CILLIERS, J.J. and KING, R.P. Steady-state simulation and microcomputers: Meeting and challenge. APCOM 87. Proceedjngs of the Twentieth International Symposium on the Application of Computers and Mathematics in the Mineral Industries. Volume 2: *Metallurgy*. Johannesburg, SAIMM, 1987. pp. 63 – 70.

Steady-state Simulation and Microcomputers: Meeting and Challenge

J.J. CILLIERS and R.P. KING

Department of Metallurgy and Materials Engineering, University of the Witwatersrand, Johannesburg

A powerful, interactive metallurgical process simulator for use on microcomputers has been developed in the Department of Metallurgy and Materials Engineering at the University of the Witwatersrand. MICROSIM presently allows simulation of all ore-dressing flowsheets and a selection of hydrometallurgical unit operations, and is extendable to all such processes.

MICROSIM is written in Pascal. This modern language provides very versatile data structures which can be effectively used to solve the major problem in generalised simulation programs, namely storage and transmission of data relating to liquid and solid phases. Dynamic memory allocation allows optimal utilisation of computer storage space.

The simulator utilises very efficient precalculation algorithms that determine the tear streams for optimal sequential modulator calculation convergence.

Microcomputer graphics simplifies flowsheet specification, and form-filling data capture with supplied default values eliminate most common input errors.

Introduction

The use of steady-state simulation in the metallurgical industry has shown tremendous growth during the 1980's. Not only have simulators moved from the research and consulting environments into industry, but simulation has come to be recognised as a viable engineering tool.

In order to keep ahead of the increasingly demanding requirements of industrial users, simulators have had to move from their mainframe bases to the friendlier personal computer (PC) environment. Users have become used to software that practically runs itself. To be successful, simulators cannot require data files to be written, but must rather use natural interaction, such as graphics and form-filling data capture.

The success that industry has had with ore-dressing simulation has further shown the need for simulators that can accurately mimic hydrometallurgical operations, and will in future be able to deal with complete metallurgical circuits, including pyrometallurgy.

With all these requirements in mind the University of the Witwatersrand developed the MICROSIM system. MICROSIM is at present being used in industry at several installations, and is considered extendable to their future needs.

Let us consider some of the difficulties associated with the move to microcomputers; specifically the microcomputer environment and the data structures, tearing and ordering routines and user interfaces required for the simulation of advanced flowsheets.

Microcomputers

Having a complete simulator on a microcomputer has both advantages and disadvantages. The system being smaller and slower than a mainframe means that code must be compact and efficient and that algorithms and data structures must conserve memory space.

The advantages of using a PC include the creation of much friendlier software with extended graphics capabilities and simplified data capturing such as form-filling. With a dedicated system at the disposal of the metallurgist, results are often produced faster, and certainly cheaper, than on a time-shared, non-local mainframe. Furthermore, users are generally not required to learn operating systems, or a simulator control language.

By exploiting the microcomputer environment, the simulator has gained local acceptability to plant personnel and operating metallurgists. A microcomputer-based simulator is directly available to them and easy to use, as operating tools should be.

Flowsheets

Introduction

The success that users unskilled in simulation techniques have had in ore-dressing simulation in industry has brought the realisation that successful simulation of hydrometallurgical operations could yield great rewards for both management and metallurgists alike.

The simulation of hydrometallurgical operations does however produce a host of associated problems not previously encountered in ore-dressing simulation. Two of these are the data structures appropriate for hydrometallurgical operations, and efficient and reliable tearing and ordering algorithms. The question of suitable hydrometallurgical unit operation models is a subject that deserves a discussion of its own, but is beyond the scope of this paper. It should however be mentioned that many good hydrometallurgical models for gold extraction unit operations have been developed lately[1] and have been incorporated into the simulator with great success.

