Correspondence

To the Editor
The Computer Journal

Sir
I have been in trouble with our typing service who complain that my spelling of ALGORITHM is incorrect. It appears that the Oxford Dictionary quotes ALGORISM, derived from the name of an Arab mathematician, and refers to the —ITHM ending as a modern mis-spelling.

Are we all out of step?

Yours faithfully,
H. A. Marriott

Management Services Manager
Private Communications Division
Standard Telephones and Cables Limited
Maidstone Road
Sidcup
Kent DA14 5HT
13 July 1973

To the Editor
The Computer Journal

Sir

Observations on a decision rule for binary programming

Wyman (1973) has recently formulated a quantitative rule for deciding between the use of a heuristic and a Balasian-type optimising algorithm for the solution of a particular class of 0-1 linear programs. While the decision rule may have some merit if one is faced with choosing between existing implementations of the particular heuristic and the particular optimising algorithm mentioned in the paper for the IBM 360/67 used in the study, it seems to me to be of little general applicability because:

1. In many cases, the algorithms will not be in the same state of availability in which case considerations other than the cost of computer time become important, e.g. the choice may be between in-house development of a code for the heuristic and hiring an existing code for the optimising algorithm. In regard to this point, my own Algol code for Senju and Toyoda's heuristic occupies some 150 statements including input-output whereas my Algol code (Proll, 1971) for a Balasian-type algorithm with no surrogate constraint feature but with a heuristic start procedure occupies approximately 370 statements and has a considerably more complicated structure. Development costs for the two codes could thus be expected to be very different. The significance of this could depend on whether the problem to be solved is of a one-off or recurrent nature and whether the chosen technique has other potential uses.

2. The timing predictions which form one component of the decision rule are both machine and implementation dependent. Assuming both algorithms to be implemented in a high level language, the timing predictions for the heuristic and optimising algorithm may be affected to a different degree by the efficiency of the object code generated by the compiler. Consequently the cost of and time involved in establishing the prediction equations may need to be considered if one is contemplating using this rule.

3. Wyman's predictors are based on a series of experiments in which each constraint in a particular problem has the same degree of slackness. It is hardly to be expected that this will be the case in practice but no indication of the effect of variations of slackness in the constraints has been given. For Petersen's problems 3, 4 and 5 (Petersen, 1967) which have factor values within the ranges used in the study, the following results were obtained:

Problem	*B*	*C*	*Range of Slackness*	*Z_0/Z_u Predicted*	*Z_0/Z_u Actual*
P3	10	70·8	67·0-76·9	0·793	0·777
P4	10	57·1	50·9-72·4	0·681	0·707
P5	10	70·7	60·9-85·8	0·781	0·800

Whilst the differences between actual and predicted values in these cases are not so great as to make one discard the given predictor, further testing of the regression model is surely necessary.

4. The decision rule is critically dependent upon the predicted ratio Z_0/Z_u and, as Wyman remarks, the regression model 'has limited validity for levels of factors *B* and *C* beyond the scope of this study.' To reinforce this remark, I quote the following results for Petersen's problems 6 and 7 and the two problems given in Senju and Toyoda's paper (Senju and Toyoda, 1968).

Problem	*A*	*B*	*C*	*Z_0/Z_u Predicted*	*Z_0/Z_u Actual*
P6	5	39	67·1	0·805	0·721
P7	5	50	62·8	0·787	0·735
ST1	30	60	36·7	0·574	0·820-0·888*
ST2	30	60	61·5	0·794	0·921-0·955*

*Z_0 unknown, range calculated from lower and upper bounds on Z_0.

A rule of the type proposed by Wyman seems also to undervalue Balasian-type optimisers since it assumes that the only information gained from such an algorithm is a verified optimal solution. In general Balasian algorithms yield a sequence of feasible solutions of increasing value and can easily be adapted to find the *k* best solutions of the 0-1 linear program. It is also well known that optimal or near optimal solutions are obtained relatively quickly and that much of the time taken by such algorithms is in trying to verify the optimality of a previously obtained solution (Proll, 1971). Thus it seems to me that a more relevant comparison between the two techniques might involve the time taken by the Balasian algorithm to obtain a value at least as great as that obtained by the heuristic.

Finally, in the last section of his paper, Wyman remarks that it would be useful to have a heuristic for the general 0-1 linear program which could be imbedded in the optimising algorithm. Such a device has been constructed (Byrne & Proll, 1969) and has indeed been found to accelerate the progress of the algorithm.

