Section II-D

ρ = distance function in state space (pseudo-metric).
$'$ = transpose of the matrix.
Q = positive semidefinite matrix.
$(\ )^*$ = equilibrium values.
$(\ )^0$ = optimal values.

Section II-E

$\mathcal{P}$ = performance index.
h; h_i = optimal control function.

Section III.

x; x_i = incremental state vector; incremental state variables.
m; m_i = incremental control vector; incremental control variables.
F = infinitesimal transition matrix in linear case.
D = matrix denoting instantaneous effect of control variables in linear case.
$\Phi(\tau)$ = (finite-interval) transition matrix in linear case.
$\Delta(\tau)$ = matrix denoting effect of control variables in linear case (finite-interval).
A_N, B_N, P_N, R_N, S_N constant matrices.

Simulation of Human Problem-Solving

W. G. BOURICIUS† AND J. M. KELLER†

SIMULATING human problem-solving on a digital computer looks deceptively simple. All one must do is program computers to solve problems in such a manner that the computer employs the identical strategies and tactics that humans do. This will probably prove to be as simple in theory and as hard in actual practice as was the development of reliable digital computers. One of the purposes of this paper is to describe a few of the pitfalls that seem to lie in the path of anyone trying to program machines to "think."

The first pitfall lies in the choice of an experimental problem. Naturally enough the problem chosen should be of the appropriate degree of difficulty, not so difficult that it cannot be done, and not so trivial that nothing is learned. It should also involve symbology and manipulations capable of being handled by digital computers. At this stage of problem consideration, a devious form of reasoning begins to operate. Usually the people engaged in this type of research will have had a thorough grounding in conventional problem-solving on computers. Consequently, they are conversant with the full range of capabilities of computers and have an appreciation of their great speed, reliability, etc. They also know what kinds of manipulations computers do well, and conversely, what kinds of things computers do in a clumsy fashion. All of this hard-earned knowledge and sophistication will tend to lead them astray when the time arrives to choose a problem. They will try to make use of this knowledge and hence choose a problem that will probably involve the simulation of humans solving problems with the aid of computers rather than the simulation of humans solving problems with only paper and pencil. Consequently, the characteristics of present-day computers may confine and constrict the area of research much more than is desirable or requisite. What is liable to happen, and what did happen to us, is that the experimental problem chosen will develop into one of large size and scope. If this always happens, then those human manipulative abilities that are presently clumsy and time-consuming on computers will never get programmed, simulated, or investigated. Fortunately for us, the two experimental problems we chose were of such a nature that they could be easily miniaturized, and this was done as soon as the desirability became apparent.

The second pitfall which must be avoided is the assumption that one knows in detail how one thinks. This delusion is brought about by the following happenstance. People customarily think at various levels of abstraction, and only rarely descend to the abstraction level of computer language. In fact, it seems that a large share of thinking is carried on by the equivalent of "subroutines" which normally operate on the subconscious level. It requires a good deal of introspection over a long period of time in order to dredge up these subroutines and simulate them. We believe people assume that they know the logical steps they pursue when solving problems, primarily because of the fact that when two humans communicate, they do not need to descend to the lower levels of abstraction in order to explain to each other in a perfectly satisfactory way how they themselves solved a particular problem. The fact that they are likely to have very similar "subroutines" is obvious and also very pertinent.

† IBM Res. Center, Yorktown Heights, N. Y.

The third pitfall consists of the following fact: any problem chosen as an experimental vehicle is likely to produce interesting results if the program is successful. We consider these results to be byproducts of the general problem of studying the methodology of problem-solving, though they do serve as a test for success of the methods employed. As these byproducts accumulate, one is increasingly tempted to spend a disproportionate amount of time on obtaining useful and/or interesting byproducts. This diverts the researchers and they make little progress along the lines originally intended.

The class of problems we chose was the following: given a set of elements, and a criterion of compatibility between any two of its elements, find a subset of mutually compatible elements satisfying given constraints. This covers a large class of problems to which no satisfactory analytic solutions are known. An example would be: given the set of all airplanes near an airport, find the largest subset of those which are on compatible (*i.e.*, noncollision) courses.

The first experimental problem we chose consisted of this: a word list was given together with a table of synonyms for each word. The test of compatibility between any two words on the list was provided by a speech-recognition machine which was not as discriminating as the human ear. The problem was to find a word list the same length as the original list, but with synonyms substituted wherever necessary so that all the words on the resultant list could be unambiguously identified by the speech-recognition machine. The difficulty encountered in substituting the synonyms is that each substitution may cause added incompatibility relationships. The final list of mutually compatible words would be a practicable working vocabulary for voice control of, say, an air defense center.

We decided that the following three heuristic methods would probably be used by humans. These are:

Method 1—test the first word against all the following words. Wherever two words are incompatible, substitute synonyms for the second word until a compatible synonym is found, then proceed. Repeat with the second word of the list. After the last two words are tested, reiterate. Eventually a word list meeting the compatibility criteria will be found, or else the tables of synonyms will be exhausted.

