Backtrack Programming

SOLOMON W. GOLOMB* AND LEONARD D. BAUMERT

Jet Propulsion Laboratory, Pasadena, Calif.

Abstract. A widely used method of efficient search is examined in detail. This examination provides the opportunity to formulate its scope and methods in their full generality. In addition to a general exposition of the basic process, some important refinements are indicated. Examples are given which illustrate the salient features of this searching process.

1. *Introduction*

Virtually all problems of search, optimization, simultaneous or sequential decision, etc. can be fitted into the following formulational framework.

From the direct product space $X_1 \times X_2 \times \cdots \times X_n$ of the n "selection spaces" $X_1, X_2, \cdots, X_n$, which may or may not be copies of one another, find the "sample vector" $(x_1, x_2, \cdots, x_n)$ with x_i an element of X_i, which is extremal with respect to (e.g., which maximizes) the *criterion function* $\phi(x_1, x_2, \cdots, x_n)$. The value of the criterion function for the extremal case is also to be determined.

If the extremal vector is not unique, it may suffice to find one such vector or it may be necessary to find them all, depending on the nature of the problem. Frequently the criterion function is two valued, corresponding to acceptable and unacceptable (1 and 0), and it may not be known in advance whether acceptable solutions exist. The formulation covers this case as well as the multivalued or continuous-valued criterion function, since one or all sample vectors which maximize the criterion function are still sought.

Techniques of differential calculus and of the calculus of variations are appropriate for dealing with certain kinds of differentiable criterion functions on certain kinds of continuous product spaces. The methods of linear programming apply to linear criterion functions on convex polyhedra. For higher degree polynomial criterion functions there are more elaborate techniques (e.g., quadratic programming). For multistage decision processes which satisfy Bellman's "principle of optimality," the viewpoint of dynamic programming (which is indeed more of a viewpoint than a precise method) is applicable. Undoubtedly there are other special techniques for other specialized situations. However the purpose of this article is to describe a completely general approach to all such problems, whether or not more specialized techniques are applicable. This rather universal method was named *backtrack* by Professor D. H. Lehmer of the University of California at Berkeley. Backtrack has been independently "discovered" and applied by many people. A fairly general exposition of backtrack was given by R. J. Walker [11]. Even so, it is believed that this is the first attempt to formulate the scope and methods of backtrack programming in its full generality. Naturally when more specialized techniques are applicable they are frequently more efficient than backtrack, although this is not always the case.

2. *Basic Formulation of Backtrack*

We wish to determine the vector $(x_1, x_2, \cdots, x_n)$ from the Cartesian product space $X_1 \times X_2 \times \cdots \times X_n$ which maximizes the criterion function

* Present address: University of Southern California.

$\phi(x_1, x_2, \cdots, x_n)$. It is assumed that each of the selection spaces X_i contains only a finite number of distinguishable values. The finite numbers may be quite large and may correspond to quantization of continuous intervals. (Since all numbers in a digital computer are quantized this assumption is not a severe restriction.) Let M_i be the number of distinguishable values in X_i, and let $M = \prod_{i=1}^{n} M_i$. The *brute force approach* is to form each of the M possible sample vectors, evaluate each one in the criterion function ϕ, and see which one produces the biggest value. The *backtrack algorithm* is designed to yield the same answer with far fewer than M trials. When M is large (as it usually is—otherwise there is no problem) this becomes an important saving.

One important digression is called for at this point. There are several techniques called *learning programs and hill-climbing programs* aimed at the same class of problems. The underlying idea in these programs is to progress from a vector $(x_1, x_2, \cdots, x_n)$ to another vector adjacent to it (in some metric defined on the Cartesian product space) which yields a larger value of the criterion function ϕ. The program stops (converges) when a relative maximum is found. While the backtrack algorithm lacks such glamorous qualities as learning and progress, it has the more prosaic virtue of being exhaustive. For the user who wants the computer to yield the best possible answer (i.e., the absolute maximum), and who is not particularly interested in whether the computer used humanoid psychology to arrive at a (possibly suboptimal) result, the backtrack algorithm is to be recommended.

So much for what backtrack will do. Let us now turn our attention to how backtrack does it. The basic idea of backtrack is to build up the sample vector one component at a time and to use modified criterion functions to test whether the vector being formed still has a chance of success. The power of the method is this: If the partial vector $(x_1, x_2, -, -, \cdots, -)$ is already seen to be inherently suboptimal, then $\prod_{i=3}^{n} M_i = M/M_1M_2$ possible test vectors may be ruled out in one fell swoop, without having to examine them individually. A well-constructed backtrack program is one which rules out large regions of the Cartesian product space on the basis of simple tests.

