthwhile the additional computation required to do relativistic calculations.

Many, nay most, heuristics *are* merely conjectural, empirical, aesthetic, or in other ways epistemologically less secure than the Basic Strategy in Blackjack and Newtonian physics. The canonical *use* of heuristics is to guide future behavior in cost-effective channels; the canonical use of a conjecture is to guide a search for a proof of it. If a conjecture turns out to be false (such as Newtonian mechanics, or the assertion that there is always a generality vs. power tradeoff) it may yet stand as a useful heuristic.

3.4. The space of heuristics

Imagine graphing the utility of an entire *set* of heuristics, as a function of the tasks it's being applied to. Not surprisingly, the curve produced would resemble the one produced by a single heuristic (Fig. 16), for it is (to first approximation) a huge compound heuristic (call it a Mega-heuristic). Hopefully, the set of heuristics is more useful than any member, thus its graph is probably much broader and taller (or less negative) than that of any single heuristic inside it.

One cannot simply 'superpose' or 'max' the curves of its members; the interactions among heuristics are often quite strong, and independence is the exception rather than the rule. Often, two heuristics will be different methods for getting to the same place, or one will be a generalization or isomorph of the other, etc., and as a result the set will really not benefit very much from having both of them present. On the other hand, sometimes heuristics interact synergistically, and the effects can be much greater than simple superposition would have predicted. The opposite of this sometimes happens: two experts have each provided a set of heuristics which works, yet some heuristics in each set directly contradict some in the other set. Using either half-corpus would solve your problem, but mixing them causes chaos (e.g., one mathematician gives you heuristics for finding empirical examples and generalizing, while a second gives you heuristics for formally axiomatizing the situation; either may suffice, the unstructured mixing of the two sets can be catastrophic).

Just as a set of heuristics can be conceptually grouped into a large Mega-heuristic, so an individual heuristic may be atomized into a cloud of much smaller heuristics. Much of the expertise we tap from human experts, when building expert systems, is their feel for the proper *level* at which to state and use heuristic knowledge. If the heuristics are too small, they stop being meaningful chunks of wisdom to the human expert, and risk having many stray interactions. Often languages which enforce a small grain size for rules have facilities to 'chain' them together to prevent such crosstalk. If the heuristic rules are too large, we begin to lose the benefits of taking a heuristic rule-guided approach: additivity, synergy, ease of entry and explanation and modifiability. Ultimately, we are left with one 'heuristic' which is an opaque lump of LISP code performing the entire task.

Heuretics is interested in the space of all the world's heuristics. What is its structure? What regularities in it can be exploited? The sheer size of this space—and our as yet minuscule experience in navigating within it—make these tantalizing questions difficult to investigate.

By examining—and generalizing—heuristics from a dozen disparate fields (including set theory, number theory, biological evolution, evolution of naval fleets, LISP programming, game-playing, and oil spill cleanups), we have built up some data—and some conjectures—involving heuristic-space. Consider arranging all the world's heuristics in a generalization/specialization hierarchy, with the most general ones at the top. At that top level lie the so-called weak methods (generate and test, hill-climbing, matching, means-ends analysis, etc.). At the bottom are millions of very specific heuristics, involving domain-specific terms like 'King-side' and 'Arsenic'. One may picture a Christmas tree, with a pure angel at the top, and the worthwhile gifts at the bottom.

In between are heuristics such as those depicted in Fig. 19: "Look for fixed points", "Examine extreme cases", "See what happens when a process is repeated", "Given $f(x,y)$, examine what happens when $x = y$". These are more specific than the weak methods at the top of the tree, yet are far from domain-dependent heuristics below them. Progressing downward, more and more conditions appear on the left-hand sides of the heuristics (if's), and more specialized advice appears on the right-hand sides (then's).

A purely 'legal-move' estimate of the size of this tree gives a huge final number: Based on the lengths and vocabularies of heuristic rules in AM, one may suppose that there are about 20 blanks to be filled in in a typical heuristic, and about 100 possible entries for each blank (predicate, argument, action, etc.) related to AM's math world. So there are 10^{40} syntactically well-formed heuristics just in the elementary mathematics corner of the tree. Of course, most of these are never (thankfully!) going to fire, and almost all the rest will perform irrelevant actions when they do fire. From now on, let's restrict our attention to the tree of only those heuristics which have positive utility at least in some domains.

What does that tree actually look like? One can take a specific heuristic and generalize it gradually, in all possible ways, until all the generalizations collapse into weak methods. Such a preliminary analysis (using AM's heuristics) led us to expect the tree to be of depth about 50, and in the case of an expert system with a corpus of a thousand rules, we might expect a picture of them arranged so to form an equilateral triangle. But when we went through this partial tree, analyzing the power of the rules therein, it quickly became apparent that most

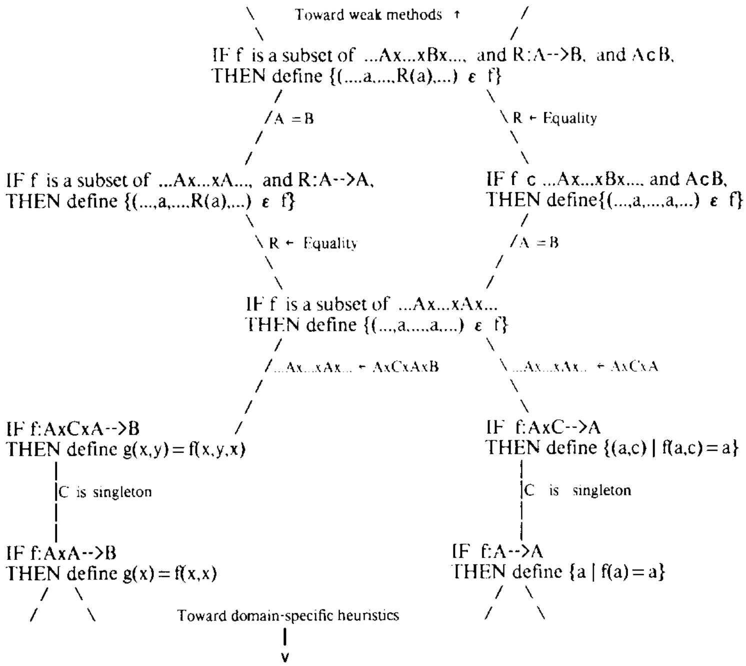

FIG. 19. A tiny fragment of the graph of all heuristics, related by generalization/specialization. Note the similar derivation of coalescing and fixed-points heuristics.

generalizations were no less powerful than the rule(s) beneath them! Thus the specific rule can be eliminated from the tree. The resulting tree has depth of roughly 3 or 4, and is thus incredibly shallow and bushy. Herbert Simon, Woody Bledsoe, and the author analyzed the 243 heuristics from AM, and were able to transform their deep (depth 12) tree into an equivalent one containing less than fifty rules and having depth of only four.

Looking at heuristics arranged in a tiny tree (e.g., Fig. 19), we observed that all but the top and bottom levels can be eliminated. A similar phenomenon was seen earlier in the case of a heuristic which said to smack a vu-graph projector in case it acted up; it and several levels of its generalizations can be eliminated, since they are no more powerful than the general "Smack a malfunctioning device" heuristic. Some very specific rule, such as "Smack a Nanook 807 vu-graph projector on its right side if it hums", might embody some new, powerful, specific knowledge (such as the location of the motor mount and this brand's tendency to misalign), and thus need to stay around.

This 'shallow-tree' result should make advocates of weak methods happy, because it means that there really is something special about that top level of the hierarchy. Going even one level down (to more specific rules) means paying attention not to an additional ten or twenty heuristics, but to hundreds. It should also please the knowledge engineering advocates, since most of the very specific domain-dependent rules also had to remain. It appears, however, to be a severe blow to those of us who wish to automatically synthesize new heuristics via specialization, since the result says that that process is usually going to produce something no more useful than the rule you start with. Henceforth, we shall term this the *shallow-tree problem.*

There are two ways out of this dilemma, however. Notice that 'utility of a heuristic' really has several distinct dimensions: efficiency, flexibility, power for pedagogical purposes, usefulness in future specializations and generalizations, etc. Also, 'task features' has several dimensions: subject matter, resources allotted (user's time, cpu time, space, etc.), degree of complexity (e.g., consider Knuth's numeric rating of his problems' difficulty), time (i.e., date in history), paradigm, etc. If there are n utility dimensions and m task dimensions, then there are actually $n \times m$ different power curves to be drawn for each heuristic. Each of them may resemble the canonical one pictured in Fig. 16. If by specializing a heuristic we create one which has the appearance of Fig. 17 *in any one of these* $n \times m$ *graphs*, then it is a useful specialization. So, while a specialization is unlikely to be useful in any particular utility/task graph, it is quite likely to be useful according to some one of the $n \times m$ such graphs.

Consider the 'Focus of Attention' heuristic; that is, one which recommends pursuing a course of action simply because it's been worked on recently. Using this as one reason to support tasks on its agenda made AM appear more intelligent to human observers, yet actually take longer to make any given discovery. Thus, it is useful in the 'Convincingness' dimension of utility, but may be harmful vis a vis 'Efficiency'.

As another example, consider the heuristics "Smack a vu-graph projector if it's acting up", "Smack a child if it's acting up", and "Smack a vu-graph projector or child if it's acting up". There may be some utility dimensions in which the third of those is best (e.g., scope, humor). However, the rationale or justification for the first two heuristics is quite different (random perturbation toward stable state vs. reinforcement learning). Therefore the third heuristic is probably going to be deficient along other utility dimensions (clarity, usefulness for analogizing, ease of teaching).