Data structures

In order to mimic a true process stream, it is required to describe each component of that stream separately. As different parts of a stream are treated differently by different unit operations, the process stream may be divided into separate parts, or 'substreams'. The concept of 'substreams' was described by Britt[2] and used in the ASPEN system. In MICROSIM, each separable phase is represented by a separate substream. This is a very elegant way of representing stream information.

MICROSIM is written in Pascal (specifically Turbo-Pascal). This new generation language allows very efficient usage of memory, as data structures may be initialised dynamically when required. It was primarily for these extended data structuring capabilities and its general acceptability that Pascal was selected as a suitable language for the simulator.

Pascal allows for the creation of a record, this being a data structure capable of describing a certain concept. Thus each substream may be represented by a substream record. To simulate a complete process stream, the appropriate substream records for that stream may be combined into a stream record. As substreams consist of separable phases, a leach feed stream, for example, would therefore consist of solid and lixiviant substreams. The product from a filter would however only contain a

lixiviant substream.

As with all programs, the dimensions of the problem are required before the problem can be solved. This is indeed why it is so difficult to write general purpose software. Using FORTRAN, this problem has often been solved by having a program write a suitable declaration section as the main program, compiling this section and linking it to the remainder of the precompiled, general subroutines, or by simulating dynamic data structures with long arrays.

In Pascal, memory may be allocated dynamically during execution, thereby allowing much greater flexibility in programming and execution of the simulator. These dynamic data structures are essentially the plex structure proposed by Evans, Joseph and Seider[3].

In MICROSIM, the plex is implemented in the following way:

All substream records are defined as dynamic data structures. These substreams may be brought into existence by initialising a pointer to that specific substream, which accesses the complete record.

Therefore to create streams, substream pointers may be combined into permutations of substreams that adequately mimic the true stream.

In order dynamically to create complete streams, further pointers may be allocated to records containing basic stream information: its type (or combination of substreams), the descriptive dimensions and the source and destination units. This record further contains a choice of pointers to stream type records - records consisting of combinations of substream pointers that mimic the true process stream. Figure 1 illustrates this general data structure used to describe any process stream.

To bring a process stream into existence for a simulation, the procedure is as follows:

Knowing the stream number, n, pointer n in an array of pointers is initialised, creating the first descriptive level (Figure 1). On this level, a choice of all possible stream type pointers exists. Knowing the required stream type, the correct pointer is initialised, thereby creating the second descriptive level, a record of pointers in the correct combination of substreams.

Initialising each of the pointers on the second descriptive level brings the required substreams into existence on the third descriptive level, containing the stream descriptive information.

In order to accommodate the large amounts of data often associated with solid phases, the data structure in Figure 2 was implemented. Each element of the static array of pointers, if initialised, accesses an array of real numbers. Knowing a priori how many positions will be required, the required number of pointers is initialised to bring into existence the required number of real storage positions. In this way, memory wastage is restricted to unused pointers and the unused elements of the last array initialised. Greatly varying amounts of data may be accommodated dynamically using this technique.

Tearing and ordering the flowsheet

To effect any sequential modular simulation, the calculation order of each unit operation in the process is required. Should the circuit contain recycles, some recycles have to be 'torn' or assumed in order to start the calculation. Each unit operation in the simulation is then sequentially calculated until the torn streams converge to a solution.

