An interactive approach for the solution of a class of discrete optimisation problems

E. S. Page and L. Waller

Computing Laboratory, University of Newcastle-upon-Tyne, Claremont Tower, Claremont Road, Newcastle-upon-Tyne NE1 7RU

The paper describes the design and implementation of an interactive system for attacking a wide class of discrete optimisation problems. Features requiring consideration in such a system are the interface with the human, the requirements from the operating system, and the variety of heuristic techniques that must be provided. Some details are given of interactive facilities for limiting the amount of computation expended in a backtracking approach for a 'branch-and-bound' search.

(Received August 1972)

1. Introduction

Certain types of discrete optimisation problems have the characteristic that a solution may be represented by a vector v which minimises some cost function $C(v)$. In effect, the cost function, C, is a 'black box' which takes as input a vector v and outputs a cost associated with the vector. The difficulties presented by such problems in general arise from the great number of possible input vectors and make the determination of one which produces the minimum cost far from trivial. An orthodox approach is to utilise knowledge of the black box's behaviour to attempt to arrive at an analytical solution to the problem. There are, however, problems in which such knowledge provides little help in deriving an analytical solution or of devising a computationally practical deterministic algorithm. The usual approach is to adopt a heuristic method which provides a 'good' approximate solution and attempts to back it up with an assessment as to how good the solution is. To attack a wide class of discrete optimisation problems there is thus a need for a set of general purpose heuristics. The heuristic approach described here is an interactive one intended to utilise the powerful facilities offered by modern multi-programming computers. Man-machine interaction provides the ability to direct computational effort into areas which one's judgement and/or intuition suggest are worth exploring. The human can be ignorant of the mechanism of the black box function but may learn that 'similar' input vectors yield 'similar' costs. The class of combinatorial problems that may be attacked by the interactive system described here may be termed 'permutation problems'; i.e. the input vector to the cost function can be represented as a permutation of 1 to n, where n is a positive integer but other problems could be tackled in a similar fashion and similar techniques developed. Examples of permutation problems are the job-shop scheduling problem (Sisson, 1959), the travelling salesman problem (Little, Murty, Sweeney and Karel, 1963) and a problem of assigning facilities to locations (Gavett and Plyter, 1966).

The interactive system described in this paper has been named IMPACT ('an Interactive System for the Manipulation of Permutations for attacking Combinatorial Problems'); it has been implemented on the IBM 360 Model 67 computer running under the Michigan Terminal System (MTS) at the University of Newcastle-upon-Tyne but system dependence is limited to a small number of features like the handling of 'attention' interrupts.

2. Design objectives of IMPACT

Several objectives were formulated in the design of IMPACT. It clearly needed to be applicable to a wide range of discrete optimisation problems. A set of powerful heuristics should be made available with interactive access to permit decision taking and provide immediate feedback from the system to the decision taker. The system had to be easy to use so that the user could easily concentrate his intellectual effort into the problem-solving without being encumbered with troublesome system details. Furthermore, whilst the system was to be easy to use it should also be difficult to abuse in the sense that any typing or logical errors by the user should not result in a catastrophe but should be detected as minor errors and be catered for by IMPACT so that the user could be prompted to remedy his mistake.

Extendability was required so that new heuristics or commands could be introduced and to cater for different problems. IMPACT should also lend itself to easy transfer under different operating systems and to depend as little as possible on a particular machine; it was therefore decided that IMPACT ought to be written in a high level language and FORTRAN was chosen. Recourse to ASSEMBLER language has been permitted only where absolutely necessary, namely to access certain system functions (the current version of IMPACT uses only three such routines).

Extendability is obtained by modularity in the programming of IMPACT. Each facility provided occupies its own module and has access to a common database (areas of COMMON blocks). A skeleton of a main program provides a switching network between these modules and hence additional features may be easily incorporated by the writing of an appropriate module and a slight modification to the main program.