But there is an even more basic way in which the "shallow-tree" problem goes away. There are really a hundred different useful relationships that two heuristics can have connecting them (Possibly-triggers, More-restrictive-if-part, Faster, My-average-power-higher-than-your-peak-power, Asks-fewer-questions-of-the-user, etc.) For each such relation, an entire graph (note that even the Genl/Spec relation generated a graph, not a tree—see Fig. 15) can be drawn of all the world's heuristics; pragmatically, we considered only those in a given program. In some of these trees or graphs, we found the broad, shallow grouping that was found for the AM heuristics under Genl/Spec. For others, such as Possibly-Triggers, we found each rule pointing to a small collection of other rules, and hence the depth was quite large (approximately 30 for AM, not including cycles). There are still many difficult questions to study, about this phenomenon, even with the theory in this primitive state: How does the shape of the tree (the graph of heuristics related by some attribute R) relate to the ways in which R ultimately proves itself to be useful or not useful? Already, one powerful correlation seems to hold: In cases where the tree depth is great, that relation is a good one to generalize and specialize along; in cases where the resulting tree is very broad and shallow, other methods (notably analogy) may be more productive ways of getting new heuristics.

3.5. The first-order theory of heuristics

There are several things wrong with the 0th-order theory: it presumes that knowledge is complete and unchanging; that is, it ignores the 'potato in the tailpipe' problem, and 'solves' the frame problem by asserting that assertions never change their validity (another way to view this that it *spawns* the frame problem). Corollary 1 above (see Fig. 11) presumes that the axis of 'Situations' is well defined and continuous, when of course it is neither. As we said earlier, the items in Fig. 9 are 2nd-order correction terms to a theory of heuristics, and Fig. 11 is a very simplified 0th-order theory. Intermediate between them lies a theory which interfaces to each. That first-order theory says that the 0th-order theory is often a very useful fiction. It is cost-effective to behave as though it were true, if you are in a situation where your state of knowledge is very incomplete, where there is nevertheless a great quantity of knowledge already known, where the task is very complex, etc. At an earlier stage, there may have been too little known to express very many heuristics; much later, the environment may be well enough understood to be algorithmized; in between, heuristic search is a useful paradigm. Predicting eclipses has passed into this final stage of algorithmization; medical diagnosis is in the middle stage where heuristics are useful; building programs to search for new representations of knowledge is still pre-heuristic (Fig. 20).

Notice that the 1st-order theory is itself a heuristic! This is not too disturbing, since it is dubious that we will ever know enough about thinking to supplant it. Until your model of me is *absolutely* perfect, your predictions of my behavior will diverge more and more as time proceeds, and after a relatively short interval you will have to rely upon heuristics again to understand and predict my thoughts and actions. And there is probably something akin to Heisenberg's uncertainty principle to guarantee that your model of me can never be perfectly complete.

1st: If you are in a complex, knowledge-rich, incompletely-understood world, Then it is frequently useful to behave as though it were true that APPROPRIATENESS(Action,Situation) is continuous and time-invariant.

FIG. 20. The first-order theory of heuristics: the 0th-order theory is a useful fiction.

3.6. The second-order theory of heuristics

The second-order corrections in Fig. 9 (and, as we shall soon see, Fig. 21 below) now apply to the first-order theory (e.g., the division of heuristics into generators and pruners). Additionally, some new second-order ones are apparent. For instance, the adjective 'frequently', used in Fig. 20, can be replaced by a body of rules which govern when it is and is not useful to behave so. Finally, careful examination of the use of heuristics in AM reveals some regularities which seem to be the opposite of the claims of the 0th-order theory.

Heuristics are compiled hindsight: they are nuggets of wisdom which, if only we'd had them sooner, would have led us to our present state much faster. This means that some of the blind alleys we pursued would have been avoided, and some of the powerful discoveries would have been made sooner.

Even the synthesis of a new discovery can be considered to be the result of employing guidance heuristics, rules of good guessing based on analogy, aesthetic criteria such as symmetry, or random combination. A few typical such rules would be "Analogies are useful in formulating biological and sociological theories", "Symmetry is useful in postulating the existence of fundamental particles in physics", "Randomly look at empirical data for regularities in elementary number theory and plane geometry", "Once a correlation is observed, consider the extreme cases of that relationship". Those guidance heuristics were in turn based on several past episodes, hence are themselves compiled hindsight. Nilsson and others have argued for the primacy of search; we are simply stating the very special case where we cannot decide which node to investigate next, but rather must let Time carry a stream of events past us, each event serving as a node for our observation and recording: the primacy of compiled experiential knowledge.

As new empirical evidence accumulates, it may be useful to 'recompile' the new hindsight into heuristics (synthesize new heuristics and modify old ones). AM demonstrated that, certainly by the time you've opened up a whole new field, you *must* recompile. Working in point-set topology with geometry heuristics is not very efficient, nor was AM's working in number theory using only heuristics from set theory. The set of heuristics must evolve: some old ones are no longer useful, some must be refined to suit the new domain, and some entirely new heuristics may be useful. As the task varies, or as time varies and one gains new experiences, one's set of guiding heuristics is no longer optimal. The utility of a heuristic will vary, then, both across tasks and across time, and this variance is not necessarily continuous.

Exactly what kinds of changes can occur in a domain of knowledge that might require you to alter your set of heuristics? In other words, what are the sources of *granularity* in the space of 'fields of knowledge'?

First, there might be the invention of a new piece of apparatus. This could be theoretical (such as Gödel's theorem) or technological (such as the electronic digital computer). The first few painful experiences with a new invention quickly lead to a specialized corpus of heuristics: rules which tell you how to use such a thing, where not to poke your fingers, when it's relevant, how to fix one, what kind to buy, etc. In addition, many of the old heuristics may be less or (rarely) more useful than they used to be. The invention of the airplane invalidated most of the long distance travel heuristics then extant, reinforced the heuristic that said to be skeptical of printed timetables, and led to the creation of many new rules of thumb for dealing with air travel.

Second, there might be a new technique devised, one which doesn't actually depend upon any new apparatus. Again, this can be theoretical (such as the recent widespread application of divide and conquer in complexity theory) or practical (such as Maxam and Gilbert's ingenious method for sequencing DNA). New heuristics about reliability, applicability, etc. become useful.

Third, a new phenomenon may be observed. When a new invention (e.g., the telescope) occurs, there are often two immediate new phenomena: the sociological one of how the invention is used, and the 'real' one now observable using the invention.

Fourth, and most unusually, there may be a newly-explicated or newly-isolated concept or field, one which was always around but never spoken about explicitly. Three such concepts, recently out of the closet, are: paradigms in scientific research, the whole field of heuristics itself, and the analysis of algorithms.

In brief, the four sources of granularity in the space of 'domains of knowledge' are precisely those components which, if varied, lead to a new domain of knowledge. In other words, they *define* what we mean by a domain of knowledge: a set of phenomena to study, a body of specific problems about those phenomena which are considered worth working on, and a set of methods (both theoretical and experimental, mental and material) for attacking such questions.

The space of domains is granular, quantized, hence the 'power curves' we drew earlier for individual heuristics are really step-functions (or histograms) rather than smooth curves as we've drawn them. One implication of this is that there is a very precise point along the task axis where the utility drops from positive to negative (or zero). Often this is a large, sudden drop across a single discontinuity in the axis (e.g., when a product emerges, an expert dies, a theorem is proved).

One frequent problem we face when trying to apply heuristics is not being able to evaluate their if- parts, their conditions. We may not know whether the acyclic preconditions demanded by Pert techniques are satisfied; we may not know for sure whether the difficulty of the request from the aircraft database is neither too trivial nor too complex; etc. In such a situation, we rely on heuristics for deciding which heuristics to apply. A few such are:

(1) *Nonmonotonic reasoning*: assume that some of the uncertain conditions hold, and tag dependencies so that it is easy to undo consequences of that heuristic application if it later turns out that the assumption was wrong.

(2) *Deferral*: if all of the alternative heuristics would cause a certain sub-action to be taken (as one entry of their then- parts), then take that action now and hope that by the time it finishes more knowledge will be available to aid in choosing among the competing heuristics.

(3) *Approximation*: weaken some of the conditions for applicability of the heuristics. E.g., replace 'all' by 'most', 'equal' by 'similar', eliminate one entire conjunct from a condition comprised of many conjuntive tests, etc. This applies to heuristics for choosing heuristics as well; thus one could weaken (2) above, into a rule that said "if *most* of the alternative heuristics would cause a certain action to be taken ...", replacing the technique of *guaranteed* deferral with *plausible* deferral.

This section has now contributed three new elements to our growing theory of heuristics (see Fig. 21).

(v) HEURISTICS ARE COMPILED HINDSIGHT
(vi) THE SPACE OF 'DOMAINS OF KNOWLEDGE' IS GRANULAR
(vii) USE HEURISTICS TO DECIDE WHICH HEURISTIC TO APPLY NEXT

FIG. 21. Three additional (see Fig. 9) elements of a theory of heuristics.

4. EURISKO: The Origin of New Heuristics

Recently, the AM program has been extended into EURISKO, a program capable of discovering new heuristics as well as new mathematical concepts. The AM heuristics were originally coded as opaque lumps of LISP code—immutable and uninspectable by the system. In EURISKO these have each been recast as full-fledged units, with their content spread out into dozens of kinds of slots. The corpus of heuristics guides the synthesis, data gathering, and judgmental evaluation of new concepts—be they new math concepts (PrimeNumOfDivis), representation concepts (VolatileSlots), or heuristics (GeneralizeRareOp). This section briefly recounts some of the design considerations and runtime experiences we have had to date with EURISKO.

4.1. Meta-heuristics are just heuristics

Is there something special about the heuristics which inspect, gather data about, modify, and synthesize other heuristics? That is, should we distinguish 'meta-heuristics' from 'domain heuristics'? According to our general theory, as presented in Section 3, domains of knowledge are granular but nearly continuous along every significant axis (complexity of task, amount of quantification in the task, degree of formalization, etc.) Thus, our first hypothesis is that it is not necessary to differentiate meta-level heuristics from object-level heuristics—nay, that it may be artificial and counterproductive to do so.

