Comp Jnl 2 1959 A108

Generating Strategies for Continuous Separation Processes

by E. E. Bernard and P. D. A. Mole

Summary: In designing a best continuous separation process in the field of chemical engineering, it is often necessary to be able to specify all the possible designs, in order to evaluate each for construction and operating cost. Hand methods of generating these designs involve two risks: either (1) not getting all the possible designs, as the result of non-systematic generation, or (2) having to tolerate many duplications, as the result of too weak conditions. This paper reports a general method of generating all, and only, the distinct designs for any such process, and includes results of the method for mixtures of up to 10 components.

In order to find the most economic scheme for separating an n-component mixture by continuous fractionation, it is necessary first to know what and how many distinct arrangements of $n-1$ columns there are. In this paper a simple method for generating such distinct arrangements on a digital computer is reported, and it is shown that the method generates all possible distinct arrangements.

A mixture of n components $C_1, C_2, \ldots, C_n$ is regarded as being strictly ordered by the separation factor S of each component, so one can write

$$S_1 < S_2 < \ldots < S_n,$$

where each S_i is a real number. Clearly, if for some (say) two components C_i and C_j in the mixture, we have that $S_i = S_j$, $i \neq j$, then they cannot be separated by such a process, and can be regarded in this context as a single component. The process consists in first partitioning the mixture into two parts $C_1, C_2, \ldots, C_k$ and $C_{k+1}, \ldots, C_n$ where, for some real number P, $S_k < P < S_{k+1}$. Each of the two parts is similarly partitioned, and so on, until the entire original mixture is separated into its individual components. Moreover, since every component will be isolated, there will be a real number P interpolated between each ordered pair of components, giving a new strictly ordered set

$$S_1 < P_1 < S_2 < P_2 < \ldots < P_{n-1} < S_n.$$

It is significant that this new sequence has exactly $2n-1$ elements, n separation factors, and $n-1$ numbers P.

By a "strategy" will be meant the way in which the sequence of P's is chosen. For example, if the original mixture contains three components C_1, C_2, and C_3, with $S_1 < S_2 < S_3$, one can first interpolate a P between C_1 and C_2, or first between C_2 and C_3. These are the two possible strategies, and may be illustrated diagrammatically as:

These diagrams illustrate simply enough what is intended by "non-equivalent", and it is clear that they are the only two non-equivalent strategies for a mixture of three components.

Every "net", or diagrammatic representation of a strategy, will be regarded as orientated on the page in the same way as those in the above example, with the first partition at the top. By an "echelon" will be meant a horizontal cross-section through an array of vertical lines in such a net. It is then stipulated that any vertical line can lie in only one echelon. This stipulation results in no loss of generality, and one can then refer to the first echelon, last echelon, etc., with the obvious meaning of top, bottom, etc., as the net is orientated on the page. This convention will be of great convenience in all that follows.

In every net, two types of lines will be identified:

(1) a vertical line which terminates in a "branch" (i.e. a horizontal line), and

(2) a vertical line which does not.

In each of the above nets there are two lines of type (1) and three of type (2).

With each type, a binary digit is associated by

Rule 1: The digit "1" is associated with each type-(1) line.

Rule 2: The digit "0" is associated with each type-(2) line.

Thus, digits are associated with the above nets as follows:

These arrays of digits are then written as sequences by reading down and from left to right, resulting in 11000 and 10100, respectively. Such a sequence of digits will be called a "word"; this is the way every net will be represented and, therefore, the way every strategy will be represented. It is immediately evident that the associated word is unique; with every net there is associated just one word. For the converse, a slightly weaker statement is needed; if a word represents a net at all, then it represents only one net.