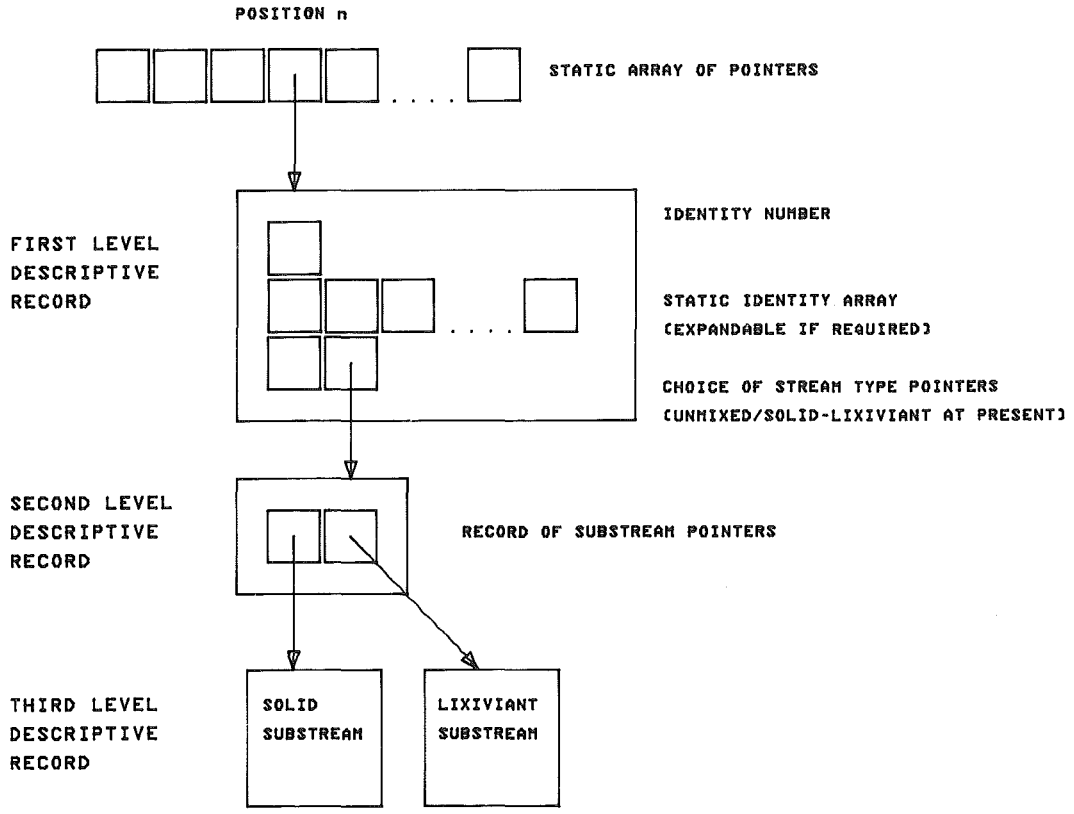

FIGURE 1. Schematic representation of a solid-lixiviant stream

The literature abounds with algorithms for the successful ordering and tearing of recycles. Upadhye and Grens[4,5] have largely solved the problem of optimal tear streams for convergence acceleration. They define a valid decomposition as a set of torn streams that open all cycles at least once. A redundant decomposition is a valid decomposition from which at least one stream can be removed without rendering the resulting decomposition invalid; a non-redundant decomposition has no such stream. They propose an exact method based on dynamic programming to determine the non-redundant decompositions. This is a very efficient algorithm, but uses large amounts of memory. A more subtle restriction , however, is the largest integer representable by the computer, generally restricting the algorithm to flowsheets containing sixteen or fewer cycles. This is due to the way the algorithm represents the 2^n-1 'states' reached during the algorithm, each state requiring an integer reference number.

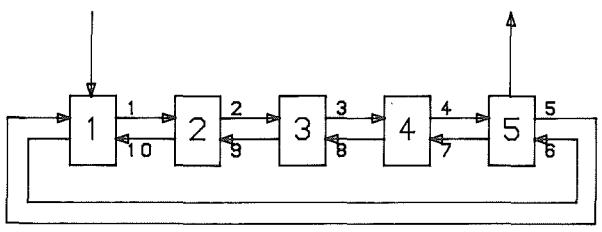

FIGURE 2. Schematic of solid substream

Probability methods for selecting tear streams were not used, as the general requirements of a branch and bound algorithm for certain flowsheets could lead to large programs and possibly long calculation times.

Another exact algorithm which has been generally rejected owing to its large combinatorial problem is that of Lee and Rudd[6] where streams and sets of streams are considered for 'containment' in other stream or sets of streams, and are subsequently eliminated for tearing.