This is one hypothesis upon which the design of EURISKO rests. Fig. 22 illustrates three heuristics which can deal with both heuristics and mathematical functions. The first one says that if some concept f has always led to bad results, then f should be marked as less valuable. If a mathematical operation, like Compose, has never led to any good new math concepts, then this heuristic would lower the number stored on the Worth slot of the Compose concept. Similarly, if a heuristic, like the one for drawing diagrams, has never paid off, then its Worth slot would be decremented. EURISKO put this rule to frequent and good use, so there was little chance in practice of it applying to itself (though in principle it might have).

$H12$: **if** the results of performing f have always been numerous and worthless,
then lower the expected worth of f

$H13$: **if** the results of performing f are only occasionally useful,
then consider creating new specializations of f by specializing some slots of f

$H14$: **if** a newly-synthesized concept has slots that coincide in value with those of an already-existing concept,
then the new concept should be destroyed because it is redundant

FIG. 22. Three heuristics capable of working on heuristics as well as math concepts.

The second heuristic $H13$ says that if some concept has been occasionally useful and frequently worthless, then it's cost-effective to seek new, specialized versions of that concept, because some of them might be much more frequently utile (albeit in narrower domains of relevance). Composition of functions is such a math concept—it led AM to some of its biggest successes and failures. $H13$ added a task to AM's agenda, which said "Find new specializations of Compose". When it was eventually worked on, it resulted in the creation of new functions, such as 'Composition of a function with itself', 'Composition resulting in a function whose domain and range are equal', 'Composition of two functions which were derived in the same way', etc. $H13$ also is present in EURISKO, but there it also sometimes applies to heuristics, in fact once $H13$ applied to *itself*. How did that happen? $H13$ was sometimes useful and sometimes not, and so it truly did pay to seek new, specialized variations of $H13$. Four of the many specializations were: heuristics which demand that f has proven itself useful at least 3 times, that f be specialized in an extreme way, that f have proven itself extraordinarily useful at least once, and that the specializations still be capable of producing any of the successful past creations of f. EURISKO's full results in this case were as follows.

2 heuristics that were more specialized and potentially more useful and more

powerful (including '. . . then specialize one of its criterial (not merely descriptive) slots').

4 heuristics which looked more specialized but were exactly the same as the original one (including '. . . and which has been used several times . . . .').

180 heuristics which were more restricted in applicability, yet performed actions identical to the original when they were applicable (e.g., '. . . and the concept represents a heuristic rule . . .').

107 heuristics which were so specialized they would (essentially) never fire (e.g., '. . . and the concept is Set-Union', 'and the concept is a set-theory function and a geography-function').

5 heuristics which were simply wrong—i.e., would cause much more harm than good if they were used in guiding the program (including 'if the results of applying f are never useful', 'then specialize a noncriterial slot').

The conclusion is that heuristics can operate on each other (and themselves) to synthesize new heuristics, but the process is very explosive, and must be heavily constrained if it is to be worthwhile pursuing.

Near the end of Section 3.3, we found it feasible to constrain the 'choose the next heuristic to apply' problem by using a few heuristics for guidance. A similar approach was tried in the above case, not by hand but by EURISKO itself:

Rather than hand-crafting some 'meta-rules', we simply re-ran EURISKO all over again, but keeping the four synthesized heuristics to which EURISKO had given its highest Worth ratings. These are shown in Fig. 23. The first two are special cases of $H13$. Each of them also claims to *subsume* $H13$, thereby effectively turning it off for the duration of the second run. Heuristic $H15$ suggests specializing only those slots of f which are Criterial (defining rather than commentary). Thus, a terrible specialization such as used to arise by altering only the EnglishStatement slot could no longer occur. $H16$ limits its recommendations to those slots which, viewed as units in their own right, have

$H15$: **if** the results of performing f are only occasionally useful,
then consider creating new specializations of f by specializing some *criterial* slots of f

$H16$: **if** the results of performing f are only occasionally useful,
then consider creating new specializations of f by specializing some *highly-rated* slots of f

$H17$: **if** modifying any 'if- part' of a heuristic H,
then don't replace 'and' by any other predicate.

$H18$: **if** a newly-synthesized concept has *criterial* slots that coincide in value with those of an already-existing concept,
then the new concept should be destroyed because it is redundant

FIG. 23. Four new heuristics synthesized by EURISKO. Two 'constrained generation' heuristics and an 'implausible pruning' heuristic replace $H13$, yielding less explosive results.

high Worth values. Occasionally, both rules ($H15$ and $H16$) support the same task, and that task will jump to the top of the Agenda and is worked on almost immediately.

The next heuristic, $H17$, is a bit of compiled hindsight which, if only it had existed all along, would have prevented one of the disastrous explosions of worthless concepts due to the synthesis of a terrible heuristic. While the other heuristics in Fig. 23 are small perturbations on existing heuristics, $H17$ is completely new (though synthesized using prexisting templates, to be sure). How did this rule get synthesized?

EURISKO originally used $H13$, sometimes to good advantage, and decided to generalize it. A task to that effect was placed on the Agenda, and eventually it was selected as the best task to work on for a while. EURISKO chose, at random, the IfPotentiallyRelevant slot as piece of $H13$ to generalize. This had contained 'if the task is to specialize C, and no slot to specialize has yet been chosen'; that is, this test was a predicate with two conjuncts. EURISKO generalized this by replacing 'and' by 'TheFirstOf'—i.e., by eliminating the second conjunct. In this manner a new, generalized heuristic, $H13$b, was created. Why was it so terrible? Instead of placing tasks on the Agenda only when a particular slot hadn't been decided, $H13$b fired even when the selected slot was known! This resulted in a continuous stream of new tasks, and eventually new concepts, being synthesized. Finally, another heuristic caught this, by noticing the sudden influx of uninvestigated, uninstantiated concepts. It destroyed the mutant $H13$b, and synthesized a few new heuristics, rules which would have been capable of preventing such a mutant from ever being created. One of those eventually got a high Worth rating, and it appears as $H17$ in Fig. 23.

The final heuristic in Fig. 23 needs little commentary; it is a specialization of the final heuristic in Fig. 22, but is much more useful, as the empirical results of rerunning EURISKO showed. With the four heuristics from Fig. 23 added to the initial state of the EURISKO System, the results changed dramatically. For the particular case above, of $H13$ applying to itself, they were:

2 heuristics that were more specialized and potentially useful;
4 heuristics which looked more specialized but were not;
9 heuristic which applied less often and did the same thing;
20 heuristics which were so specialized they would never fire;
4 heuristics which were simply wrong and harmful.

The very good—and the very dangerous—heuristics were still generated and passed on for future consideration; the intermediate ones, the ones which would appear foolish to a human on first reading them, were almost completely suppressed. The only way to eliminate any of the four harmful specializations from being considered, however, was to add (by hand) new pruning heuristics.

Overall, the number of new heuristics synthesized was reduced by an order of magnitude. Five hundred tasks were worked on during the first execution, but only 75 tasks needed to be run during the second execution (with the four new rules from Fig. 23). The times for these runs were, respectively, 34 and 9 cpu minutes (on a DEC 20/60, running Interlisp). The 256 k of address space was quickly exhausted, and it was necessary to employ a means to swap units out onto disk (we used the RLL language) or a machine with a larger virtual address space (we now have access to a Xerox Dolphin).

When run for very long periods of time, EURISKO invents ways of entering infinite loops (e.g., a mutant heuristic which manages to alter the situation so that it will soon be triggered again). Much of our current work involves adding new capabilities to the program to detect and break out of such infinite loops, and to compile its experiences into one or more heuristics which would have prevented such situations from arising. It is not always easy to explain what is wrong with a certain 'bad product'. For instance, one newly synthesized heuristic kept rising in Worth, and finally I looked at it. It was doing no real work at all, but just before the credit/blame assignment phase, it quickly cycled through all the new concepts, and when it found one with high Worth it put its own name down as one of the creditors. Nothing is 'wrong' with that policy, except that in the long run it fails to lead to better results.

One additional factor which appears to have a dramatic effect upon the quality and rapidity of heuristic synthesis is the precise set of slots that are known to the system. This is the topic of Sections 4.2, 4.3, and 4.4.

4.2. Attributes of a heuristic

In AM, heuristics examine existing frame-like concepts, and lead to new and different concepts. To have heuristics operate on and produce heuristics, EURISKO represents each heuristic as a full-fledged frame-like concept. E.g., $H12$ (see Fig. 22) needs to reset the value of the Worth slot (attribute) of the concept f it operates on, hence even if f is a heuristic it must have a Worth slot (else we cannot run $H12$). Similarly, a heuristic that referred to such slots as Average-running-time, Date-created, Is-a-kind-of, Number-of-instances, etc. could only operate upon units (be they mathematical functions or heuristics) having such slots.

Fig. 24 illustrates (some of the slots from) a heuristic from EURISKO. Notice its similarity to the representation of a mathematical operation (Fig. 5). The heuristic resembles the math function (compare Figs. 24 and 5) much more than the math function resembles the static math concept (compare Figs. 5 and 6).