Modularity also made possible the objective of being able to attack a wide class of combinatorial problems. To use IMPACT upon a different problem of the class defined earlier, a user must write a suitable module to calculate the cost of a given input to his particular black box function. This and a suitable routine to place any data in the common database within IMPACT allows use of many of the powerful heuristics that exist in the system. Similar but more extensive changes and additions would permit different classes of problems to be tackled. Use of corrective programming allows the system to protect the user from himself. Checking is performed to ensure that common errors by the user are trapped. Parameters required by the commands are checked; for example, if the user enters what he considers to be a permutation of 1 to some positive integer n a suitable subroutine performs a test to see that this is so. Errors result in a meaningful diagnostic message being printed out and a chance to re-enter is provided for the user. Ease of use also stems from a ready feedback from IMPACT to the user and from the provision of facilities which permit the user to interrupt IMPACT at will. Interrogation by the user is then available as is modification of certain major variables. Finally the user may instruct IMPACT to continue the interrupted process or to abandon this process and initiate another (possibly different one).

IMPACT possesses a powerful set of heuristics, some of which are described below; certain of them were introduced as interactive problem solving with the system suggested them.

3. The operating system and IMPACT

IMPACT appears to the operating system as one large program which when running needs only a small portion of the compiled program to be kept in main storage since in essence IMPACT consists of a set of subroutines which share a common database and are linked by a main program which performs the function of allowing the user to switch from one subroutine to another. Under MTS any subroutine called which does not reside in main storage is 'paged' into main storage from secondary drum storage by the operating system. The virtual storage requirement for IMPACT is currently about 85 pages (1 page = 4096 bytes) and only a small proportion of this (10–20 pages) is required in core at any one time. In a particularly interactive session the large amounts of 'think' time would mean that almost all of IMPACT would reside on secondary storage should the main storage be required by other users' programs in MTS. The core storage requirements for IMPACT are thus not excessive.

Central processor usage time for IMPACT is dependent upon which facilities the user requires and how heavily he uses them. Most facilities provided require computing times of the order of a few milliseconds, whilst some involving powerful search techniques can demand computing times of a few minutes or more. Controls are however provided for interrupting or policing such time-consuming processes.

4. Usage of IMPACT

Once the user has initiated the execution of IMPACT and has provided basic data to specify the particular problem he is attacking he can invoke any of a set of heuristics by simply specifying a particular command word whenever IMPACT places him in *command mode*.

Commands consist of the name of a particular operation to be carried out and may require certain parameters.

In command mode the user may enter a command name and be prompted for parameters, or he may enter the name and the parameters together. An effort has been made to make command names meaningful, e.g. 'CMC' is used to invoke a Chain Monte Carlo technique. Any parameters entered are positional parameters and a single parameter may be an integer number or a permutation (part permutation) name.

The ability to interrupt execution of command provides a powerful facility in IMPACT.

Certain commands are specified as being interruptable and the depression of the attention button places the user in *interrupt mode*. In such a mode a meaningful message will be displayed and will be followed by a single line

?

and the unlocking of the keyboard. The user may then enter local (or sub-commands) which allow the display and modification of major variables. As an example, the state of search may be ascertained, modified and resumed. Examples of such techniques are given.

Storage display commands allow IMPACT to be interrogated by the user to ascertain various features of the interactive session. Typically, the best solution found so far may be viewed (RECAP); the permutation which the interaction is currently considering may be viewed and manipulated (BASE, POPCAP); permutations may be defined and displayed (GIAN, CATALOG); the CPU time used over an interval can also be determined (TIME).