The Lee and Rudd algorithm was modified

to reduce the combinatorial problem by the assumption that no redundant tear set shall be smaller in size than the smallest non-redundant tear set. In this way, sets of possible tear streams simply have to be tested for validity; if a valid tear set is found, all other possible tear sets of this size are generated to determine possible redundancy. All possible tear sets in the circuit do not however have to be generated to find the smallest non-redundant tear set.

This algorithm is very efficient and can accommodate any flowsheet with any number of recycles. The Pascal set operations make the generation, addition and comparison of sets trivial and reduces the code size considerably.

The modified algorithm has been verified using the examples of Upadhye and Grens[5], in all cases producing the smallest non-redundant tear set. A further test containing 27 cycles is shown in Figure 3, which is too large for the Upadhye and Grens algorithm.

The determination of cycles is done using the path-searching algorithms of Tarjan[7,8], the former considered by Evans[9] to be the best currently available. As the algorithm operates recursively, its coding in Pascal is very compact and efficient.

Once the stream-cycle matrix is known, tearing of the flowsheet is done, as described. The calculation order is determined by simple searching for units of which all feed streams are system feeds, tear streams or have been previously calculated. For the simulation to proceed, this is the only information required.

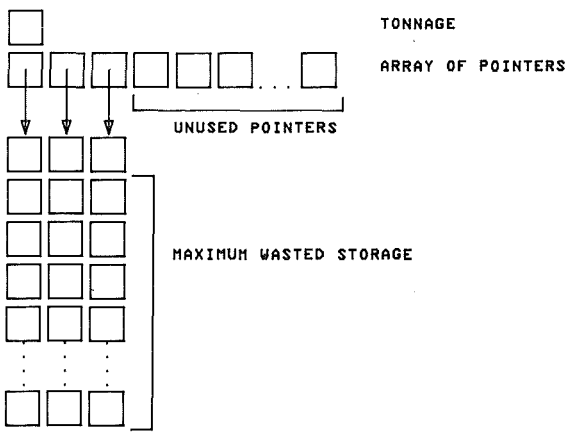

FIGURE 3. Strip process

Interfaces

The importance of ease of use of any simulation package must not be underestimated and may well be the most important factor in determining the usefulness of a simulator.

The user interface provides the means whereby a user interacts with the executive part of the program and provides the objects and methods required to perform the task efficiently.

The MICROSIM user interface was developed to accommodate two user types: those users performing routine simulation studies and people actively involved in the modelling of unit processes and the development of new simulation techniques[1]. While the first group of users would be satisfied using the models and facilities provided with the simulator, the second group would wish to utilise new models or extend the present system. Both groups thus expect a package that communicates effectively while being used (the external interface), while the second group further requires facilities for the insertion of additional unit models and the effective use of existing program fragments (the internal interface).

The MICROSIM external interface is based on graphical input and output with data capture through form-filling. The graphical interface uses command driven dialogue during the flowsheet description phase, units and streams being directly represented on the screen with icons. Stream and unit drawing may be combined, e.g. a stream

may be started, a unit drawn and the stream completed. As the keyboard is used for input, each operation requiring a single keystroke, some familiarity is required for efficient use. An online help facility reduces this need for familiarity to a minimum.

Once the flowsheet has been drawn, it is removed from the screen and data pertaining to the specific flowsheet are requested. No irrelevant or unnecessary data are required. All data input is screened for errors before being accepted, eliminating 'finger trouble' mistakes, for example where letters instead of numbers are entered or a set of entered values does not sum to an acceptable total. As a microcomputer is used, each keystroke may be checked individually before being submitted to the program, a facility at present not available on mainframe computers.

The use of form-filling data capture allows the program to supply default values for all the parameters pertaining to a unit model. These defaults provide a valuable guide for inexperienced users, as a simulation may be set up and executed with very little data supplied by a user merely exploring the capabilities of the system.

The essential requirement of the internal interface is that users must be able to insert their own models and icons into the simulator without difficulty. As the data structures for stream description are so flexible and versatile, they must be accessible to users to tailor to their own specific needs.