The physical interpretation of the net is of course given by the method of constructing a net to represent a strategy. The lines of type (1) represent partition

points (the *P*'s in the sequence) and the lines of type (2) represent final, isolated components (the *S*'s in the sequence). As there were $n - 1$ partitions for a mixture of n components, it is clear that every net for n components will have $n - 1$ lines of type (1) and n lines of type (2). Hence, the associated word will have $n - 1$ ones and n zeros, and will contain $2n - 1$ digits in all. Moreover, every net begins with a line of type (1); therefore every word begins with a one. It is clear that every branch must terminate in a pair of type-(2) lines; in particular the rightmost branch must terminate in a pair of type-(2) lines and, therefore, every associated word must terminate "00."

By a "proper initial sub-word" will be meant a consecutive string of digits contained properly within the word (i.e., not the whole word itself), and whose first digit is the first digit of the whole word. Thus, the word 10100 contains the proper initial sub-words 1, 10, 101, and 1010. Similarly, a "proper initial sub-net" would be a net contained in a larger net, which begins at the first echelon of the larger net, and is not the larger net itself. It is apparent that no net can properly contain an initial sub-net, for the requirement that every branch of the sub-net terminate in lines of type (2) prohibits that branch being continued; continuation of the branch would require that at least one of the type-(2) lines be altered to a type-(1) line. It follows then that no word associated with a net can contain a proper initial sub-word, which is also associated with a net.

Now one can define a "well-formed net" as a net

1. which begins with a type-(1) line;
2. all of whose branches terminate in type-(2) lines;
3. which consists of $n - 1$ lines of type (1) and n lines of type (2); and
4. which does not contain a proper initial sub-net satisfying conditions 1, 2, and 3.

Similarly, a "well-formed word" is a word

5. whose first digit is a one;
6. whose last two digits are zeros;
7. which consists of $n - 1$ ones and n zeros; and
8. which does not contain a proper initial sub-word satisfying conditions 5, 6, and 7.

The problem originally stated in the opening paragraph of this paper can be put more succinctly: to generate all distinct well-formed nets, whose number of terminal lines is equal to n, for any integer n. A method of unique representation of nets by words has been given, so the problem reduces to one of generating all the well-formed words, whose number of digits is $2n - 1$, for any integer n. This can be done by considering all possible words, for some particular n, and then applying conditions 5 to 8, as selection criteria, to each word as it is generated. All possible words, of $2n - 1$ digits, are given by the sequence W of binary integers

$$1\ldots11 = 2^{2n-1} - 1$$
$$\vdots$$
$$0\ldots00 = 0.$$

This set W contains all possible words of $2n - 1$ digits, so it certainly contains all the well-formed words for n components. By applying the selection rule that every word in W which does not begin with a one is rejected, exactly half of the words are thrown out, and a new set X remains:

$$1\ldots11 = 2^{2n-1} - 1$$
$$\vdots$$
$$1\ldots00 = 2^{2n-2}.$$

Clearly, W–X contains no well-formed word, for every word in W–X begins with a zero, and no well-formed word does so.

Three-fourths of the words in X are rejected by applying the selection rule that every word in X which does not end with a pair of zeros is rejected. This follows from the fact that only the integers congruent to zero (mod 4) end in a pair of zeros, and therefore a new set Y is formed:

$$1\ldots00 = 2^{2n-1} - 2^2$$
$$\vdots$$
$$1\ldots00 = 2^{2n-2}.$$

By a similar argument to the above, it is clear that Y contains all the well-formed words. A still tighter upper bound can be put on the set of well-formed words, by noting that the largest well-formed word is

$$1\ldots10\ldots00 = 2^{2n-1} - 2^n,$$

and the smallest well-formed word is

$$101010\ldots10100 = 2^{2n-2} + 2^{2n-4} + \ldots + 2^2.$$

The new set Y' of words, containing those in Y which lie between these bounds (inclusively), was generated by a Ferranti Pegasus computer (at the Electronic Computing Laboratory, University of Leeds), by the simple process of successive additions of 2^2 to 2^{2n-2}. Selection rules corresponding to conditions 7 and 8 were then applied to each word so generated, and every word which passed the criteria was printed out as a binary integer. Application of these last two criteria to each word of Y' will cause the rejection of some, and the set Z of words which remains will pass all the criteria. That $Z \neq Y'$ is clear by considering the word 11100001100, which satisfies the first three conditions for a well-formed word, but not the fourth, and does not represent a net. Also, by considering the word 11100, which satisfies conditions 5, 6, and 8, but not 7, and does not represent a net, it is seen that conditions 7 and 8 are independent.