In detail, some estimation is made of the desired value for the criterion function. A threshold value T may be picked based on an evaluation of a randomly selected vector, or on the best advance guess, or, as the program progresses, on the best case thus far. (This is greatly simplified in the case of the two-valued criterion function "unsatisfactory" and "satisfactory" where "satisfactory" is picked as the threshold value!) Also required is a *modified criterion function* which gives some upper bound to the value which can be attained by the original criterion function, in the most favorable case, given the first k choices of the n choices to be made. The effectiveness of a backtrack program is determined largely by the skillfulness with which the modified criterion function is selected and programmed, since a very weak upper bound will lead to the consideration of too many cases, while a very elaborate modified criterion function may involve too much time being spent on the cases which must be considered.

Before starting (and even in the course of) the "backtracking" process proper, it is prudent to make use of known facts about the selection product space, and in particular its symmetries relative to the criterion function, so that as small a portion of the product space as possible need be subjected to actual "search." From this standpoint, it may also be advantageous to permute the sample spaces

$X_1, X_2, \cdots, X_n$ in some fashion before proceeding to extract samples from them. For a discussion of these aspects of the problem see Swift [10].

Now for the backtrack process itself. To begin, the first element, x_{11}, is selected from the first selection space, X_1, and evaluated in the modified criterion function $\phi_1(x)$ (modified for the case $k = 1$, i.e., that only the first of the n selections has been made). If $\phi_1(x_{11}) < T$, then it is known that any sample vector from the subspace $x_{11} \times X_2 \times \cdots \times X_n$ is below threshold, and the second element, x_{12}, from X_1 is considered. The examination of the elements of X_1 is continued until an element x_{1j} for which $\phi_1(x_{1i}) \geq T$ is found. (If *no* such element exists, then the threshold level T is unattainably high, and one must either lower his sights or abandon the quest entirely.) Having found x_{1j}, the elements of X_2 are tested in turn until an element x_{2k} of X_2 is found for which the second modified criterion function $\phi_2(x, y)$ satisfies $\phi_2(x_{1j}, x_{2k}) \geq T$. (The modified criterion functions must satisfy the condition:

$$\phi_1(x_1) \geq \phi_2(x_1, x_2) \geq \cdots \geq \phi_n(x_1, x_2, \cdots, x_n) = \phi(x_1, x_2, \cdots, x_n)$$

for all possible vectors $(x_1, x_2, \cdots, x_n)$, to ensure that the method considers all possibilities.) If the previous tests are successful the procedure is applied to the elements of X_3, finding the first x_{3l} in X_3 for which $\phi_3(x_{1j}, x_{2k}, x_{3l}) \geq T$, and thence to X_4, etc. If however a situation appears where the first k choices seem satisfactory by themselves, but no representative can be adjoined from X_{k+1} to keep ϕ_{k+1} at or over the threshold, then it becomes necessary to backtrack to X_k and to try a new choice for the kth component. It may indeed be necessary at times to backtrack several steps before a new path leading even further forward is encountered. Ultimately, there must occur one of two situations. Either no satisfactory solution is reached after having backtracked through all the elements of X_1, in which case no solution attaining the threshold level T exists; or else a satisfactory solution is obtained. In the latter case three types of situations may be distinguished. It may suffice to find one solution attaining the threshold T, in which case there is nothing further required; or it is possible that a more desirable solution attaining some threshold $T^* > T$ exists, in which case the program is continued with the increased threshold level T^*; or else all solutions attained (or exceeding) the threshold level T are sought, in which case the first solution is stored or printed out and the backtrack program proceeds on past it in its quest for the next solution.

3. *A Simple Example*

A classic problem of combinatorial analysis, discussed in Netto [6, p. 64], is to place eight "queens" on an 8×8 chessboard so that no two "attack," i.e., so that no two of them are on a common rank, file or diagonal. This problem is ideally suited to solution by backtrack. To simplify the exposition, the problem of placing *four* queens on a 4×4 chessboard so that no two attack will be considered. Rather than beginning by an examination of the $\binom{16}{4} = 1820$ ways of selecting the locations for the four queens among the 16 squares on the board, the first observation is that each rank (i.e., row) is to contain exactly one queen. Thus, a queen is placed on the first square of the first rank (Figure 1a), and then another queen on the