The program simplifies the understanding of the simulator structure at source code level through three techniques:

Technical documentation illustrates the program structure, the data structures and the methods for insertion of graphical icons and/or unit models by simple example.

The use of structured programming allows the program to be broken down into separate modules that do not interact. The addition of unit models or icons restricts changes to a single module,

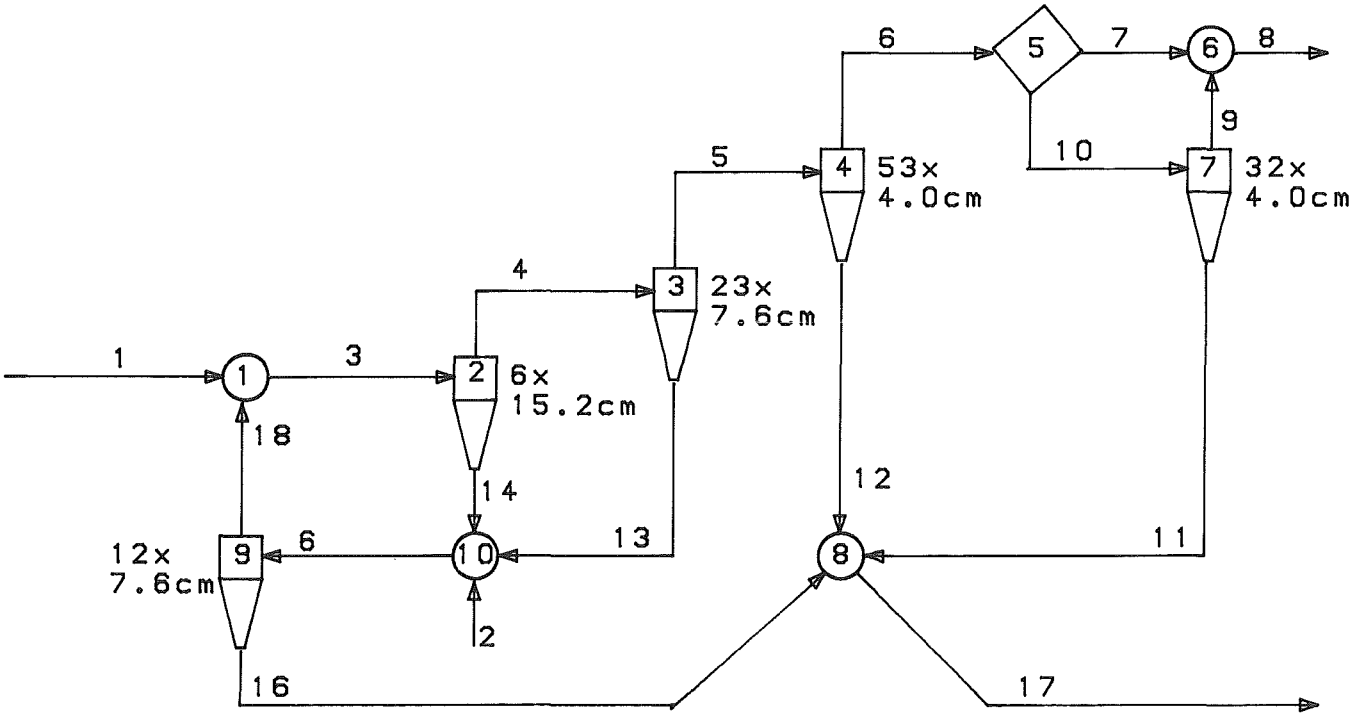

FIGURE 4. Renison fine classification circuit

while data structure alterations will clearly have a wider effect.

Source code is written to be understandable and readable with liberal comments and the use of descriptive variables and procedure names.

The combination of these techniques by the user makes user specific additions to the general simulator simple and efficient.