Since the selection criteria are, in essence, the definition of "well-formed word," it is clear that every element of Z is a well-formed word, and by the same kind of argument given for condition 5, it is seen that Z contains all the well-formed words. That the elements of Z are all (and only) the words which represent well-formed nets is clear from the definitions of "well-formed net" and "well-formed word," and the unique assignment of a word to a net.

To illustrate the results of the computation, the well-formed words for $n = 5$ are:

111100000	110011000
111010000	110010100
111001000	101110000
111000100	101101000
110110000	101100100
110101000	101011000
110100100	101010100

The total number of well-formed words, for each $n = 2, 3, \ldots, 10$, is:

n	well-formed words
2	1
3	2
4	5
5	14
6	42
7	132
8	429
9	1430
10	4862

The computer program for generation, selection, and printing of these words is relatively short; it can be held entirely within the computing store of the Pegasus while the computation is taking place. As seen in the table above, the number of words becomes rather large when the number of components is greater than five. Because, in ordinary practice, mixtures of greater than five or six components are seldom encountered, it did not seem worth while to reproduce the full tables here; should any reader desire them, the tables or the program can be obtained from the authors.

APPENDIX

The Number of Nets

by E. S. Page

Let the number of nets defined in the preceding paper (Bernard and Mole, 1958) for a mixture of n components be u_n. If the first partition divides the mixture into k and $n - k$ components the number of nets with this first partition is $u_k u_{n-k}$. Hence

$$u_n = \sum_{k=1}^{n-1} u_k u_{n-k}. \tag{1}$$

Define the generating function $U(x) = \sum_{k=1}^{\infty} u_k x^k$; then

$$U^2(x) - U(x) + u_1 x = 0.$$

Since $u_1 = 1$, $u_2 = 1$, $U(x) = [1 - (1 - 4x)^{1/2}]/2$.

Hence
$$u_n = \frac{(2n-2)!}{n!(n-1)!} \tag{2}$$

Stirling's formula for $n!$ gives

$$u_n \sim 2^{2n-2}/(\pi n^3)^{1/2}, \tag{3}$$

which shows the rate of increase of u_n as n increases.

This derivation of the number of nets suggests a method of enumeration different from that given by Bernard and Mole. The storage required limits its utility for a computer.

On Taking the Square Root of a Complex Number

The square root of a complex number provides an interesting example of the concealed dangers in using algebraic coding. If $(u + iv)^2 = x + iy$ the ordinary formulae for u and v are

$$u = \sqrt{\{\tfrac{1}{2}[\sqrt{(x^2 + y^2)} + x]\}}$$
$$v = \sqrt{\{\tfrac{1}{2}[\sqrt{(x^2 + y^2)} - x]\}}.$$

Any attempt to code these formulae using single-length floating arithmetic such as that normally used by automatic coding routines will lead to serious error whenever y is small. This is caused by the cancellation which occurs in one or other of the expressions $\sqrt{(x^2 + y^2)} \pm x$. If the error of the inner square root is ε, the error in the final result can be as much as $\sqrt{(\frac{1}{2}\varepsilon)}$. Thus, to get single-length accuracy from these formulae it is not enough to find the single-length square root of a double-length number: the inner square root must itself be found to double-length accuracy.

Once this danger has been appreciated there is, of course, no real difficulty in getting round it. With some machines it is not difficult to find the inner square root to double-length accuracy, and this allows a very straightforward program. On others it may be more convenient to evaluate only one of u and v from the formula (the one in which no cancellation takes place) and to find the other from the equation $2uv = y$.

C. Strachey