Method 2—this method is basically the same as method 1, with this added sophistication: whenever a compatible synonym is substituted, it is immediately tested further for compatibility with all words in the list occurring before the particular two words concerned.

Method 3—determine which words are incompatible with the largest number of other words in the list and substitute synonyms for these highly incompatible words first. Then reiterate.

Two submethods also suggest themselves as processes to apply prior to trying the above methods. These are:

Submethod 1—sort the words on the first phoneme.

Submethod 2—try a batch procedure: divide the original list into two or three parts, apply one of the three main methods to each in turn, then put the batches back together before trying the final manipulation.

All of these methods were programmed and put together in a master program which determined the sequence of the application of each of the methods and submethods. Quite naturally, this sequence was: first, the "quick and dirty" method 1; then, the more sophisticated method 2; and last, the more complicated and most powerful method 3. At the start, each of the methods was allotted a certain amount of time, and if no compatible word list was produced within the allotted time, then the master program switched to the next method. This procedure was modified so that the time was extended if an analysis showed that the method had a good chance of successfully producing a compatible word list. To determine which submethod to employ first, a random choice was made, and lack of success automatically switched control to the other submethod.

To simulate these methods we employed a random number generator to generate 18-bit pseudo words, each consisting of three 6-bit pseudo phonemes. Inasmuch as the English language contains approximately 43 phonemes, 6-bit pseudo phonemes can reasonably be expected to represent adequately human phonemes. The human ear was considered capable of distinguishing between any two 6-bit pseudo phonemes that differed in one or more bit positions. The hypothetical speech-recognition machine, being not so discriminating, was considered capable of differentiating between any two 6-bit pseudo phonemes if and only if they differed in two or more bit positions. Mathematically stated, two pseudo phonemes P_i and P_j are considered compatible whenever

$$W(P_i +. P_j) > 1$$

where the weight function, W, merely counts the number of ones in the argument, and the operation $+.$ between the two pseudo phonemes is bit-by-bit addition modulo two, which is equivalent to exclusive OR.

Our experimental byproduct results are given in Table I, where the "word-list length" is defined as being the length of word lists that have a 50 per cent chance of being satisfactorily manipulated.

The second experimental problem consisted of finding the largest mutually compatible set of 12-bit numbers satisfying the following criterion of compatibility:

$$W(A_i +. A_j) \geq 5.$$

This set will have the characteristics that double error detection and correction is possible when employing it as an information transmission code.[1]

[1] R. W. Hamming, *Bell Sys. Tech. J.*, vol. 29, pp. 147–160; April, 1950.

TABLE I

EFFECTIVENESS OF SEVERAL METHODS OF FINDING COMPATIBLE WORD LISTS

Method	Word-List Length	Average Computing Time in Minutes
1 alone	325	0.65
1+sub 1	300	0.74
1+sub 2	325	0.78
2 alone	550	3.70
2+sub 1	525	2.55
2+sub 2	550	2.95
3 alone	740	3.82
3+sub 1	740	3.94
3+sub 2	740	4.50
Master Program	750±	Dependent on Word List Length

While in general the resulting code is nonconstructive, in this paper we confine ourselves to a subgoal, namely, the results obtained when constraining the set to be a constructive group code. With this constraint, the original compatibility criterion of $W(A_i+.A_j) \geq 2E+1$, where E is the number of errors to be detected and corrected, can be reduced to the following set of constraints on binary numbers whose lengths are those of the check bits only; the so-called parity check matrix:[2]

$$W(C_i) > 2E - 1$$
$$W(C_i +. C_j) > 2E - 2$$
$$W(C_i +. C_j +. C_k) > 2E - 3$$
$$\cdots\cdots\cdots\cdots$$
$$W(C_i +. C_j +. C_k +. \cdots C_n) > 0.$$

The size of the set of C's determines the maximum number of information bits that can be handled by check bits whose formulas for construction are given by the columns of the C's. These constraints were programmed and the size of the matrix determined by exhaustion.

The experimental byproducts are given in Table II.

Having satisfied your curiosity regarding the possibility of obtaining useful and/or interesting results from these methods, we now wish to discuss those aspects that are pertinent to the simulation on computing machines of human problem-solving strategies and techniques.

In the beginning we had what we thought were very good ideas of how to proceed. We had solved all of the problems on a medium level of abstraction. We soon learned, however, that an abstraction is nearly always a refuge from accurate knowledge. Explaining to each other how to reach a particular goal did not teach us how to program the simulation of the manipulations involved until we descended the abstraction ladder to the level of the symbolic coding process we were employing. For example in submethod 1, sorting the words according to the size of the leading 6-bit phoneme is useless. For detecting 1-bit differences it is a useful procedure, but for detecting 2-bit differences, one must order the words in an order that bears a useful relationship to the compatibility criterion. In our first problem we found this relationship in the weight of the phonemes, and accordingly arranged the words in order of increasing weight of the first phoneme. The result was a quicker finding and weeding out of incompatible words in isolated cases.