5. Heuristic techniques

Several heuristics are provided by permutation manipulation commands. A single permutation specified by the user may be submitted to the cost function and the resulting cost displayed (HUNCH). Elements in a particular permutation may be interchanged (INTCH) as may different sized blocks of elements (MOVE). Parts of permutations may be reversed (REV) or cycled (CYCLE) and specific elements in the permutations may be tried in other positions (WEAVE, TONFRO). After each of the above commands the resulting permutation is displayed. At any stage in an interactive session the user will be informed if there has been any improvement in the best value found so far. Random permutations may be generated (SHUFFLE), as may permutations that are part random in the sense that certain elements of a base permutation are not to be disturbed from their original position (RANFIX). Various more complex heuristic techniques are also provided. Some of these are based upon sorting techniques (SELECT, EXCHANGE, MERGE) (Page, 1961), whilst others adopt a Monte Carlo approach (CMC) (Page, 1965). Methods which permit complete enumeration are also included. One (PERLEX) allows the generation in a lexicographic manner of all permutations of 1 to n, or restricts this generation to part of an original permutation. Others allow the use of branch-and-bound approaches with different methods of performing the branching. One method of branching (TRAKBAK) utilises a straightforward backtracking method (Lomnicki, 1965), while another permits an approach developed interactively (see Waller, 1971). These branch-and-bound techniques require, of course, that the user provide suitable routines for calculating lower bounds.

Commands which are peculiar to the particular cost function being investigated may be inserted in IMPACT. Typically, such facilities have been so far used for measuring the value of a solution found by a heuristic technique, for looking inside a cost function, and for testing out ideas. Although IMPACT is aimed at providing a set of heuristics into which a user could simply insert his own cost function and thus arrive at some 'solution' without any exploitation of special features which the cost function might have, the extendability of IMPACT permitted the introduction of special commands which can be made interruptable without undue difficulty.

6. Aids to ease of use of IMPACT

Human factors enter prominently into the design of an interactive system; one of these concerns the size, number and form of keyboard entries that are required. In a system manipulating permutations it is slow (and productive of error) for a user to enter complete permutations of 1 to n unless n is very small; accordingly, permutations or part permutations may be named either explicitly or implicitly. Explicit definition is achieved by the use of the command GIAN ('Give It A Name') and at any later stage named permutations may be retrieved for use as input parameters to commands. Certain names are reserved and have predefined meanings.

Implicit naming uses the concept of a current active permutation which refers to a temporary storage mechanism for permutations which the user may use and retrieve without the necessity of having to define them explicitly. A certain number (an IMPACT parameter at generation time, and currently three) of permutations are stored in a push-down stack manner and may be referred to by CAP 1, CAP 2, . . . etc. (CAP 1 being the most recently implicitly defined permutation, CAP 2 the one before that, and so on). Whenever a new implicitly defined permutation enters the stack the permutation which was CAP 1 becomes CAP 2, the old CAP 2 becoming CAP 3 and so on. (The permutation at the lowest level is discarded if necessary). Also manipulated with this stack of permutations is a stack with their associated cost. These costs may be referred to as CC1, CC2 . . . etc., in the same manner as above.

Most of the commands available have as result a permutation and cost which become CAP 1 and CC1 respectively, thus causing stack manipulation.

If after the use of a command which created a new member of the CAP family, the user wishes to base his next command on the previous permutation he may do so by use of the command

POPCAP. There is, of course, no means of recovering permutations which had to be discarded from the bottom of the stack so the situation retrieved may be different in this respect.

The reserved permutation names are BEST, NATU, NTO1, PADD and the generic name CAP k mentioned above (where k represents a digit in the range 1 to 9).

By referring to BEST in a string of parameters to a command the user will obtain the permutation corresponding to the lowest cost (stored at BC) found so far. NATU will result in the natural permutation $(1, 2, \ldots, n)$ being supplied and NTO1 gives its reverse $(n, n-1, \ldots, 2, 1)$. PADD is used to indicate that the parameters supplied to date are to be padded out with those elements of 1 to n in the natural order which have not yet been entered. For example, if $n = 8$ the parameters 6, 4, 8, PADD; are equivalent to 6, 4, 8, 1, 2, 3, 5, 7;. Thus PADD; is equivalent to NATU;.