Example

As an example of a plant simulation, the Renison tin desliming circuit as discussed by Kavetsky and McKee[10] was simulated. The flowsheet is shown in Figure 4. The feed size distribution was assured to be Rosin-Rammler, with k = 12 and n = 1,29.(2) Fifteen size classes were used, with a top screen size of 180 μm and a √2 reduction in size.

The Plitt[11] model was used for all the cyclones, with a standard geometry. The circuit converged to within 0,01% in five iterations using direct substitution. Mass balance results are shown in Table 1.

These results compare very well with the results obtained by the original study. Closer correlation can be obtained by the manipulation of the cyclone spigot diameter in the model.

TABLE 1

RESULTS OF EXAMPLE SIMULATION

Strm No	Mass Flow-rate TPH	% Solids	Grade %	Cassiterite Recov %
1	50.00	33.30	2.21	100.00
2	-	-	-	-
3	77.14	29.67	2.11	147.42
4	42.06	23.15	1.79	67.95
5	23.08	17.44	1.57	32.86
6	11.93	12.03	1.43	15.49
7	5.96	12.03	1.43	7.74
8	9.47	10.81	1.41	12.06
9	3.50	9.22	1.36	4.32
10	5.96	12.03	1.43	7.74
11	2.46	21.25	1.54	3.43
12	11.14	33.67	1.72	17.37
13	18.97	38.42	2.04	35.10
14	34.91	44.89	2.78	87.91
15	53.89	34.29	2.52	123.01
16	26.74	56.50	3.12	75.59
17	40.35	43.85	2.64	96.38
18	27.14	24.72	1.93	47.42

Conclusion

The MICROSIM process simulator was designed to operate in the PC environment, while not neglecting any of the features required for effective metallurgical process simulation. Data structures allow the representation of any metallurgical process stream while tearing and ordering algorithms can accommodate any flowsheet.

The user interface is designed for ease of use and flexibility on both the internal and external levels allowing the simulator to be altered with ease at source code level as required. The use of graphical input and form-filling data capture supplying default values makes the simulator accessible to the most inexperienced user.

On these fronts, MICROSIM has met the challenge of microcomputer simulation in the mineral industry.

References

1. STANGE, W. M.Sc. thesis in preparation. University of the Witwatersrand, Johannesburg, 1985.
2. BRITT, H.I. Multiphase stream structures in the ASPEN process simulator. In Foundations of Computer-aided Chemical Process Design, Vol. 1, 471-510. R.S.H. Mah and W.D. Seider. Engineering Foundation, New York, 1980.
3. EVANS, L.B., JOSEPH, B. and SEIDER, W.D. System structures for process simulation. AIChE J. 23, 1977. p.658-666.

4. UPADHYE, R.S. and GRENS, E.A. II An efficient algorithm for optimal decomposition of recycle streams. AIChE J. 18, 1972. p.533-539.
5. UPADHYE, R.S. and GRENS, E.A. II Selection of decompositions for chemical process simulation. AIChE J. 21, 1975. p.136-143.
6. LEE, W. and RUDD, D.F. On the ordering of recycle calculations. AIChE J. 12, 1966. p.1184-1190.
7. TARJAN, R. Depth-first search and linear graph algorithms. SIAM J. Comput. 1, 1972. p.146-160.
8. TARJAN, R. Enumeration of the elementary circuits of a directed graph. SIAM J. Comput. 2, 1973. p. 211-216.
9. EVANS, L.B. Advances in process flowsheeting systems. In Foundations of Computer-aided Chemical Process Design, Vol.1, 425-468. R.S.H. Mah and W.D. Seider. Engineering Foundation, New York, 1980.
10. KAVETSKY, A. and McKEE, D.J. Analysis and design of industrial grinding and classification circuits by use of computer simulation. Proc. 18th Int. Symp. on Application of Computers and Mathematics in the Mineral Industries. Instn. Min. Metall., London, 1984. p.57-68
11. PLITT, L.R. A mathematical model of the Hydrocyclone classifier.CIM Bull. 69, Dec. 1976, p.114-123.