TABLE II

DOUBLE ERROR-CORRECTION CODE LENGTHS

Number of Check Bits	Number of Information Bits
6	2
7	4
8	≥ 9
9	≥13
10	≥16

A more subtle difficulty arose in the second of our problems. Consider the following five beginning members of a larger set satisfying the constraints previously described:

1) 11 11 00 00
2) 00 11 11 00
3) 00 00 11 11
4) 11 00 00 11
5) 10 10 10 10.

If element number 5 was tried and discarded because it was found not to yield a sufficiently large set of C's, then the number 01 01 01 01 need not be tried because it will yield similar unsatisfactory results. When doing this by hand on a piece of paper, one can immediately detect the equivalence between the two trial numbers for the fifth element. The criterion for this equivalence is that the two sets of five numbers containing different fifth elements are identical upon column permutation. It took us literally days to describe this simple relationship to the 704 computer whereas it takes only a minute to describe it to you. One of the reasons, of course, is that this concept involves two-dimensional visualization at which humans are particularly good and 704's are particularly inept. As a result, we wasted a lot of time trying to circumvent the transposition of the matrix elements in order to avoid using an excessive amount of machine time. A second-level difficulty arose when it was realized that all of the possible equivalences were not discovered by the original code, which did not contain column transposition. Consider, for example, the situation when the fifth member is 10 10 11 00. Then the number 10 10 00 11 is equivalent, but an involved

[2] D. Slepian, *Bell Sys. Tech. J.*, vol. 35, pp. 203–234; June, 1956.

column permutation had to be programmed in order to detect such an equivalence.

In conclusion, we would like to give our motivation in trying to simulate human behavior. At the present time, there are many problems whose nature is such that machines do not handle them successfully even though they are regularly solved by humans.[3] Therefore, it seemed desirable for us to study a "working model" in action, to determine its component functions and to analyze each function to find out what part it played in the solution of these difficult problems. Our main goal is to be able to solve these kinds of problems, partly by new and different programs, and partly by new and different machine-instruction sets. We do not insist that the machines do all the work, because we are convinced that a combination of a human and a machine working in tandem will always have superior problem-solving powers than either working alone. To complement each other, better communication is required between humans and machines, and this implies communication on higher abstract levels than the ones now employed. In order for this to be possible the machines must have either wired-in or programmed "subroutines" to interpret the directions and/or suggestions given them by their human colleagues. These "subroutines" should overlap those of humans in order to be useful. We feel that this goal will be furthered by continued research along the lines described in this paper.

The authors would like to express their sincere thanks to N. Rochester for suggesting the problems and outlining the heuristic programming methods utilized.

[3] A. Newell and H. A. Simon, "Current Developments in Complex Information Processing," RAND Rept.; May 1, 1956.
H. L. Gelernter and N. Rochester, "Intelligent behavior in problem-solving machines," *IBM J. Res. Dev.*, vol. 2, pp. 336–345; October, 1958.

The Role of the University in Computers, Data Processing, and Related Fields*

LOUIS FEIN†

Introduction

SINCE the Fall of 1956, the author has been studying the genesis and operation of university programs in the fields of computers, data processing, operations research, and other relatively new and apparently closely related fields. The specific purposes were:

1) To study and evaluate the organization, curriculum, research program, computing equipment, financing, and facilities of universities in the United States having computer and/or data processing and/or related programs.
2) To identify those fields of study (some already accepted and identified as disciplines as well as those not yet so designated) that are unambiguously part of the computer and data processing fields and those closely related fields that might legitimately be part of a university program.
3) To appraise the role of the universities in these fields and to determine what universities might do to build distinguished programs in these fields.

During the course of the study it became clear that a university program in any field is an important function of the role of the university in society itself. The identification of this role thus became a fourth separately identifiable purpose of the study.

Source information was obtained by formal interviews and informal sessions with university administrators, directors of computing centers, faculty members, students, industrial representatives, and other interested persons. Places of interview were universities, scientific meetings, social gatherings, industrial plants, and research institutes. Important information was obtained from a few publications.

A questionnaire intended for mailing was considered and dropped because it became clear very early that only personal interviews could bring out the important facts and opinions. Hence, the conclusions and recommendations are not always derived from a mass of accumulated data. They do reflect the author's experience in the computer and data processing field, information about what universities are doing and especially what they are not doing, and the influence of those individuals interviewed who have experienced and reflected seriously on the same problems as those considered in this study.

* Part of this work was undertaken when the author was a consultant to Stanford University.
† Consultant, Palo Alto, Calif.