Yours faithfully,
L. G. Proll

Operational Research Unit
Centre for Computer Studies
University of Leeds
Leeds LS2 9JT
14 August 1973

References

Byrne, J. L., and Proll, L. G. (1969). Initialising Geoffrion's implicit enumeration algorithm for the zero-one linear programming problem, *The Computer Journal*, Vol. 12, pp. 381-384.

Petersen, C. C. (1967). Computational experience with variants of the Balas algorithm applied to the selection of R and D projects, *Man. Sci. A13*, pp. 736-750.

Proll, L. G. (1971). Further evidence for the analysis of algorithms for the zero-one programming problem, *CACM*, Vol. 14, pp. 46-47.

Senju, S., and Toyoda, Y. (1968). An approach to linear programming with 0-1 variables, *Man. Sci. B15*, pp. 196-207.

Wyman, F. P. (1973). Binary programming: A decision rule for selecting optimal vs. heuristic techniques, *The Computer Journal*, Vol. 16, pp. 135-140.

Mr. Wyman replies:

In general, the observations made by L. R. Proll of my recent article are quite interesting but I feel that his remarks do not merit characterisation of the decision rule as being 'of little general applicability'. The objective of my paper was to suggest a method of systematically comparing heuristic to optimising techniques, and not really to uphold either technique used for illustration as being ultimate in any sense. The fact that variations in results can occur due to inefficiencies of programming, compiling, and accuracy of timing measurement would seem to be rather self-evident.

The extensive evaluation implied by my paper is obviously not to be recommended for a single application of 0-1 programming, but rather for a situation of extensive repetitive application as in weekly truck-routing, vessel-scheduling, airline crew assignment, project selection, etc. Computer timing is indeed a delicate question. Nevertheless, the basic fact that timings of 0-1 codes are exponentially related to problem size (number of variables) means optimisation timings will eventually exceed heuristic timings regardless of machine implementation, coding, surrogate constraints, etc.

Dr. Proll is correct in indicating that the degree of slack is never the same for each constraint in reality. I felt that consideration of this factor would have hopelessly overburdened an already extensive task of experimentation. I can only invite Dr. Proll and others to illuminate the effect of variable slackness by conducting additional investigations. My intuitive prediction would be that variation in slackness would decrease the number of 'effective' constraints since the most severe constraints should dominate the path to optimisation. Unfortunately the three problems of Petersen presented by Proll do not provide conclusive evidence one way or the other as to the effect of variability of slackness.

As stated in the paper the specific regression results are recommended only for problems in the range of the factors B and C as studied. Rather obviously, regression should be performed over the ranges of importance to a specific user if larger problems are involved. The examples shown by Dr. Proll do indeed exhibit a divergence of Z_0/Z_u predicted from the Z_0/Z_u actual. Nevertheless a broader, systematically generated sample would give us a better idea as to whether the divergence exhibited is specific to the problems reported or more general in nature.

Quite frequently, problems in integer programming do not allow meaningful 'intermediate' feasible solutions. This is the case in problems with equal c_j values in certain formulations of line balancing, multi-project scheduling, set covering, and trim loss problems. Thus Balasian techniques do not always yield 'a sequence of feasible solutions of increasing value'. Nevertheless this is indeed sometimes the case as Proll points out. There have been several techniques proposed (and used) for initialising branch-and-bound codes including the 'quick-trick' and linear programming starts of Geoffrion and Salkin-Spielberg, the heuristics of Holcomb, Lemke-Spielberg as well as the Byrne and Proll procedure. It would be of particular interest to see a systematic comparison of all these techniques with regard to their ability to find a feasible solution quickly. However, I strongly suspect they would nearly always outperform the original Balasian search for feasibility since it was not designed for special efficiency. It would also be interesting to see a comparison of the techniques' capacity to reduce final optimisation times. Other promising avenues of research include applying 0-1 heuristic techniques to general integer programming (e.g. the Senju-Toyoda heuristic in Kochberger, McCarl, Wyman (1973) and the Byrne-Proll heuristic in Bedenbender (1972)), and the efficiency of alternative Balasian codes on specific problem classes (Patterson, 1973).

The main point of my paper is really that a feasible solution is frequently identical to an optimal solution at least in the eyes of a manager, if not in point of fact. Hence there are circumstances where the use of heuristics needs no apology and in fact may be economically superior to the use of optimising algorithms.

References

BEDENBENDER, R. J. (1972). General Integer Programming: A Heuristic Algorithm. Unpublished MBA Paper, Pennsylvania State University.

