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.

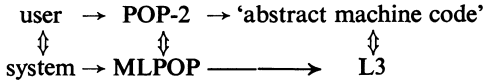
As POP-2 is an interactive language it requires the compiler to co-exist with user programs; thus the system area consisting of compiler and run-time functions and structures is mixed with the user area consisting of user defined functions and data structures. The design philosophy of POP-2, being centred 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.



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