Earlier we defined a heuristic to be a contingent piece of guidance knowledge: In some situation, here are some actions that may be especially fruitful, and here are some that may be extremely inappropriate. While some heuristics have pathological formats (e.g., algorithms which lack contingency; delta function spikes which can be succinctly represented as tables), most heuristics seem to be naturally stated as rules having the format 'if-*conditions*, then-

```
NAME:        Generalize-rare-predicate
ABBREVIATION:   GRP
STATEMENT
   English:  If a predicate is rarely true, Then create generalizations of it
   IF-just-finished-a-task-dealing-with:  a predicate P        \   THESE 3 ATTRIBUTES COMPRISE
   IF-about-to-work-on-task-dealing-with:  an agenda A        |--- IF-POTENTIALLY-RELEVANT
   IF-in-the-middle-of-a-task-dealing-with:  *never*           /
   IF-truly-relevant:   P returns True less than 5% of Average Predicate
   IF-resources-available:  at least 10 cpu seconds, at least 300 cells
   THEN-add-task-to-agenda:  Fill in entries for Generalizations slot of P
   THEN-conjecture:  P is less interesting than expected
                     Generalizations of P may be better than P
                     Specializations of P may be very bad
   THEN-modify-slots:   Reduce Worth of P by 10%
                        Reduce Worth of Specializations(P) by 50%
                        Increase Worth of Generalizations(P) by 20%
   THEN-print-to-user:   English(GRP) with "a predicate" replaced by P
   THEN-define-new-concepts:
CODED-IF-PART:   λ(P) ...   <LISP function definition omitted here>
CODED-THEN-PART:    λ(P) ...   <LISP function definition omitted here>
CODED-IF-THEN-PARTS:    λ(P) ...   <LISP function definition omitted here>
COMPILED-CODED-IF-THEN-PARTS:  #30875
SPECIALIZATIONS:   Generalize-rare-set-predicate
   Boundary-Specializations:  Enlarge-domain-of-predicate
GENERALIZATIONS:  Modify-predicate, Generalize-concept
   Immediate-Generalizations:    Generalize-rare-contingent-piece-of-knowledge
   Siblings:      Generalize-rare-heuristic
IS-A:   Heuristic
EXAMPLES:
   Good-Examples: Generalize Set-Equality into Same-Length
   Bad-Examples:   Generalize Set-Equality into Same-First-Element
CONJECTURES:   Special cases of this are more powerful than Generalizations
   Good-Conjec-Units:  Specialize, Generalize
ANALOGIES:   Weaken-overconstrained-problem
WORTH: 600
VIEW:  Enlarge-structure
ORIGIN:  Specialization of Modify-predicate via empirical induction
   Defined-using:  Specialize
   Creation-date:  6/1/78  11:30
HISTORY:
   NGoodExamples:    1             NBadExamples:      1
   NGoodConjectures:  3            NBadConjectures:   1
   NGoodTasks-added:  2            NBadTasksAdded:    0
   AvgCpuTime:  9.4 seconds        AvgListCells:      200
```

FIG. 24. Frame-like representation for a heuristic rule for AM. The rule is composed of nothing but attribute:value pairs.

actions'. As the body of heuristics grows, the conditions fall into a few common categories (testing whether the rule is potentially relevant, testing whether there are enough available resources to expect the rule to work successfully to completion, etc.). The actions of the rules also begin to fall into a few common categories (add new tasks to the agenda, print explanatory messages, define new concepts, etc.). Each of these categories is worth making into a separate named attribute which heuristic rules can possess; Sections 4.3 and 4.4 will show the power which can arise from drawing such distinctions.

So instead of a heuristic having simply an if-slot and a then-slot, it has a bundle of slots which together comprise the conditions of applicability of the heuristic, and another bundle of slots which comprise the actions (see Fig. 24). In addition, there are several non-executable slots that describe the heuristic, that facilitate indexing of it, that relate it to other heuristics, etc.

By a 'slot' of a unit, we mean something closely related to the standard attribute/value pairing provided by property lists in LISP. However, there is no requirement that the value for the slot actually be stored explicitly; rather, we require only that it be retrievable upon demand. Thus our system, EURISKO, has a slot called Compiled-Coded-If-Then-Parts; no rule ever explicitly writes a value on such a slot, but some rules (such as those which define a rule interpreter) access such slots and EVAL them. When one is accessed, and found to be nonexistent, the unit called Compiled-Coded-If-Then-Parts is fetched, and its Definition is found. That definition says to access the Coded-If-Then-Parts slot, and then run the LISP compiler on that value. But suppose the Coded-If-Then-Parts slot doesn't exist, either; then its definition is consulted. That results in the Coded-If-Part and the Coded-Then-Part being accessed, and their values being put together into a Conditional expression. The Coded-If-Part doesn't exist, and the Definition slot of the unit called Coded-If-Part says to access—and conjoin—all the slots called If-Potentially-Relevant, If-Truly-Relevant, If-Resources-Available, etc. This looking up of slots' definitions continues until the only slots called for are ones which are *primitive*, which are actually stored on the property list of the unit. This is reminiscent of macro expansion, but more semantically guided; ontologically it actually has closer kinship to the style of knowledge-based automatic program synthesis done by Balzer, Barstow, Green, and others. See [7] for the origins of this paradigm, and [10] for more details of this malleable representation scheme.

One analogue of hardware caching is to store the *virtual* slot's values as they are computed; thus the property list of Generalize-Rare-Predicate might eventually look like that shown in Fig. 24, even though very few of those slots had their values stored there explicitly. Should the If-Truly-Relevant slot of Generalize-Rare-Predicate ever change, the system automatically updates the virtual slots defined using If-Truly-Relevant (in EURISKO, this currently would include If-Relevant, If-Parts, Coded-If-Parts, If-Then-Parts, Coded-If-Then-Parts, and Compiled-Coded-If-Then-Parts.)

These two features—software caching of slot's values, plus the ability to have virtual slots defined in terms of more primitive ones—lead to the dynamic expansion of the vocabulary of legal slots. Thus the original EURISKO system had heuristics with primitive Coded-If-Part and Coded-Then-Part slots; these were later given definitions in terms of new, more primitive slots (such as Then-Define-New-Concepts). Any existing rule, which had only the Coded-If-Part and Coded-Then-Part lumps of code, still runs for all purposes. All rules which ask for either of those slots still run. But new rules have the option of being specified in terms of more refined slots, and their Coded-If-Part and Coded-Then-Part slots are assembled upon demand out of those smaller pieces.

All the previous attributes (If- parts, Then- parts, Coded-... parts) have been *effective*, executable conditions and actions. These are paramount, since they serve to define the heuristic—they are the *criterial* slots. Many non-effective non-criterial slots are important as well, for describing the heuristics. Some of these relate the heuristic to other heuristics (Generalizations, Specializations), to classes of heuristics (Isa), and to non-heuristic concepts (View). Several slots record the heuristic's origins (Defined-using, Creation-date) and the case studies of its uses so far (Examples).

Once a rich stock of slots is present for heuristics, several new ones can be derived from them by choosing an n-ary relation R, and n slot names, and defining $R(S_1, S_2, \ldots, S_n)$ as a new type of slot.

First, consider choosing just a single kind of slot (e.g., Examples), and asking some questions about it: how does it evolve over time in length? what relationships exist among entries that fill it? how useful are those values?, etc. Each such question spawns a new kind of slot, e.g., AvgNumberOfExtremeExamples, RelnsAmongMyExtremeExamples, AvgWorthOfExtremeExamples. In EURISKO, these are thought of, and implemented, as full-fledged slots in their own right, *not* as subparts of slots. In our program, the various if- slots have not been relegated to second-class citizenship beneath Coded-If-and-Then-Parts. Indentation (in Figs. 5, 6, 24) is used merely as a visual aid, not to reflect extra levels of parentheses in LISP.

We now have an ad hoc way in which to generate new kinds of slots out of old ones. To accomplish this in a *principled* way, one would draw a flowchart of the primitive slots functions (Get, Put, Assert, etc.), and categorize—for each kind of flow chart primitive—what 'questions' one can ask about it. Thus, for a flowchart arrow that symbolizes a Write, one could ask about the old value, the new value, the amount of time the old value was present, the source of the new value, etc. More complex slots (such as average length of entries written) could be defined from these more elementary records. The above method focused on $R(S)$, i.e. on slots defined by asking unary questions about other slots, but the method generalizes:

One can take a pair of slots (say ThenConjecture and If-Truly-Relevant) and a relation (such as Implies) and define a new unary function on heuristics—a new kind of slot that any heuristic can have—where H_i would list H_j as an entry on that slot only if (in the present case) the ThenConjecture slot of H_i Implies the IfTrulyRelevant slot of H_j. A good name for this new slot might be 'CanTrigger', because it lists some heuristics which might trigger when H_i is fired.

If there are n slots, and m binary relations then this technique generates a space of mn^2 'cross-term' type slots. Naturally most of them won't be very useful, but this provides a *generator* for a large space of potentially worthwhile new slots. (This space is actually infinite, as $n \leftarrow mn^2$ after an exhaustive application of this process, and one must start all over again.) Some heuristics guide EURISKO in selecting plausible ones to define, monitoring the utility of each selection, and obliterating any losers (slots which, empirically, fail to facilitate the statement of or discovery of a highly-rated concept of any type). An excerpt from EURISKO illustrating this process is given in Section 4.3.

Again, there is nothing magical about the number 'two', and one could pick an n-ary relation R and n slot names, and use them all to build a new slot, as mentioned in the first paragraph of this subsection. 'Two' is slightly special, though, in that 'a kind of slot whose values are names of units'—such as 'Is-a'—is actually a binary relation, i.e., a subset of Units $\times$ Units.

4.3. Discovering a new heuristic

The heuristics present in AM and EURISKO create new concepts via specializing existing ones, generalizing (either from existing ones or from newly-gathered data), and analogizing. These are the three 'directions' new heuristics will come from. We have exemplified Specialization already. One point about Generalization is worth making: Heuristics which serve as plausible move generators originate by generalizing from past *successes*; heuristics which prune away implausible moves originate by generalizing from past *failures*. Since successes are much less common than failures, it is not surprising that most heuristics in most heuristic search programs are of the pruning variety. In fact, many authors define heuristic to mean nothing more than a pruning aid.

One of the typical 'common sense number theory' heuristics which AM lacked was the one which decides that the unique factorization theorem is probably more significant than Goldbach's conjecture, because the first has to do with multiplication and division, while the latter deals with addition and subtraction, and Primes is inherently tied up with the former operations.

How could such a heurstic be discovered automatically? This is the starting point for the example we now begin, an example which concludes in the following Section 4.4. What is the tie between these two sections? That is, what in the world does discovering heuristics have to do with representation of knowledge? The connection is much deeper than we originally suspected.

Consider just the special case where we restrict our representations to frame-like ones. The larger the number of different kinds of slots that are known about, the fewer keystrokes are required to type a given frame (concept, unit) in to the system. For instance, if NGoodConjecs were not known, it might take 40 keystrokes rather than 1 to assert that 'there are 3 good conjectures known involving prime numbers.' Moreover, no special-purpose machinery to process such an assertion would be known to the system. The larger your vocabulary, the shorter your messages can be.