At the end of a problem-solving session a user may obtain access to a permanent machine-readable record of the session stored upon a disc file. This record may be used for a variety of purposes. One possibility is that scrutiny of it together with explanations inserted by a COMMENT command might provide insight into the problem-solving approaches used by humans and thus suggest more powerful heuristic techniques. Judicious use of the attention button also allows one to ask for information which might be too voluminous for terminal output but which will be recorded onto the disc file and hence be available for a more suitable output device, e.g. a line printer, character display, or graph plotter.

7. Bounds and backtracking

Initial attempts upon problems using heuristic approaches gave some apparently 'good' results but it was not possible to assess just how good they were or whether they were obtained by good fortune or by some particular decision mechanism. There was no indication as to whether a search ought to be made in the neighbourhood of a permutation obtained or whether effort ought to be directed elsewhere. If the problem needed a minimum cost permutation access to the lower bounds on the costs associated with part permutations was desirable; accordingly a command 'BOUNDS' was implemented. Upon entering 'BOUNDS', which of course needs a user-provided routine appropriate to the particular cost function, and providing a part permutation $P_j = (i_1, i_2, \ldots, i_j)$ to IMPACT the lower bounds associated with the part permutations $(i_1, i_2, \ldots, i_j, k)$ are displayed in ascending order. If the user decides that he has seen enough of such information during the printing the remainder may be suppressed by the depression of the ATTENTION button. For example, the command, bounds, 7, 3; could produce the lower bounds for part permutations beginning with (7, 3)

(2 — 142)
(1 — 142)
(5 — 145)
(6 — 147)
(4 — 149)
(8 — 151)

A backtracking 'branch-and-bound' approach to permutation problems lends itself to an easy implementation by the use of a 'push-down' stack. At the first level of the tree n nodes are generated and ordered according to their lower bounds. The one with the lowest lower bound is retained and the remainder placed upon the stack so that in the event of any of them being required later the smallest will be retrieved first. Similarly at the next level of the tree $n-2$ lower bounds will be placed upon the stack, again ordered amongst themselves. **Fig. 1** depicts the arrangement of the lower bounds on the stack. When a complete permutation is reached bounds are removed from the top of the stack and either rejected (the lower bound is too high) or explored further. When a node is explored further any new nodes generated can be added to the stack as before. It can thus be seen that only as many nodes will be added as have been removed and hence no more storage is required. The maximum number of storage locations required for the stack is thus

$$(n-1) + (n-2) + \ldots + 2 = \frac{n(n-1)}{2} - 1 .$$

An approximate solution can be measured against the bounds shown in the backtracking approach and potentially fruitful regions for the search identified. Backtracking can however be very time-consuming and limits need to be set. The command 'TRAKBAK' takes a part permutation upon which backtracking is to be performed, which restricts the amount of computation to be performed as well as providing a facility for following up 'hunches'. The maximum number of vertices to be examined in such a search must also be specified.

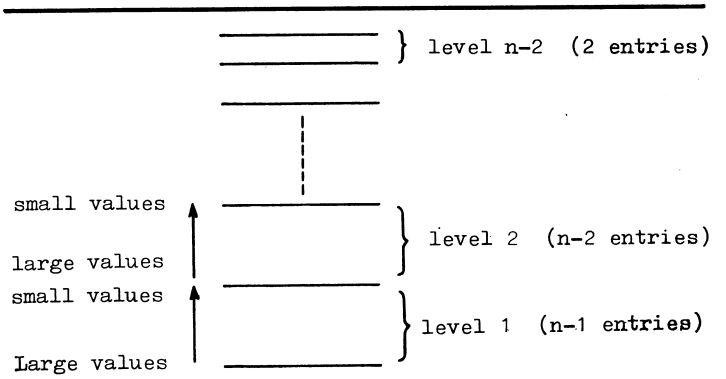

Fig. 1 Arrangement of a 'push-down' stack for backtracking

8. Interaction and backtracking

During the course of backtracking through the tree of permutations, situations may arise where the bounds show that the computational effort of the search is directed in a part of the tree which could give at most a small decrease in the lowest cost found so far. Further computational effort might be better employed in a different part of the tree. Facilities have therefore been provided for allowing one to interrupt the backtracking procedure.