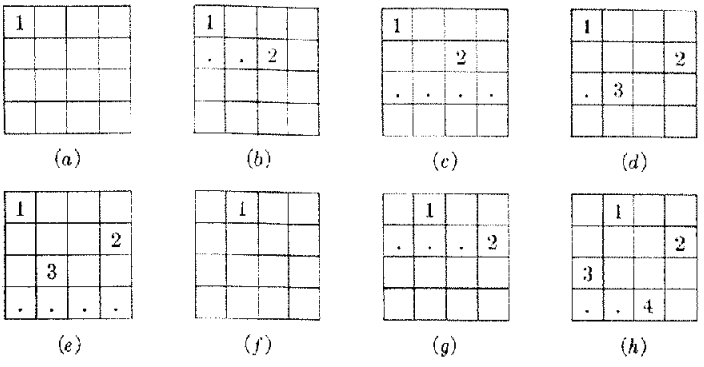

FIG. 1a–h. Example of a backtrack solution on a 4 × 4 chessboard with 4 queens

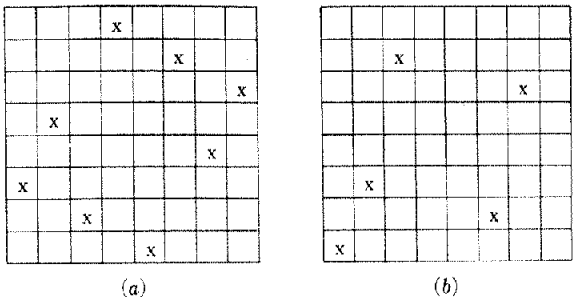

FIG. 2. Maximal and minimal solutions to nonattacking queens on the 8 × 8 chessboard

first available square of the second rank (Figure 1b). Observing that this prevents placing a queen anywhere in the third rank (Figure 1c), it is necessary to backtrack to the second rank and move the queen there one square to the right (Figure 1d); then there is room for another queen in the third rank. Now, however, there are no available squares (Figure 1e) in the fourth rank! Therefore it is necessary to backtrack all the way to the first rank (Figure 1f) after which there is one available location in the second rank (Figure 1g), followed by suitable locations in the third and fourth ranks (Fig. 1h), thus giving a solution. Further backtracking reveals that this solution is unique except for its mirror image, which is of course also a solution.

The same method also works efficiently for placing eight queens on the 8 × 8 chessboard, where instead of examining $\binom{64}{8}$ configurations (some 4.4 billion!) all 12 inequivalent solutions can be found in about two hours of hand computation.

To fit these examples into the formal description of backtrack programming, the n rows of the chessboard, R_1, R_2, $\cdots$, R_n, are considered to be the selection spaces, and from each row, R_i, a square, x_i, numbered between 1 and n, is to be selected. The criterion function ϕ is to have the value 0 if $x_i = x_j$ for any $i \neq j$, and also if $|x_i - x_j| = |i - j|$ for any $i \neq j$, otherwise, ϕ has the value 1. The modified criterion functions for the first k selections use the same rule but apply it only to $x_1, x_2 \cdots x_k$.

One of the 12 inequivalent solutions to placing eight mutually nonattacking queens on the 8 × 8 chessboard is shown in Figure 2a. The minimum number of

queens which can be placed on the 8×8 chessboard so that no two attack, but so that no more queens could be added without an attack occurring, is five. This type of problem is also readily treated by backtrack, and one of the minimal solutions appears in Figure 2b. (For an excellent history of the Eight Queens problem, see Ginsburg [2].)

4. *Another Example: Comma-Free Codes*

In the investigation of codes with favorable synchronization properties, one may consider the n^k "words" which consist of k symbols from an n-symbol alphabet, and look for the largest subset of these words which has the *comma-free property*. (Golomb, Gordon, and Welch, [3]). This property requires that if $a_1a_2 \cdots a_k$ and $b_1b_2 \cdots b_k$ are in the subset, then none of the "overlaps" $a_2a_3 \cdots a_kb_1$, $a_3a_4 \cdots b_1b_2$, $\cdots$, $a_kb_1 \cdots b_{k-2}b_{k-1}$ are in the subset. Such a subset is called a *comma-free dictionary* because in any "message" consisting of consecutive words from this collection, such as $a_1a_2 \cdots a_k$, $b_1b_2 \cdots b_k$, one may omit the commas or other separations between words, without the possibility of ambiguity as to word beginning and ending, even if one begins at a random starting point within the message. As an example with three-letter English words, if MAN and DOG were in the dictionary, then AND could not be in the dictionary because of the ambiguity in mANDog.