This is akin to the power Interlisp derives from the thickness of its manual, from the huge numbers of useful predefined functions. A broad vocabulary streamlines communication. Not only does a profusion of slot types facilitate entering (typing in) a concept, it makes it easier to modify it once it's entered. This is because (i) fewer keystrokes are needed in toto, and (ii) the possible kinds of things you might need to type in are explicitly presented to you (in a menu).

Not only does a profusion of slot types facilitate *entering* a concept and *modifying* a concept, it makes it easier to *discover* new concepts—in particular new heuristics—because (i) it is a process of combining terms in a more powerful, higher level language, and (ii) specialized knowledge may exist, rules which refer to particular slots of heuristics, telling when and how the combination process should be done.

We are thus claiming that the task of discovering heuristics can be profoundly accelerated—or retarded—by the choice of slots we make for our representation. In the case of an excellent choice of slots, a new heuristic would frequently be simply a new entry on one slot of some concept. Let's see how that can be.

Recall that primes were originally discovered by the AM system as extrema of the function 'Divisors-of'. This was recorded by placing the entry 'Divisors-of' in the slot called 'Defined-using' on the concept called 'Primes' (see Fig. 6). Later, conjectures involving Primes were found, empirically-observed patterns connecting Primes with several other concepts, such as Times, Divisors-of, Exponentiation, and Numbers-with-3-divisors. This is recorded on the Good-ConjecUnits slot of the Primes concept. Notice that all the entries on Primes' DefinedUsing slot are also entries on its GoodConjecUnits slot. This recurred several times while running EURISKO, that is for several concepts besides Primes, and ultimately the heuristic $H19$ (Fig. 25) became relevant (its if- part became satisfied). The notation $u \cdot r$ means slot r of unit u.

$H19$ said that it would probably be productive to pretend that DefinedUsing was always a subslot[2] of GoodConjecUnits. I.e., $H19$ applied in the current situation, with $r =$ Defined Using and $s =$ GoodConjecUnits. It created a new heuristic, whose effect was the following: "As soon as Eurisko defines any new concept X in terms of Y, it should expect there to be some interesting conjectures between X and Y." In our usual if/then-format we might express this rule the way that $H20$ is worded (Fig. 25).

[2]Our usage of the term *subslot* is drawn from subset, subgroup, etc.; namely, r is a subslot of s iff (for all concepts u) any entry on $u \cdot r$ is also a valid entry one could place on $u \cdot s$. So Extreme-examples is a subslot of Examples, since any extreme example of a concept u is also an example of u. Mother is a subslot of Parent. Subslot is a subslot of Specializations. Another way to formulate this is to say that, for every concept u, the legal entries for its r slot are a subset of the legal entries for its s slot. The inverse of the subslot relation is called *superslot*. Unlike some uses of these words, the fact that one slot is a superslot of another has no bearing on how it is stored, retrieved, etc., nor on whether one is primitive and the other virtual.

$H19$: **if** (for many units u) most of the entries on $u \cdot r$ are also on $u \cdot s$,
then-assert that r is a subslot of s (with justification $H19$)

$H20$: **if** a concept u is created with a value in its DefinedUsing slot,
then place that value in u's GoodConjecUnits slot (justif $= H19$)

FIG. 25. A heuristic which notices and conjectures a containment relationship between slots, followed by one of the fruits of its labors—a new heuristic.

There is already a very general rule in the system, which says to verify suspected members of any slot (members whose justification is questionable). When $H20$ appears in the system, and is used to add suspected entries to the GoodConjecUnits slots of units, this general rule will cause tasks to appear on the Agenda, tasks which try to confirm or deny whether they deserve to be there.

The main point here is that $H20$ was not synthesized as a long, complicated expression such as shown above in Fig. 25. Rather, all EURISKO did was to go to the concept called DefinedUsing (the data structure which holds all the information the program knows about that kind of slot in general), and record that one of its Superslots is GoodConjecUnits. In other words, it added one atom to one list. EURISKO also gave this an explicit justification, namely $H19$, since it is a heuristic, not a fact. That required a second trivial action at the LISP level. Fig. 26 shows what this record looks like currently in EURISKO. The 'new heuristic' is simply the first word which is emboldened below; all the non-bold text was present in the program already (though most of it was written by the program itself at earlier times, not filled in by human hands). The second emboldened word gives the epistatus (epistemological status) of the new heuristic—namely, it is a heuristic and owes its existence to the speculations of heuristic $H19$.

Thanks to the large number of useful specialized slots, large if/then- rules

NAME: Archetypical-"Defined-Using"-slot
SPECIALIZATIONS:
  SubSlots: Really-Defined-Using, Could-Have-Defined-Using
GENERALIZATIONS:
  SuperSlots: Origin, **GoodConjecUnits**
    Justification: **H19**
IS-A: Kind of slot
WORTH: 300
ORIGIN: Specialization of Origin
  Defined-using: Specialize, Origin
  Creation-date: 9/18/79 15:43
AVERAGE-SIZE: 1
FORMAT: Set
FILLED-WITH: Concepts
JUSTIFICATION: Formal
CACHE? Always-Cache
MAKES-SENSE-FOR: Concepts

FIG. 26. Part of the concept containing centralizing knowledge about all DefinedUsing slots.

can be compactly, conveniently, efficiently represented as simple links. Some of these useful slots are very general, but many are domain dependent. Thus, as new domains of knowledge emerge and evolve, new kinds of slots must be devised if this powerful property is to be preserved. The next natural question is, therefore, "How can useful new slots be found?" The last two sentences are the final two points of our original five-point programme, and the next section answers them by way of continuing the example we've begun in this section.

To reiterate: EURISKO has already almost a thousand separate kinds of slots, most of which are defined using other slots, all of which were useful at some time or times. As a result of this large vocabulary of useful slot types, many entire heuristics can be recorded succinctly as a single atom or two placed in the right slot. Heuristic $H20$ was added to the program (by the program itself) merely by adding the atom GoodConjecUnits to the slot called SuperSlots of the unit called Archetypical-'Defined-Using'-slot.

It is important to make clear that the semantics of a value v appearing as an entry on slot s of concept c does *not* necessarily mean that it is formally proven that v merits a position there; rather, it is merely plausible. Any entry v can have an explicit justification, but in lieu of any information to the contrary, the default justification is merely empirical. Thus, when an entry, say Palindromes, is on the GoodConjecUnits slot of Primes, it may mean that some interesting conjectures have been found between Primes and Palindromes, or just that it is suspected—and expected—that such conjectures can be found if one spends the trouble looking for them.

How does the EURISKO program know what the justification of a slot is, if it isn't explicitly recorded? It goes to the unit for the archetypical representative of that slot, looks up a slot called Justification, and retrieves that value. In the case of the Defined-Using slot, there is almost never any question of uncertainty about its values—the definition of one slot in terms of another has to be spelled out in black and white. Therefore, as Fig. 26 shows, the Justification slot for the unit called Archetypical-'Defined-Using'-slot is filled with the entry 'Formal'. Things are not so clearcut for entries on most units' Worth slots, and therefore in the EURISKO system, on the Justification slot of the Archetypical-'Worth'-slot unit, there is no entry. Rather, by inheritance from the very high-level unit called Any-Slot, the justification for Worth values is determined to be 'Empirical'.

Thanks to the large number of useful specialized slots, thousands of heuristics which would be bulky if stated as if/then- rules can be compactly, conveniently, efficiently represented as simple links—as a single atom entered on the appropriate slot of the appropriate unit. Most of these useful slots are very general (e.g., Examples, Worth, SuperSlots), but some are domain dependent (e.g., Predators, Toxicity, HullArmor). Thus, as new domains of knowledge emerge and evolve, new kinds of slots must be devised if this powerful property is to be preserved. The next natural question is, therefore.

"How can useful new slots be found?" By way of answering those two questions, the next section continues—and concludes—the example we have begun in this section.

4.4. Heuristics used to extend existing representations

Each kind of representation makes some set of operations efficient, often at the expense of other operations. Thus, an exploded-view diagram of a bicycle makes it easy to see which parts touch each other, sequential verbal instructions make it easy to assemble the bicycle, an axiomatic formulation makes it easy to prove properties about it, etc.

As a field matures, its goals vary, its paradigm shifts, the questions to investigate change, the heuristics and algorithms to bring to bear on those questions evolve. Therefore, the utility of a given representation is bound to vary both from domain to domain and within a domain from time to time, much as did that of a given corpus of heuristics. The representation of today must adapt or give way to a new one—or the field itself is likely to stagnate and be supplanted.

Where do these new representations come from? The most painless route is to merely select a new one from the stock of existing representational schemes. Choosing an appropriate representation means picking one which lets you quickly carry out the operations you are now going to carry out most frequently.

In case there is no adequate existing representation, you may try to extend one, or devise a whole new one (good luck!), or (most frequently) simply employ *a set* of known ones, whose union makes all the common operations fast. Thus, when I buy a bicycle, I expect both diagrams and printed instructions to be provided. The carrying along of multiple representations simultaneously, and the concommitant need to shift from one to another, has not been much studied—or attempted—in AI to date, except in very tiny worlds (e.g., the Missionaries and Cannibals puzzle; graphics).

There are several levels at which 'new representations' can be found. At the lowest level, one may say that AM changed its representation every time it defined a new domain concept or predicate, thereby changing its vocabulary out of which new ones could be built. At the highest level would be true open-ended exploration in 'the space of all representations of knowledge'. The latter may someday be possible, but we currently lack adequate experience to formulate the necessary generation rules.

The example below lies intermediate between these two extremes: it shows how EURISKO discovers new kinds of slots which can be used to advantage. For instance, when AM found the unique factorization conjecture (UFT), it would have been helpful if AM had at that instant defined a new kind of slot, Prime-Factors, that every Number could have possessed. *H*21 is a EURISKO rule capable of this sort of second-level representation augmentation (see Fig. 27).