Whenever the backtracking is interrupted by a depression of the attention button the number of vertices of the permutation tree (i.e. part-permutations) that have been examined is displayed together with the number of complete permutations that have been examined. This gives an indication of whether the computational effort has been expended in nearly complete permutations. Local commands provide the ability to display the tree remaining for investigation. For example, IMPACT might respond to an attention with

YOU'VE INTERRUPTED THE BACK TRACKING
AFTER 39 VERTICES AND 4 COMPLETE
PERMUTATIONS
?

entering 'HOWF' might then result in

howf
THE DEPTH OF THE SEARCH IS 5
AND THE PERMUTATION BEING EXPLORED IS 5 2 8 6 7
WITH LOWER BOUND = 117
?

'STAC'; shows the stack representation of those vertices of the tree that have lower bounds less than the current target value **(Fig. 2).**

?
stac

ID	BACK	BOUND	LEVEL
24	3	119	5
22	1	116	4
21	7	117	4
20	3	118	4
18	6	119	3
13	1	114	2
12	4	114	2
11	6	114	2
10	8	115	2
9	7	117	2
7	7	117	1

?

Fig. 2 The stack representation of a tree

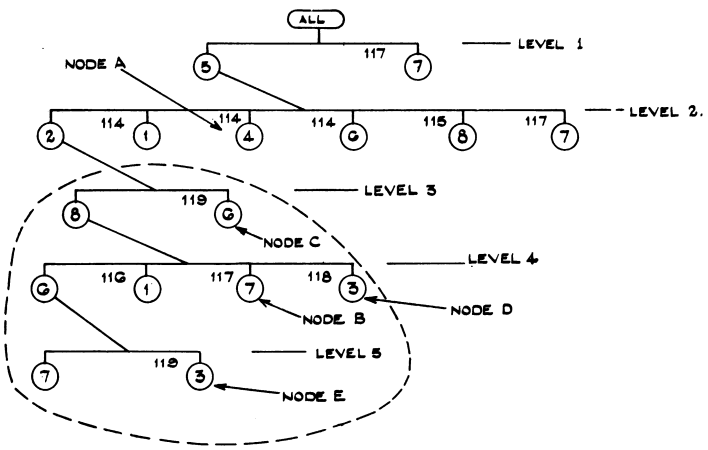

Fig. 3 The tree corresponding to Fig. 2

?
draw
117 ← Lower bound of part permutation being investigated

```
.
.
.          119
5(7)———(3)
.
.
.          116      117      118
4(6)———(1)———(7)———(3)
.
.
.          119
3(8)———(6)
.
.
.          114      114      114      115      117
2(2)———(1)———(4)———(6)———(8)———(7)
.
.
.          117
1(5)———(7)
↑
level
```

Fig. 4 The DRAW command

The above information is interpreted in the following manner: the permutation beginning with (5, 2, 8, 6, 3) is in position number 24 in the stack and has a lower bound of 119, whilst that beginning with (5, 4) is in position number 12 and has a lower bound of 114. The state of the search is thus depicted by **Fig. 3.** Parts of the tree that are still active are represented by nodes and their lower bounds are attached to them.

Tree drawing

The same information that is printed by the commands HOWF and STAC may be displayed graphically by use of the local command DRAW (**Fig. 4**).

Nodes lower down the tree are displayed first, and thus the part of the tree that will be examined soonest is seen first by the user. A parameter to the DRAW command gives the user the choice of drawing the root first or last. Use of the ATTENTION button during printing of the tree will terminate the printing although a drawing of the tree will be stored away automatically onto the recording file and will not be suppressed. Thus if one wishes only to store a drawing for later viewing the DRAW command can be issued and followed immediately by a press of the ATTENTION button.