No *constant word*, such as AAA, can be in a comma-free dictionary because in the overlap of AAA,AAA the word AAA itself reappears. More generally, no word with a periodic structure, such as ABAB, can be in a comma-free dictionary because of abABABab. A *nonperiodic* word is one which is distinct from all its cyclic permutations. When such a word is included in a comma-free dictionary, all its cyclic permutations must be excluded since they appear in the overlaps of the word followed by itself. Thus if EAT is in the dictionary, we must exclude ATE and TEA because of eATEat.

This gives an upper bound to the size of a comma-free dictionary of k-symbol words over an n-symbol alphabet, namely, $[n^k - P]/k$ where P is the number of *periodic* words. A more explicit expression for P is given in [3], where it is also shown that this upper bound is attained for odd $k \leq 15$ and all n, but *not* attained for even k and large n.

The experimental tool for finding the largest comma-free dictionary for a given k and n is backtrack. Consider the case where $k = 4$, $n = 2$. There are $2^4 = 16$ binary words of length 4. Four of these (0000, 1111, 0101, 1010) are excluded as *periodic* and the remaining twelve are arranged into three classes of four each, where members of the same class differ only by cyclic permutation:

I	II	III
0010	1001	1011
0100	0110	0111
0001	1100	1110
1000	0011	1101

The upper bound, $[n^k - P]/k = [2^4 - 4]/4 = 3$, is attained if one representative from each of the three classes can be selected. As a starting point the first member,

0010, of I is chosen. Next, an attempt is made to adjoin the first member of II, 1001, but the overlap 10010010 is observed. Hence 1001 is discarded and 0110 is tried. The pair (0010, 0110) is compatible with the comma-free constraint, so an attempt is made to adjoin the first member of III, 1011. Here 10110010 is encountered, which leads to the rejection of 1011, and the next member, 0111, of III is tried. At this point it is found that the triple (0010, 0110, 0111) is comma-free, thus furnishing an example of a maximum-sized comma-free code for $n = 2$, $k = 4$.

A backtrack computer program [5] was used to find several codes for $k = 8$, $n = 2$ achieving the upper bound of 30 words; but for $k = 4$, $n = 4$ the upper bound of 60 is unattainable, and backtrack was used to determine that the largest such comma-free dictionary has only 57 words.

5. *Refinements of Backtrack*

Suppose one wishes to determine whether the sets $A = (2, 5)$, $B = (1, 3)$, $C = (1, 2, 3)$, $D = (1, 2, 3)$ possess a system of distinct representatives, and if so, to display such a system. Simple backtrack would solve the problem by examining the sequences *abcd* of elements a from A, b from B, etc. which follows: 2 1 1, 2 1 2, 2 1 3 1, 2 1 3 2, 2 1 3 3, 2 3 1 1, 2 3 1 2, 2 3 1 3, 2 3 2, 2 3 3, 5 1 1, 5 1 2 1, 5 1 2 2, 5 1 2 3. Clearly there is a great deal of wasted effort in this process, since for example the initial choice of 2 for A excludes 2 as a legitimate choice for C or D. Yet in the sequence of trials above this exclusion was tested 4 times. Let the search be performed again using such information. First starting with 2 from A and noting that this implies 2 cannot be picked from C or D, we next add 1 from B, noting that now 1 cannot be selected from C or D. Finally adding 3 from C it is seen that D has been exhausted so no system of distinct representatives can start 2 1 3. Carrying out this process from the beginning yields 2 1 3, 2 3 1, 5 1 2 3 as a successful sequence of trials terminating in 3 steps with a solution, whereas the previous process required 14 steps to deliver the same answer 5123. Clearly then the use of the additional information led to a more efficient search process.

Use of this type of information in the manner just illustrated is called *preclusion*. One would say for example that 2 from A *precludes* 2 from C or 2 from D. Depending on the nature of the problem, and also the nature of the problem solver (*i.e.*, human being, digital computer, etc.) preclusion may or may not be a more efficient process than straightforward backtrack. In a typical distinct representatives problem done by hand, the example above shows that in general preclusion would be desirable. On the other hand if the same problem were to be done by digital computer, preclusion would probably not be used. The reason for this is that there is usually no way outside of directly searching the sets to see whether any of them contains members already selected, and direct search is a relatively time-consuming thing for a digital computer to do.