*H*21: **if** most units in the system have very large *s* slots (i.e., have many entries stored therein),
then propose a new task: replace *s* by new specializations of *s*

*H*22: **if** a slot *s* is very important, and all its values are units,
then-create-new-kind-of-slot which contains "all the relations among the values of my *s* slot"

FIG. 27. Heuristics which occasionally lead to new kinds of slots worth having.

The vague terms in the rule have specific computational interpretations, of course, in EURISKO; for instance, 'large' is coded as 'more than twice the average size of all slots, and also larger than the average number of slots a unit has'. In one experiment, the various types of examples (extreme, typical, boundary, etc.) were not given separate slots initially, but were unioned into huge Examples slots. The above rule then caused the program to focus on defining new specializations of Examples; recall that we term such specializations 'subslots', though this does not mean that they are implemented as pieces of their superslots; the old Examples slot still exists and has many entries, even if every one of those entries also exists on some subslot(s) of Examples. Note that the subslots will not in general be disjoint. In a more domain-dependent usage, the above rule causes Factors to be split up into PrimeFactors, OddFactors, LargeFactors, etc.

A slightly more advanced level at which 'new representations' are synthesized by EURISKO is to actually shift from one entire scheme to another—potentially novel—one. The first two rules in Fig. 28 indicate when a certain type of shift is appropriate. All the heuristics of this type are specializations of the third, general one, *H*25.

*H*23: **if** the problem is a geometric one,
then draw a diagram

*H*24: **if** most units have most of their possible slots filled in,
then shift from property lists to record structures

*H*25: **if** some operation is performed frequently,
then shift to a representation in which it is inexpensive to perform

FIG. 28. Heuristics which occasionally effect a change of representation.

Let us continue our example. *H*22 (shown above, in Fig. 27) is capable of reacting to a situation by defining an entirely new slot, built up from old ones, a new slot which it expects will be useful. When the number stored in the Worth slot of the GoodConjecUnits concept is large enough, the system attends to the task of explicitly studying GoodConjecUnits. Several heuristics are relevant and fire; among them is *H*22, the rule shown above. It then synthesizes a whole new unit, calling it RelationsAmongEntriesOnMy'GoodConjecUnits'Slot. Every known way in which entries on the GoodConjecUnits slot of a concept C relate to each other can be recorded on this new slot of C. In practice, this slot typically had only a few entries, for most units: only relations which were explicitly defined could be perceived and recorded therein (e.g., all the various types of slots), and EURISKO is not designed to spend its time in undirected searching for entries for that slot.

How was the new slot used by the program? Take a look at the Primes concept (Fig. 6). Its GoodConjecUnits slot contains the following entries: Times, Divisors-of, Exponentiation, Squaring, and Numbers-with-three-divisors. The first two of these entries are inverses of each others; that is, if you look over the Divisors-of unit you will see a slot called Inverse which is filled with names of concepts, including Times. Similarly, still looking over the Times unit, one can see a slot called Repeat which is filled with the entry Exponentiation, and one can see a slot called Compose filled with Squaring. So Inverse and Repeat and Compose are some of the relations connecting entries on the GoodConjecUnits slot of Primes, hence the program will record Inverse and Repeat and Compose as three entries on the RelationsAmongEntriesOnMy 'GoodConjecUnits'Slot slot of the Primes concept.

Now it so happens that several concepts wind up with 'Compose' and 'Inverse' as entries on their RelationsAmongEntriesOnMy'GoodConjecUnits' Slot slot. The alert reader may suspect that this is no accident, and an alert program should suspect that, too. Indeed, heuristic $H26$ (Fig. 29) says that it might be useful to behave as if 'Compose' and 'Inverse' were always going to eventually appear there. There is no formal justification behind this kind of *anticipation*, but it is cost-effective to follow such a policy; it is akin to the psychological phenomenon of expectation-filtering.

$H26$ causes EURISKO to add Compose and Inverse to the slot called ExpectedEntries of the concept called RelationsAmongEntriesOnMy'GoodConjecUnits'Slot. This one small act, the creation of a pair of links, is in effect creating a new heuristic equivalent to $H27$ (Fig. 29).

$H26$:	**if**	(for many units u) the s slot of u contains the same values v_i,
	then-add-value	v_i to the ExpectedEntries slot of the Typical-s-slot unit
$H27$:	**if**	a concept u gets entries X and Y on its GoodConjecUnits slot,
	then-predict:	u will also eventually get Inverse(X), Inverse(Y), and Compose (X, Y) there as well

FIG. 29. Entries which usually crop up on s slots can be expected to appear there. $H26$ says this, and one of its applications spawned a heuristic equivalent to $H27$ (but more compact).

How is this actually used? Consider what occurs when the program defines a new concept, *C*, which is DefinedUsing Divisors-of. As soon as that concept is formed, the heuristic link from DefinedUsing to GoodConjecUnits automatically fills in Divisors-of as an entry on the GoodConjecUnits slot of *C*. Next, the links just illustrated above come into action, and place Inverse and Compose on the RelationsAmongEntriesOnMy'GoodConjecUnits'Slot of *C*. That in turn causes the inverse of Divisors-of, namely Times, to be placed on the GoodConjecUnits slot as well as the already-present entry, Divisors-of. Finally, that causes the program to go off looking for conjectures between *C* and either multiplication or division. When a conjecture comes in connecting *C* to one of them, it will get a higher a priori estimated worth than one which does not connect to them.

If only we had the new heuristics back when Primes was first defined, they would have therefore embodied enough 'common sense' to prefer the Unique Factorization Theorem to Goldbach's conjecture. If we had them then, these heuristics would have led us to our present state much sooner. Because of our assumptions about the continuity of the world, such heuristics are still worth having and using—we expect them to be useful from time to time in the future.

Notice that there is nothing special about mathematics—the newly synthesized heuristics have to do with very general slots, like DefinedUsing and GoodConjecUnits. For instance, as soon as a new concept (say Middle-Class) is defined using the old slot Income, the program immediately fills in the bold printed information in Fig. 30.

NAME: Middle-Class
Defined-using: Income
RelationsAmongEntriesOnMy'GoodConjecUnits'Slot: **Inverse, Compose**
Good-Conjec-Units: **Income, Spending, EarnedInterest**

FIG. 30. A fragment of a non-math concept for which some predictions have been recorded (in boldface), thanks to the heuristics shown in Fig. 29.

Thus, EURISKO goes off looking for (and will expect more from) conjectures between Middle-Class and any of Income, Spending, and EarnedInterest. In one run of the EURISKO system, some such conjectures were then found (including 'MiddleClass spends all its income'), but we primed the system with very caricatured data about Americans' incomes and spending habits. When we removed heuristic *H*22, RelationsAmong ... slots never was defined, so *H*26 did not fire, so Income and Spending were not placed on the GoodConjecUnits slot of MiddleClass, and the preceding conjecture was never found. So the new slot *is* useful, though it has a terrible name, and the new little heuristics (which looked like little links or facts but were actually *permission to make daring guesses*) were powerful after all.

We have relied heavily on our representation being very structured; in a very uniform one (say a calculus of linear propositions, with the only operations being Assert and Match) it would be difficult to obtain enough empirical data to easily modify that representation. This is akin to the nature of discovering domain facts and heuristics: if the domain is too simple, it is harder to find new knowledge and—in particular—new heuristics. Heuristics for propositional calculus are much fewer and weaker than those available for guiding work in predicate calculus; they in turn pale before the rich variety available for guiding theorem proving 'the way mathematicians really do it'. This is an argument for attacking seemingly-difficult problems which turn out to be lush with structure, rather than working in artificial worlds so constrained that their simplicity has sterilized them of *heuristic structure.*

4.5. Recent results of the EURISKO program

Much of the preceding discussion has been quite abstract. In this section we present some of the concrete results produced by the EURISKO program so far, with some glimpses into how they were obtained. EURISKO has hundreds of units for six different domains: Set Theory, Number Theory, Oil Spill Amelioration, Device Physics, Games, Heuristics itself, and Representation itself.

4.5.1. *Results in the games domain*

As our first example, let us consider EURISKO's exploits in a Games task, exploring the design of naval fleets conforming to a body of (several hundreds of) rules and constraints as set forward in *Traveller: The Trillion Credit Squadron.* EURISKO designed a fleet of ships suitable for entry in the recent Origins national wargame tournament, held at Dunfey's Hotel, in San Mateo, CA, over July 4 weekend, 1981. The traveller tournament, run by Game Designers Workshop (based in Normal, Illinois), was single elimination, six rounds. EURISKO's fleet won that tournament, thereby becoming the ranking player in the United States (and also an honorary Admiral in the Traveller navy). This win is made more significant by the fact that the author had never played this game before, nor any miniatures battle game of this type, and there were no practice rounds.

Each participant has a budget of a trillion 'credits' (roughly equal to dollars) to spend in designing and building a fleet of futuristic ships. There are over one hundred pages of rules which detail various costs, constraints, and tradeoffs, but basically there are two levels of variability in the design process:

(1) Design an individual ship: worry about tradeoffs between types of weapons carried, amount of armor on the hull, agility of the vessel, grouping of weapons into batteries, amount of fuel carried, which systems will have backups, extra replacement crew carried, etc.

(2) After designing several distinct kinds of individual ships, group sets of them together into a fleet. The fleet must meet several design constraints (e.g.,

ships having a total fuel tonnage of at least 10% of the total fleet fuel tonnage must be capable of refueling and processing unrefined fuel), and in addition must function tactically as a coherent unit.