Parts of the tree may be discarded or pruned in various fashions. A simple method is to MASK and specify which nodes are to be rejected. Implementation is performed by altering the lower bound associated with the node to be masked to a value higher than the best cost found so far. Thus to discard the nodes labelled A and B in Fig. 3 one should specify that positions 12 and 21 in the stack are to be masked.

A more powerful method of reducing the amount of computation in a search is to alter the target-value (the value with which the lower bounds are compared) by the command TARG. If at some stage the cost of the best solution found so far is C_b then altering the target-value to $C_t = C_b - \delta$ will cause only those nodes with a lower bound less than C_t to be examined. δ would normally be positive (but need not be so) and thus fewer nodes of the tree would be examined. It does not follow that no permutations giving a cost between C_t and C_b will be found since a part permutation of $n - 2$ elements may have a lower bound less than C_t and thus warrant investigation.

In the example if the best cost discovered so far is 120 then using TARG to search for values better than 118 would result in nodes C, D and E in Fig. 3 being thrown away, leaving **Fig. 5.** Implementation is performed by adjusting the value with which the lower bounds are compared rather than by altering values in the stack. Thus after a time if the manoeuvre appears to be fruitless in the sense that no improvement in C_b is made the attention button may be used and TARG reset. A certain amount of the information can thus be retrieved.

In practice the TARG facility has been found useful for making large jumps up the tree to levels where the potential payoff (with reference to the lower bounds) could be greater and for avoiding other parts of the tree where the payoff could at best be small.

The TARG command is also useful for limiting in a selective manner the amount of information displayed by the STAC or DRAW commands. The stack of lower bounds is of the order of n^2 elements but TARG permits the display of only those nodes which have lower bounds less than the value specified. The targetvalue may of course later be reset to its previous value should it be desired. Thus the user may see which parts of the tree he will be pruning before committing himself to such a course, and hence the possibility of being too severe may be avoided.

Jumping up the tree

The two previous methods of tree pruning have the disadvantage that any nodes which are rejected because they are unlikely to yield a sufficiently high payoff cannot be retrieved easily within IMPACT. An alternative means of curtailing a search is available in the 'JUMP' command and does not have the above drawback. Use of the command allows the search to jump to a higher level of the tree, disregarding (either tempor-

```
?
targ
 ENTER NEW TARGET VALUE⎫ User Command and
118;                    ⎭IMPACT's confirmation
 TARGETVALUE ALTERED FROM  120 to 118
?
draw
 117
  .
  .
  .          Node E has disappeared
5(7)———(↙
  .
  .            Node D has disappeared
  .       116      117
4(6)———(1)———(7)———(↙
  .
  .
  .
3(8)———(←———Node C has disappeared
  .
  .
  .       114      114      114      115      117
2(2)———(1)———(4)———(6)———(8)———(7)———(
  .
  .
  .       117
1(5)———(7)———(
```

Fig. 5 The effect of altering the targetvalue

```
?
jump
 ENTER LEVEL TO JUMP TO ⎫User—IMPACT dialogue
3;                      ⎭
 SHALLWESAVETHEINFORMATIONJUMPEDOVER?
yes
 ENTER A NAME TO DEFINE IT
fred
 INFORMATION STORED AND NAMED FRED
?

draw

      Lower bound of part permutation now being investigated
 119↙                    i.e. (5, 2, 6)
  .
  .
  .
3(6)———(
  .
  .
  .       114      114      114      115      117
2(2)———(1)———(4)———(6)———(8)———(7)———(
  .
  .
  .       117
1(5)———(7)———(
```