Usually, however, whenever the problem can be reformulated or the solution process modified so as to make preclusion relatively easy to carry out, the resulting algorithm is more efficient than others which do not use preclusion. Preclusion was used effectively in the comma-free code search mentioned above, and another example of its use shall be seen in the next section.

The use of preclusion allows the employment of another principle of efficient

search. This principle says in effect that, all other things being equal, it is more efficient to make the next choice from the set with fewest elements. Of course examples can be constructed for which this principle is false, but from an information-theoretic point of view, it can be shown that on the average this choice from the smallest set is more efficient. As with preclusion, the question of relative cost arises and sometimes rules it out either because it overly complicates the search process, or simply because it takes too much time to search for the smallest set. Again it often happens that a reworking of the problem formulation or of the solution process yields a case where choice from the smallest set is not too expensive a procedure. When this is the case the resulting search is usually more efficient.

6. *An Example: Sum-Free Sets*

A problem of some number-theoretic interest is that of regularity of an equation or system of equations [8]. A specific example is the question of sum-free sets (here the equation is simply $x+y = z$) where a set S of positive integers is called *sum-free* if x, y in S imply that $x+y$ is not in S. If $F(n)$ is defined to be the largest integer q such that $1, 2, \cdots, q$ can be divided into sum-free sets $S_1, \cdots, S_n$, then it is known that $F(1) = 1$, $F(2) = 4$, $F(3) = 13$, $F(4) = 44$, $F(n+1) \geq 3F(n)+1$, and $F(n) \leq [n!e]$ where the brackets indicate the greatest integer $x \leq n!e$. A digital computer backtrack program was used to determine that $F(4) = 44$. The specific algorithm used preclusion and proceeded as follows.

Relabel S_1, S_2, S_3, S_4 as A, B, C, D respectively. Now it is assumed that $F(4) > 44$, and therefore the distribution $1, \cdots, 45$ is considered. Suppose $1, \cdots, j-1$ have already been distributed. Then the sets A, B, C, D are checked in order to find the first one that can accept j (the complete process shows that if we get to this stage, then at least one set will accept j). Assume set B is the first such and already contains $b_1, b_2, \cdots, b_s$. Then j in B precludes the admission of b_i+j into $B (1 \leq i \leq s)$. Those $b_i+j \leq 45$ not previously precluded are tagged with j. If no $b_i+j \leq 45$ is thus precluded from all 4 sets, an attempt is made to place $j+1$ as before. However if some $b_i+j \leq 45$ has been precluded from all sets, no solution for 45 with $1, \cdots, j$ as distributed can be obtained. Thus an attempt is made to put j into C or D in that order. Failing this, $j-1$ is shifted into a lexicographically higher set than it now occupies. If $j-1$ cannot fit, we try $j-2$, etc.

If one insists that 1 belong to A, 2 to B and that $c_1 < d_1$, a starting point for the algorithm including "isomorph rejection" (i.e., rejection of symmetries of the criterion function) is obtained.

In addition to ruling out $F(4) \geq 45$ this process was used to compute several solutions achieving 44. For example:

A: 1, 3, 5, 15, 17, 19, 26, 28, 40, 42, 44
B: 2, 7, 8, 18, 21, 24, 27, 37, 38, 43,
C: 4, 6, 13, 20, 22, 23, 25, 30, 32, 39, 41
D: 9, 10, 11, 12, 14, 16, 29, 31, 33, 34, 35, 36

7. *Statistical Backtrack*

In the literature on coding for the "discrete noisy channel" it is typical to consider using certain binary vectors of length n as code words, with only 2^m of the

possible 2^n such vectors ($m < n$) as the actual codewords. At the receiver a binary vector of length n is received, which may differ from the transmitted vector in one or more components because of the noise in the channel. The "maximum likelihood" decoding process requires that the received vector be compared with each of the 2^m possible transmitted vectors to find the best fit. Since it is not uncommon to talk of $m \geq 200$, it is quite impractical to perform all 2^m comparisons individually.

To handle this situation the method of *sequential decoding* [12] uses a form of backtrack in which the partial criterion functions are ad hoc probabilities of ultimate success. Specifically, the first received bit restricts the set of likely messages to those starting with this bit. The next received bit further restricts this set, etc. If all n bits "fit" one of the legitimate codewords, it is assumed that this codeword has been received with no errors occurring. More typically, however, after t bits have been received ($1 \leq t < n$) this partial codeword will agree with none of the legitimate transmitted codewords. In this case one assumes the legitimate codeword (or set of codewords) which fits best and then examines additional bits, noting each time the percentage agreement. If this percentage falls below a threshold of *reasonable likelihood* (the partial criterion function, which in general may depend on how many received bits are being considered), the program backtracks to the "second-best fit" at the previous plateau, and sees whether this can be extended more successfully. If continued backtrack fails to produce a better fit, the standard of reasonable likelihood is covered and the procedure is continued as before. The most that can be claimed for sequential decoding is that for sufficiently favorable signal-to-noise conditions the process will converge on the correct transmitted message with high probability.