EURISKO was given the rules of the game, and the constraints on the design process, and spent a great amount of time (roughly 500cpu hours on a Xerox *Dolphin*) managing a heuristically-guided evolution process. Fleets would fight, and the simulated battle would be analyzed (by some of EURISKO's heuristics) to determine which design policies were winning, and—occasionally—what fortuitous circumstances could be abstracted into new design heuristics. An example of the former was when the Agility of ships gradually decreased, in favor of heavier and heavier Armor plating of the hulls. An example of the latter was when a purely defensive ship was included in an otherwise-awful fleet, and that fleet *could never be fully defeated* because that defensive ship, being very small, unarmored, and super agile, could not be hit by any of the weapons of the larger nearly-victorious fleet. The author culled through the runs of the program every 12 hours or so of machine time (i.e., each morning, after letting it run all night), weeding out heuristics he deemed invalid or undesirable, rewarding those he understood and liked, etc. Thus the final crediting of the win should be about 60/40% Lenat/EURISKO, though the significant point here is that neither Lenat nor EURISKO could have won alone. Most of the battles are tactically trivial, the contest being almost decided by the designs of the two fleets; that—and the thickness of the rulebooks—were the reason this appeared to be a valid domain for EURISKO.

One very general result EURISKO abstracted from this process was a 'nearly extreme' heuristic: In almost all Traveller fleet design situations, the right decision is go for a nearly—but not quite—extreme solution. Thus, the final ships had Agility 2 (slightly above the absolute minimum), one weapon of each type of small weapons (rather than 0 or many), the fleet had almost as many ships as it could legally have but not quite (93 instead of 95), etc. Our original intuitions were to have a moderate number of large ships, each with one enormous spinal mounted weapon capable of blasting another ship to pieces with a single shot; EURISKO gradually changed this into a fleet consisting of a large number of small ships, collectively bearing an enormous number of small missile weapons. The fleet had almost all (75) ships of this type, though there were a couple of ships which were small and agile, and a couple of ships which had weapons large enough to *destroy* any enemy ships that were small and agile.

Almost all the other entrants in the final tournament had fleets that consisted of about 20 ships, each with a huge spinal mount weapon, low armor, and fairly high agility, and a large number of secondary energy weapons (laser-type weapons). This contrasted with Eurisko's fleet, and such an enemy was rapidly decimated, at a loss of about a third of Eurisko's line ships, and no risk to its specialty ships.

Eurisko's first opponent resigned after one exchange of fire, when the pattern became clear. Its second opponent did some calculations and resigned without ever firing a shot. The subsequent opponents resigned during the second round. The few specialty ships remained unused until the final round of the tournament, battling for 1st vs. 2nd place. That opponent also had ships with heavy armor, few large weapons, low agility, etc. He was lacking any fast ships or fast-ship-killers, though. The author simply pointed out to him that if Eurisko were losing then we could put only our fast ship out in the front line, withdraw the others and repair them, and effectively start the battle all over again. This could go on until such time as Eurisko appeared to be winning, when we would let it continue to termination. The opponent did a few calculations and surrendered without fighting. Thus, while most of the tournament battles took 2–4 hours, most of those involving EURISKO took only a few minutes.

The tournament directors were chagrined that a bizarre fleet such as this one captured the day, and a similar fleet (though not so extreme) took second place. As a result, the rules for future years' TCS tournaments have been changed, to dramatically reduce the design singularities which EURISKO (working with the author) found.

4.5.2. *Results in programming and representation*

A few hundred of the most common INTERLISP function have been represented as units within EURISKO. This enables it to monitor and modify its own behavior, as well as synthesize and modify new LISP functions. EURISKO gathers data about LISP, just as it does about elementary mathematics, or games. For example, EURISKO was originally given units for EQ and EQUAL, with no explicit connection recorded between them. Eventually, it got around to recording examples (and nonexamples) for each, and conjectured that EQ was a restriction (a more specialized predicate) of EQUAL, which is true. A heuristic suggested disjoining an EQ test onto the front of EQUAL, as this might speed EQUAL up. Surprisingly (to the author, though not to EURISKO), it did! This turned out to be a small bug (since fixed) in the then-extant LISP. Once it had the conjecture about EQ being a special kind of EQUAL, it was able to look through its code and specialize bits of it by replacing EQUAL by EQ, or to generalize them by substituting in the reverse order. EURISKO analyzed the differences between EQ and EQUAL, and came up with the concept we refer to as LISP atoms. In analogue to humankind, once EURISKO discovered atoms it was able to destroy its environment (by clobbering CDR of atoms).

4.5.3. *Results in device physics*

After discussion with Bert Sutherland and Jim Gibbons, it was decided that one domain ripe for attack by an exploration system such as EURISKO is the design of VLSI circuits. We began by building in a knowledge base of concepts such as groups in the periodic table, doping, carriers, annihilation, etc. Thanks to a very carefully chosen vocabulary of nouns (n-doped-region, p-doped-region, insulator) and verbs (abut-regions, apply-electric-field), many elementary physical devices were trivial, short 'sentences' in that language (e.g., the silicon diode, MOSFET transistor). That was not particularly surprising, and further discussion caused us to go up a level of abstraction, and discuss the *conduction paths* in a circuit, rather than the specific behavior of charged carriers moving through various types of materials.

Thus, considering only NMOS technology, one can view a transistor simply as a gate which, when 'on', allows current to flow between its two terminals. In other words, emitter and collector are symmetric in NMOS, though most designs try to be less technology-dependent and don't take advantage of that symmetry as EURISKO did. Recent advances in polysilicon recrystallization fabrication techniques make it feasible to design three-dimensional VLSI devices, a series of alternating layers of (i) metal and insulator material, (ii) doped semiconductor material and channel material. EURISKO explored various configurations of these materials, using a few heuristics to constrain its search through this enormous space. Some heuristics constrain the generation process, others recognize interesting or useful functionality—or inconsistencies—in the resultant candidate circuit. Almost immediately it happened upon a surprisingly tiny design for a memory cell; see Fig. 31b for an illustration of that design, and Fig. 31a for the corresponding two-dimensional circuit diagram of

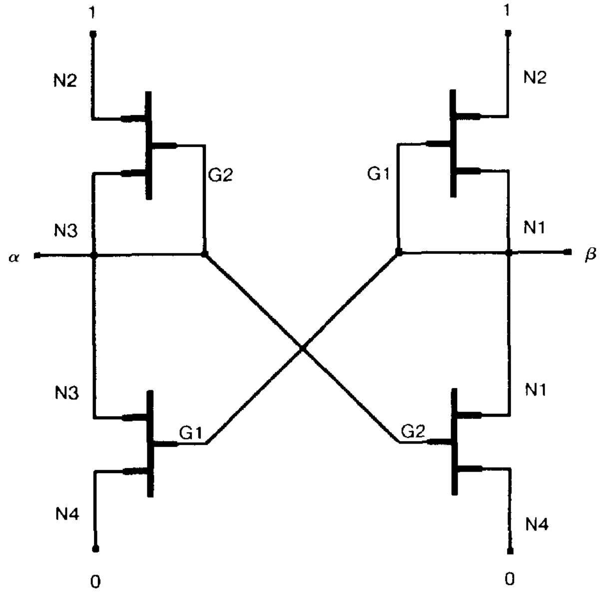

FIG. 31a. A conventional circuit for a four-transistor memory cell. Pulsing α causes α to stay at 1, and β to drop to 0. Similarly, pulsing β drives it to 1 and drives α to 0. Each state change is permanent, at least until the next pulse is received at α or β. When gate G_1 is on, it connects n_1 to n_2, and it also connects n_3 to n_4; when gate G_2 is on, it connects n_1 to n_4, and it connects n_3 to n_2.

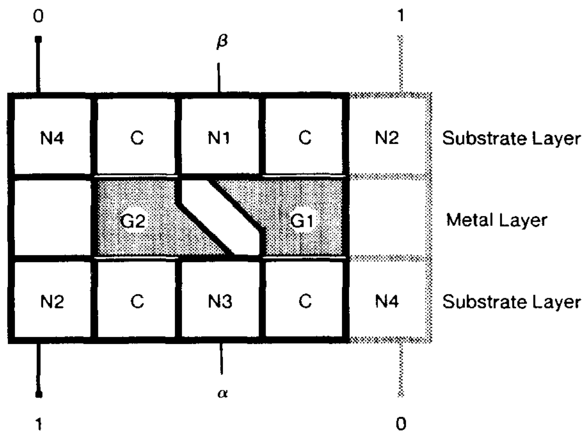

FIG. 31b. A side view of EURISKO's very compact design for building the same memory cell in 3-dimensional NMOS, specifically by using one metal layer sandwiched between two substrate layers. Shaded regions represent metal; *n* stands for n-doped semiconductor material; *c* regions represent channel material with thin oxide coatings; clear regions in the middle (metal) layer represent insulator. By co-identifying the left and rightmost columns, after giving the device a 'half-twist' in 3-space, we have a design which is realizable on the surface of a Möbius strip, and implements the memory cell in a mere dozen 'tiles'.

its functionality. Alas, the cell can be realized most efficiently on the surface of a Möbius strip, hence its commercial future may be dim. It *does* serve to illustrate the potential richness of this domain, as yet unexplored by human beings, and the potential richness of the EURISKO approach. One of the most promising uses for programs which automatically glean heuristics is in fields which are just opening up, in which people have not yet mined many of the powerful heuristics. By now, several useful new concepts have originated from the device physics line of research with EURISKO, including a higher functional level of abstraction for describing circuits quite distinct from 'sticks' diagrams (involving terms like 'potentially-connected-to' and n-pole 'abstract gates').

4.5.4. *Results in heuristics*

EURISKO has synthesized many new heuristics using the techniques presented in Sections 4.1-4.4. Sometimes this has occurred as a by-product of other activities, during the course of working in some particular task domain. Additionally, EURISKO has from time to time chosen to focus on the domain of Heuretics explicitly for a while. Among the new heuristics discovered so far by the program are:

Several specific to gaming (such as, "In Traveller, prefer armor to agility", or "In designing a military force, go nearly but not quite to extremes").

Several specific to mathematics (such as, "If an inverse function is going to be used even once, then it's probably worth it to search for a fast algorithm for computing it").