HOLCOMB, B. D. (1968). Zero-One Integer Programming with Heuristics, IBM contributed program (360D-15.2.011).

KOCHBERGER, G. A., MCCARL, B. A., and WYMAN, F. P. (1973). A Heuristic for General Integer Programming, *Decision Sciences*, (forthcoming).

LEMKE, C. E., and SPIELBERG, K. (1968). DZIP1, Direct Search Zero-One Integer Programming 1, IBM contributed program (360-D-15.2.001).

PATTERSON, J. H. (1973). Zero-One Integer Programming: A Study in Computational Efficiency and a State of the Art Survey, Unpublished Manuscript, Pennsylvania State University.

SALKIN, H. M., and SPIELBERG, K. (1969). DZLP, Adaptive Binary Programming, IBM contributed program (360L-15.001).

To the Editor
The Computer Journal

Sir

I was interested to read in the August issue of the *Journal* the two papers on mixing interpretive and machine code. As you mention, the generation of mixed code is a relatively new topic and my experience with a project in this area may help to supplement the available information. The project concerned the development of a POP-2 compiler on a CDC 6000 Series machine.

As POP-2 is an interactive language it requires the compiler to co-exist with user programs; thus the system area consisting of compiler and run-time functions and structures is mixed with the user area consisting of user defined functions and data structures. The design philosophy of POP-2, being centred on the unit of the function, allows a 'POP-2 abstract machine' to be defined by a minimum of 10 operation codes. These codes consist of stack and jump instructions. In the original implementation of POP-2 the operation codes were machine instructions or machine extracodes, and thus the compiler generated only machine code. However, for these same codes to be implemented satisfactorily on other machines usually requires that the compiler produces interpretive code. This requirement was almost a necessity on the CDC 6000 machine where it was found that a typical abstract machine code took 2 to 4 CDC words (each of 60 bits) to execute, as opposed to the half word occupied by an equivalent code to be interpreted. The use of interpretive code introduces a factor of about 4 into CPU usage but as this is not an all-important factor in the implementation of an interactive language it was clear that only interpretive code should be produced by the POP-2 compiler.

However, the relationship between the system and user areas is closer than at might first appear. The functions in the system area have a similar structure to the functions defined by the user and as almost all of them can be written in POP-2 they could assume an identical structure and be similarly interpreted. Thus in practice the line between system and user is arbitrary and the question arises as to whether system functions should be in machine code or interpretive code. The method chosen to implement POP-2 on the CDC allows a choice to be made.

The solution to the problem of transition areas is that adopted by most POP-2 compilers. As the basic unit of execution is the function, this is also taken as the unit for use of code type, and all entries to and exits from the interpreter are made in small pieces of machine code on function entry and exit. This approach means that there are no overheads (greater than usual) for running only machine code. Calling a machine code function from an interpretive code function causes an extra address to be placed on the stack. This address (the return link) is an address in the interpreter itself and thus exit from the machine code function automatically enters the interpreter. While this approach is obvious and elegant it may not be selective enough.

The generation of mixed system code followed a similar approach to that of Dakin and Poole with one significant difference. Using a macro-processor (ML/1) a number of macros were created which defined a high level system language remarkably similar in many respects to POP-2 and named MLPOP. The result of the text translation process was a low level language (christened L3) which in many respects was similar to an extended POP-2 abstract machine. The L3 operations were simple enough to be defined as macros by the CDC macro assembler, and the choice could be made to generate machine code or interpretive code. Thus while the system portability is maintained at the high level, the code type selection and generation is kept at a low level.

The relationship between the user and the system area has already been pointed out and it is interesting to see how this relationship is maintained at all levels as illustrated in the diagram.

$$\begin{tikzcd} \text{user} \arrow[r] \arrow[d, Leftrightarrow] & \text{POP-2} \arrow[r] \arrow[d, Leftrightarrow] & \text{`abstract machine code'} \arrow[d, Leftrightarrow] \\ \text{system} \arrow[r] & \text{MLPOP} \arrow[r] & \text{L3} \end{tikzcd}$$

With the significance of the development of microprogrammable computers pointed out by Dawson this relationship at the abstract machine level suggests that POP-2 will be an ideal candidate for the benefits of microprogramming.

Yours faithfully
K. J. MACCALLUM

43 Woodland Gardens
Isleworth
Middlesex
24 September 1973

To the Editor
The Computer Journal

Sir

As a member residing in the US, I get no favours. The August issue has reached me on October 10. For this I have no complaint; for your