In this case of backtrack the objective is to find that member of the set of prototype vectors having highest correlation with the empirical vector. Since the value of the highest correlation is not known a priori, the backtrack program must allow for an adjustable threshold level. An understanding of the statistical nature of the symbol errors which occur is of course quite valuable in writing an efficient program for sequential decoding. It has further been observed [13] that this approach can also be applied to the problem of the continuous noisy channel. For the "binary erasure channel" [7] the sequential decoding algorithm reduces to ordinary (deterministic) backtrack.

8. *Summary*

Backtrack programming has been successfully applied to the solution of multifarious combinatorial problems, of which the examples in the present article are merely representative. Other interesting combinatorial applications have included problems of optimum routing, problems of fitting geometric shapes into specified regions [9] and finding an Hadamard Matrix of order 92 [1]. Additional references may be found in the paper of Hall and Knuth [4]. Many people have independently "discovered" backtrack for their own applications, and a principal aim of the present paper has been to describe the method in its true generality.

As backtrack is no more than an "educated" exhaustive search procedure; it should be stressed that there exist numerous problems which even the most sophisticated application of backtrack will not solve in a reasonable length of time.

(For example the sum-free set problem for the case of 10 sets.) In fact "most" combinatorial problems grow to such an extent that there is at most one additional case beyond hand computation that can be handled by our present high speed digital computers. For many of these problems it takes an extremely sophisticated application of the principles of backtrack even to do this one additional case. Thus the success or failure of backtrack often depends on the skill and ingenuity of the programmer in his ability to adapt the basic methods to the problem at hand and in his ability to reformulate the problem so as to exploit the characteristics of his own computing device. That is, backtrack programming (as many other types of programming) is somewhat of an art.

The authors' own interests have been chiefly combinatorial. However, it is believed that significant applications of backtrack also exist in many areas which are not considered combinatorial, and it is hoped that this article may contribute to the discovery of such applications as well.

RECEIVED MAY, 1965

REFERENCES

1. BAUMERT, L. D., GOLOMB, S. W., AND HALL, M. Discovery of an Hadamard matrix of order 92. *Bull. Amer. Math. Soc. 68* (1962), 237–238.
2. GINSBURG, J. Gauss's arithmetization of the problem of 8 queens. *Scripta Math. 5* (1939) 63–66.
3. GOLOMB, S. W., GORDON, B., AND WELCH, L. R. Comma-free codes. *Can. J. Math. 10* (1958), 202–209.
4. HALL, M., AND KNUTH, D. E. Combinatorial analysis and computers. Herbert Ellsworth Slaught Memorial Papers, No. 10 (Suppl., *Amer. Math. Mon.*, 1965).
5. JIGGS, B. H. Recent results in comma-free codes. *Can. J. Math. 15* (1963), 178–187.
6. NETTO, E. *Lehrbuch der Combinatorik*. Chelsea Publishing Co., New York.
7. PETERSON, W. *Error Correcting Codes*. Technology Press, New York, 1960.
8. SALIE, H. Zur verteilung naturliche zahlen auf elementfremde klassen. Ber. Verh. Sachs. Akad. Wiss. Leipzig. *Math.-Natur Kl. 101*, 4 (1954).
9. SCOTT, J. Programming a combinatorial puzzle. Dept. of Elec. Eng., Princeton U., June 1958.
10. SWIFT, J. D. Isomorph rejection in exhaustive search techniques. *Amer. Math. Soc. Proc. Symp. Appl. Math. 10* (1960), 195–200.
11. WALKER, R. L. An enmerative technique for a class of combinatorial problems. *Amer. Math. Soc. Proc. Symp. Appl. Math. 10* (1960), 91–94.
12. WOZENCRAFT, J. M., AND REIFFEN, B. *Sequential Decoding*, Technology Press, New York, 1961.
13. ZIV, J. Coding and decoding for time-discrete amplitude-continuous memoryless channels. Sc.D. Thesis, Elec. Eng. Dept., MIT, 1962.