Fig. 6 The effect of 'JUMP'ing up the tree

arily or permanently) the subtree jumped over. On invoking the command one is prompted whether information jumped over is to be saved or not. If it is to be saved than it must be named by the user and stored on a disc file. It then may be later retrieved by its name. In the example, if the search is to jump to level 3 then the part of the tree encircled in Fig. 3 would be jumped over and the next subtree examined would be

```
  #  #  #
resume
 ENTER NAME OF THE TREE TO RETRIEVE
fred
 RETRIEVED TREE NAMED 'FRED'
   SEARCH RESTARTS WITH 1000 VERTICES AS
     INITIAL MAXIMUM
 YOU'VE INTERRUPTED THE BACKTRACKING
   AFTER
     0 VERTICES AND
     0 COMPLETE PERMUTATIONS
?
howf
 THE DEPTH OF THE SEARCH IS  5
 AND THE PERMUTATION BEING EXPLORED IS
   5  2  8  6  7
   WITH LOWER BOUND = 119
?
draw
 119
  .
  .
  .       119
5(7)———(3)———(
  .
  .
  .
          116      117      118
4(6)———(1)———(7)———(3)———(
  .
  .
  .
3(8)———(
  .
  .
  .
2(2)———(
  .
  .
  .
1(5)———(
```

Fig. 7 Retrieval of the tree previously 'JUMP'ed over

based upon the part permutation (5, 2, *i*) where *i* is the next element in the filial set of (5, 2), i.e. 6.

Fig. 6 illustrates such an effect. The subtree jumped over has been saved and named 'FRED'; it can be retrieved later by a 'RESUME' command and when retrieved will be as in **Fig. 7.**

The JUMP command is implemented by adjusting the pointers for the level of the search and for the information stack. Information temporarily discarded can be saved in a compact form. The jump from level 5 to level 3 means that the part of the tree discarded can be specified by (5, 2, 8, 6, 7), the part permutation that was being examined, and the branches of the original tree that lay between the two levels concerned, i.e. (level 4, node 1 with bound 116, node 7 with bound 117, node 3 with bound 118) and (level 5, node 3 with bound 119).

The ability to jump and save the part of the tree jumped over allows the user to make the search more like branching from the lowest bound without losing all of the attractions of backtracking.

9. Conclusion

Use of an interactive system like IMPACT exposes the desirability of new facilities and heuristics. For example, in some problems it was helpful to keep certain elements of a permu-

tation fixed in an exchanging sequence and the EXCHANGE command was modified to allow this. As a result of the use of the system a significantly improved deterministic algorithm has been developed for the job-shop scheduling problem where the cost function is that of minimising the makespan of jobs upon machines. Interaction provided insight into the cost function and resulted in the discovery of improvements in lower bounds proposed by Lomnicki (1965) for a branch-and-bound approach. The lower bounds were strengthened and a slightly different branching mechanism, which exploited the structure of the problem, was adopted leading to a computationally feasible approach for moderately sized problems.

References

GAVETT, J. W., and PLYTER, N. V. (1966). 'The Optimal Assignment of Facilities to Locations by Branch and Bound', *Opns Res.*, Vol. 14. pp. 210-232.

LITTLE, J. D. C., MURTY, K. G., SWEENEY, D. W., and KAREL, C. (1963). 'An Algorithm for the Travelling-Salesman Problem', *Opns Res.*, Vol. 11, pp. 972-989.

LOMNICKI, Z. A. (1965). 'A "Branch-and-Bound" Algorithm for the exact solution of the three-machine scheduling problem', *Opns Res., Quart*, Vol. 16, pp. 89-100.

PAGE E. S. (1961). 'An Approach to the Scheduling of Jobs upon Machines', *J Roy Stat Soc. B*, Vol. 23, pp. 484-492.

PAGE, E. S. (1965). 'On Monte Carlo Methods in Congestion Problems; 1. Searching for an Optimum in Discrete Situations', *Opns Res.*, Vol. 13, pp. 291-299.

SISSON, R. L. (1959). 'Sequencing in job shops—A review', *Opns. Res.*, Vol. 7, pp. 10-29.

WALLER, L. (1971). *Interactive Approaches to the Solution of a Class of Combinatorial Problems*, Ph.D. Thesis, University of Newcastle-upon-Tyne.