Several specific to programming (such as, "If you can use EQ instead of EQUAL, do it to save time", or "Sometimes 'and' means 'do in sequence', and sometimes it means 'do simultaneously', and those two cases are important to distinguish before you consider generalizing or specializing a piece of code").

A few specific to heuristics itself (such as, "If you're generalizing a heuristic, then avoid changing the main connective of the premises of the heuristic from 'and' to 'or'; it is a generalization but it leads to terrible results such as infinite loops and LISP errors").

Many additional heuristics have been created synergistically, with credit to both EURISKO and one or more humans working with that program. For example:

- In Traveller, having a small agile ship might give you an infinite 'restart' capability for the battle (this made the difference between 1st and 2nd place at the tournament).
- When designing three-dimensional VLSI devices, in alternating layers of metal and semiconductor material, have the 'wires' run North–South in odd metal layers, and East–West in even metal layers (amusingly, this is an analogue of an ancient TTL heuristic).
- When folding a 2-dimensional NMOS design into 3 dimensions, look for two gates whose controls are identical, and implement them as a single '2-pole' gate controlling regions both above and below it' (compare Figs. 31a and 31b to see how four transistors were replaced by only two gates).

5. Conclusions

The field of Heuretics was proposed as a promising one for AI to investigate, one which may aid us in understanding—and constructing—expert systems. We began by defining what it meant for something to be a scientific discipline, and showing that Heuretics met these criteria.

Heuretics asks "What is the source of power of heuristics?", to which our first-order reply is: "Behave as though APPROPRIATENESS(Action,Situation) were time-invariant and continuous in both variables." Heuristic search is adequate for modeling worlds which are *observable* (so heuristics can be formed), *stable* (so heuristics abstracted from past experiences will be useful in the future), and *continuous* (so that if A was (in)appropriate in S, then actions similar to A will be (in)appropriate in situations similar to S). Corollaries of this provide the justification for the use of analogy, generalization, and even for the utility of memory. The central assumption was seen to be just that—an assumption. It's often false in small ways, but nevertheless the central assumption has proven itself to be a useful fiction to be guided by.

Using the metaphor of APPROPRIATENESS being a function, we considered graphing the power curves of a heuristic (the utility of that heuristic as a function of task being worked on), and were able to see the gains—and dangers—of specializing and generalizing heuristics to get new ones. Consideration of such curves led us to an algorithm for deciding in which order to obey relevant heuristics, and suggested several specific new attributes worth measuring and recording for each heuristic (e.g., the sharpness with which it flips from useful to harmful, as one leaves its domain of relevance).

By arranging all the world's heuristics (well, at least all of AM's, and later several more from chess, biological evolution, naval fleet design, device physics, plumbing, game-playing, and oil spills) into a hierarchy using the relation 'More-General-Than', we were surprised to find that hierarchy very shallow, thereby implying that analogy (a side-to-side operation) would be more useful a method of generating new heuristic than would specialization or generalization (up-and-down operations). By noting that both Utility and Task have several dimensions, most of this 'shallow-tree' problem went away. By noting that two heuristics can have many important relations connecting them, of which More-General-Than is just one example, the shallowness problem turns into a powerful heuristic: if a new heuristic h is to differ from an old one along some dimension (relation) r, then use analogy to get h if r's graph graph is shallow, and use generalization/specialization if r's graph is deep. We also discussed some useful slots which heuristics can have, and a principled method for generating new kinds of slots.

Heuretics asks "How do new heuristics originate?", to which we recursively reply: "By generalizing other heuristics, abstracting from data, specializing other heuristics, finding analogies to other heuristics and to processes whereby other heuristics were formed." EURISKO demonstrated that these processes themselves can be guided adequately by a corpus of heuristics, that there is no need to distinguish such 'meta-heuristics' from 'object-level heuristics', and—surprisingly to us—that analogy has as much potential as generalization or specialization. In more detail: AM demonstrated the adequacy of the heuristic search paradigm to guide a program in formulating useful new concepts, gathering data about them, and noticing relationships connecting them. However, as the body of domain-specific facts grew, the old set of heuristics became less and less relevant, less and less capable of guiding the discovery process effectively. New heuristics must also be discovered.

EURISKO was developed as the successor system, one whose field of expertise was not mathematics, or diagnosis, but rather Heuretics. That is, EURISKO had a corpus of heuristics which, as they ran, gathered data about their own running, and synthesized new members of that corpus (and modified old ones). As expected, this process was very slow and explosive. By taking the four best (in EURISKO's judgment) synthesized heuristics, and rerunning the program from scratch, almost an order of magnitude improvement in performance was obtained (a factor 7 in the number of tasks executed, a factor of 8 in the number of losing heuristics synthesized, a factor of 4 in the cpu time involved, and a factor of 9 in the storage cells used). The explosive process of synthesizing heuristics was made feasible only by having 'the right representation'. EURISKO, like AM, used a schematized representation, so the right representation meant having a large repertoire of very useful kinds of slots.

We saw how, in EURISKO, heuristics led to the development of useful new kinds of slots, and to improved representations of knowledge. Note that the same representation AM used for attributes and values of object-level math concepts was also used to represent heuristics and even to represent representation. E.g., Primes (a set of numbers), GeneralizeRarePredicate (a heuristic), GeneralizeRareHeuristic (a meta-heuristic), and DefinedUsing (a representation concept) are all represented adequately as concepts (units with slots having values). Since meta-heuristics are not distinguished from heuristics, a single interpreter of necessity runs both types of rules, and is itself represented as a collection of units (and dynamically redefinable). While meta-heuristics could be tagged to distinguish them from heuristics, the utility of doing so rests on the existence of rules which genuinely treat them differently somehow—and few such rules have to date been encountered. Finally, we surveyed some recent results obtained by running EURISKO in several different domains outside mathematics, namely: wargames, programming, and circuit design.

To advance the Heuristics research programme, much more must be known about analogy, and more complete theories of heuristics and of representation must exist. Toward that goal we must obtain more empirical results from programs trying to find useful new domain-specific heuristics and representations.

ACKNOWLEDGMENT

Productive discussions with John Seely Brown, Bruce Buchanan, Bill Clancey, Ed Feigenbaum, Johan deKleer, John Doyle, George Polya, Mark Stefik, and Mike Williams have heavily influenced this work. Danny Bobrow, Bruce Buchanan, Bill Clancey, and Russ Greiner provided valuable critiques of earlier versions of this paper, which have led to substantial changes in its organization and content. Section 2 presents lessons learned from AM, from which I thank Bruce Buchanan, Ed Feigenbaum, Cordell Green, Don Knuth, and Allen Newell. The data for Section 3.4's 'shallowness' conclusion about the tree of heuristics was gathered while I was at CMU, with the aid of Herb Simon and Woody Bledsoe. Much of Section 4 relies upon RLL, a self-describing and self-modifying representation language constructed by Russ Greiner and the author. The three-dimensional VLSI design work described in Section 4.4.3 is being pursued in collaboration with Bert Sutherland and Jim Gibbons. Finally, I wish to thank XEROX PARC'S CIS and Stanford University's HPP for providing superb environments (intellectual, physical, and computational) in which to work. Financial support is provided by ONR (N00014-80-C-0609) and XEROX.

REFERENCES

1. Barr, A. and Feigenbaum, E.A. (Eds.), *Handbook of Artificial Intelligence*, Vol. II (Kaufman, Los Altos, 1981).
2. Seely Brown, J. and VanLehn, K., Repair theory: A generative theory of bugs in procedural skills, *J. Cognitive Sci.* **4** (4) (1981).

3. Clancey, W.J., Dialogue management for rule-based tutorials, *Proc. Sixth Internat. Joint Conference on Artificial Intelligence*, Tokyo, 1979.
4. Davis, R. and Lenat, D., *Knowledge Based Systems in Artificial Intelligence* (McGraw-Hill, New York, 1982).
5. Feigenbaum, E.A., Knowledge engineering: The practical side of artificial intelligence, HPP Memo, Stanford University, Stanford, CA, 1980.
6. Gaschnig, J., Exactly how good are heuristics?: Toward a realistic predictive theory of best-first search, *Proc. Fifth Internat. Joint Conference on Artificial Intelligence*, Cambridge, MA, 1977.
7. Green, C., Waldinger, R. Barstow, D., Elschlager, R., Lenat, D., McCune, B., Shaw, D. and Steinberg, L., Progress report on program understanding systems, AIM-240, STAN-CS-74-444, AI Lab, Stanford, CA, 1974.
8. Hayes-Roth, F., Waterman, D. and Lenat, D. (Eds.), *Building Expert Systems, Proc. 1980 San Diego Workshop in Expert Systems* (Addison-Wesley, Reading, MA, 1982).
9. Lenat, D.B., On automated scientific theory formation: A case study using the AM program, in: J. Hayes, D. Michie and L.I. Mikulich (Eds.), *Machine Intelligence* **9** (Halstead, New York, 1979) 251–283.
10. Lenat, D.B., and Greiner, R.D., RLL: A representation language language, *Proc. First Annual Meeting of the American Association for Artificial Intelligence* (AAAI), Stanford, CA, 1980.
11. Minsky, M., Steps toward Artificial Intelligence, in: Feigenbaum and Feldman (Eds.), *Computers and Thought* (McGraw-Hill, New York, 1963).
12. Newell, A. and Simon, H., Computer science as empirical inquiry: Symbols and search, *Comm. ACM* **19** (3) (1976).
13. Poincaré, H., *The Foundations of Science* (The Science Press, New York, reprinted, 1929).
14. Polya, G., *How to Solve It* (Princeton University Press, Princeton, NJ, 1945).
15. Pushkin, V.N. (Ed.), *Problems of Heuristics* (Keter, Jerusalem, 1972).
16. Simon, H.A., *The Science of the Artificial* (MIT, Cambridge, MA, 1969).
17. Winston, P.H., Learning structural descriptions from examples, Project MAC TR-231, MIT AI Lab, Cambridge, MA, 1970.

Received November 1980; revised version received April 1981