Book reviews

Computer Programming in Algol, by J. D. Earnshaw and W. A. R. Blackford, 1970; 170 pages. (*Sir Isaac Pitman and Sons Ltd*, £1·40)

This is a Pitman Programmed Text, suitable for self instruction or for use as part of an elementary programming course for scientists at a college or technical school. The format of the book is the familiar format of a programmed text: some material is introduced; a question is asked on the material just introduced, or on earlier material; an answer is given. The validation report quoted in the introduction indicates that this text successfully fulfils its objective of teaching Elliott 903B ALGOL.

And this is my major criticism of the book. The authors admit that there are different implementations of ALGOL 60, and that their book attempts to teach the Elliott 903B implementation; but when the chosen implementation differs from the ALGOL of the Revised Report as widely as Elliott ALGOL does, there seems to be a good case for either teaching the language of the Report (or of one of the subsets defined in the document ISO 1538), or at least making very clear the points at which the language being taught departs from the standard.

To those people wishing to learn ALGOL for the 903B, this book can be recommended.

M. C. THOMAS (London)

The Settlement of Polynesia: A computer simulation, by M. Levison, R. Gerard Ward and J. W. Webb, 1973; 137 pages. (*Minnesota University Press; Oxford University Press*, London, £5·50)

The 'Polynesia problem', concerning the origins of the people of the Pacific islands and how they came to inhabit such a vast ocean area, has been a fruitful subject for academic speculation during the past 25 years. Information and ideas have been contributed by several disciplines, including archaeology, cultural and physical anthropology, botany and linguistics, but no general theory seems to offer the prospect of being adequately tested and so providing some resolution of the conflicting interpretations. In fact, one suspects that the fascination of these problems of early human history lies in the remoteness of any final answer.

In this book, two geographers and a computer scientist describe some work which, rather insensitively one suspects, attempts to answer one major question once and for all. It is unlikely that they will convince everyone, but they have obviously made a highly significant contribution to the general debate. Their question is, 'Could the Pacific Basin have been discovered and settled by drift voyaging, as some interpretations have suggested?' To supplement this, they also consider the likelihood of navigated voyages being responsible for the colonisation of some areas.

Their method is an ingeniously devised spatial simulation model of the geography and natural conditions of the Pacific. This takes into account the probability of drift voyagers experiencing different wind and current conditions at various times of the year, based on data for the last century; the likely speed capabilities of canoes and rafts under different conditions; the probability of survival over different lengths of time; and the conditions under which a destination island might be sighted and a successful landfall achieved. The computer simulation was used to reproduce more than 100,000 drifts from 62 starting locations, including some which were run in reverse from speculated landing points. A further 8,000 voyages were simulated which were assumed to have been intentionally guided according to fairly simple navigational rules. Thus the experiment had the advantage both of taking a synoptic view of an enormous area, and of reproducing a very large number of possible drift and navigated voyages. The results, of course, simply provide probabilities of contact between the island groups. In detail these are fascinating, and the general pattern which emerges is difficult to fault. Three regions can be identified; western Polynesia could have been occupied by island-hopping and drift voyages from the west, but much of the East Polynesia area of the central Pacific could not have been colonised in this way unless the eastern-most group, the Marquesas, was settled first. The outer arc of islands, including Hawaii, Easter Island and New Zealand, are most unlikely to have been settled by drift voyages. On the other hand, the simulations of navigated voyages confirm that purposeful crossings of all the major ocean stretches would have been possible with only limited degrees of navigational skill.

The construction of the model and its sources of data are lucidly explained in the text of the book, while useful appendices include computer-drawn microfilm maps of the 'drift fields' from different starting points and a a detailed listing of and commentary on the basic simulation program, which is in ICT Atlas ALGOL. The 'Polynesia problem', as the authors acknowledge, still includes many intractable questions, especially relating to the motivations of the voyagers in setting out. Nevertheless, the computer simulation has given us the first comprehensive view of what they were likely to encounter once the journey had got under way.

P. A. WOOD (London)