CELIA - A CELLULAR LINEAR ITERATIVE ARRAY SIMULATOR

by

R. W. Baker and G. T. Herman
Department of Computer Science
State University of New York at Buffalo

ABSTRACT

CELIA is a program for simulating linear arrays of cells, each one operating under the same set of rules. The action of a cell is influenced by both its neighbors. Thus CELIA can be used for the simulation of linear iterative networks of identical modules.

The authors' main application, however, is in the field of biology, where the program can be used to test hypotheses about the developmental rules for organisms. For this reason, the division of one cell into several is allowed. If developmental rules based on the hypotheses are fed in as data to the program, it will give as output the case history of any desired number of developments, as well as statistical data concerning the development. These then can be checked against experimental results and thus the validity of the hypotheses can be established. For well validated hypotheses the program can also be used as a predictor.

The program is demonstrated by an application concerning the developmental rules for blue-green algae.

1. MOTIVATION

In recent years a great deal of effort has been expanded on the part of biologists, computer scientists and electrical engineers in studying the behavior of iterative arrays of identical modules or cells. (Apart from a very large number of papers, there are at least three books entirely devoted to this topic.[2,3,7]) A cell is assumed to have finitely many possible states, and it will change its state depending on the states of some of its neighbors. All cells in an array are assumed to be synchronized, i.e., they change their states simultaneously.

Models of this kind have provided the biologist with a plausible framework within which problems of development and self-reproduction can be discussed[2,3,4,8,9,11,12,14]. For the electrical engineer networks composed of identical modules are attractive because they are economical to manufacture, change and repair.[7,10] The parallel nature of the model makes it suitable for investigation of parallel processing both at the programming and at the microprogramming level.[3,13] In fact, it has been argued[13] that cellular logic arrays should serve as basis to much of computer hardware design. Problems of synchronization, like the firing squad synchronization problem[1], are of interest to both biologists and computer scientists.

In all this work it has often become necessary to simulate the behavior of an iterative array. Many papers explicitly mention this fact[1,3,11] and it is obvious from others that some computer simulations have been done to produce some of the examples by which theorems are demonstrated. It is clear that simulation of one iterative array must be basically similar to the simulation of another one, and hence much of the simulation effort described above must have been duplicated. There is a case therefore, for writing a computer program which contains the essence of an iterative array simulation and can be used for simulating a large variety of iterative arrays with a minimal effort on the part of the user. Since many biological phenomena can be translated into the language of iterative arrays, such a program also helps in simulating biological situations which until now have been investigated by other means.

Let us now list a number of situations in which a cellular iterative array simulator is useful. It is the last of these which we consider the most significant.

(a) To find out the way a cellular iterative array will develop is a very tedious job without the help of a computer. In fact, in all but the simplest cases time would put a stop to a pencil and paper attempt to follow such a development. Yet, this is often very important. For instance, assume we have observed a certain pattern of development in an organism and we wish to give developmental rules which would explain this pattern of development. In most cases, this is a trial and error process. Another situation where such experimenting with developments is extremely useful is when we have

conjectured a theorem about development, but we do not yet see how to prove it. The observation of a large number of developmental sequences might give us a clue why our theorem holds or provide us with a counter example if it does not. Complicated theorems (and many of the theorems in this area are proved using long and tedious constructions) can be checked, by observing on examples that we did not miss some small points in our proof. (Such a check led to a revision of an original proof in[9].) In trying to understand the proofs of others, following those proofs on some examples can be an enormous help. As the most extreme case of this use of simulation, Balzer[1] has proved the validity of his solution to the firing squad problem by simulating the first 99 cases and then using induction for the rest.

(b) Given an iterative array of logical circuits, we may be interested in whether or not it eventually gets into an equilibrium state or a cycle. This can easily be decided by simulating the array on a simulator which has a cycle-detecting capability. In such cases the regularity of a network can also be decided.

(c) Once a model of some biological phenomenon is fully tested, our simulator can be used as a predictor. This way we can get a good idea of the development of some organism under new circumstances, or test the behavior of a circuit under certain input conditions.

(d) The most sophisticated, as well as most significant, use we have found for our simulator is the testing of biological hypotheses. For example, a biologist may be interested in the circumstances under which a particular kind of cell appears in an organism. After some observations he may formulate a hypothesis regarding the generation of this kind of cell. The problem is, how to test the validity of such an hypothesis. One particular method is to incorporate the hypothesis into a simulation program, run this program long enough to collect significant information on the consequences of the hypothesis (e.g., building up a table on the distribution of distances between the given kind of cell) and then check these consequences against real life experimental data. In such fashion we can obtain a validation or reputation of the hypothesis.

We shall give examples of how our particular simulator has been used in the kind of situations described above.

2. THE BASIC MODEL AND DESIGN AIMS

We have decided that in this first simulator we shall restrict our attention to linear arrays (i.e., arrays in which all but the first and last cell has a unique predecessor and a unique successor). The justifications for such a restriction are the following.

(a) There are linear iterative arrays which have universal computing power, i.e., they can simulate the behavior of any digital computer whatsoever.[8]

(b) A very large portion of the research which is being done on iterative arrays is concerned with linear arrays, this is true particularly for most of the authors' research and for the research of the people most closely associated with them.

(c) One can often represent non-linear phenomena by linear arrays.[12] We shall give an example later on.

(d) The basic programming techniques which we applied to linear arrays would not carry over to two or more dimensional arrays.

(e) There is a theory for linear cellular iterative arrays whose applicability to biology has been well demonstrated.[8,9,11,12,14] It is far from clear what the precise analog of this theory should be in the manydimensional case. There are as yet no authorative works in this direction.

We shall describe other assumptions we have made, and give justifications for them while describing our basic model. This model has originally been suggested by Lindenmayer[11] and a mathematical theory for it has been developed, among others, by Herman[8,9] and Lindenmayer.[12]

In this model, which from now on will be referred to as Lindenmayer model, we deal with linear arrays of cells with the following properties.

(a) Each cell is capable of having one of finitely many states. The justification for assuming a finite number of states in a biological system is that there usually are threshold values for parameters that determine the behavior of the cell. Thus with respect to each of these parameters it is sufficient to specify two conditions of the cell: "below threshold" and "above threshold", although the parameter itself may have infinitely many values. Even if this is not

possible, it is nearly always reasonable to approximate the value of the parameter by one of finitely many predetermined values. In applications to circuits the finiteness of the number of states is a basic assumption of the underlying theory.

(b) The next state of any cell is determined by its present state and the states of its two immediate neighbors. This would sometimes be done in a probabilistic fashion.

This assumption that a cell is affected directly only by its own neighbors and the effect of any other cell can only reach it via these neighbors is reasonable in most situations. However, it rules out the possibility of instantaneous information transmission along a filament say by the surface tension on the exterior, or along an iterative array of circuits with no delays between them.[10] One may argue that no information transmission of this type is really instantaneous and provided our unit of time is taken small enough, we can observe how the change is transmitted from cell to cell. This can however lead to problems of essentially different time scales for the various phenomena we wish to represent. We shall discuss how such problems can be resolved.

(c) The change of state takes place simultaneously along the whole array.

This is usually assumed to be the case in a (synchronous) electrical circuit. In biology it seems somewhat unnatural, but when we consider that one possible next state can always be the present state, we can see that asynchronous operation can easily be simulated.

(d) The change of state of the two end cells of the array is partially determined by the state of the environment at that end. The environment does not have a direct influence on the intermediate cells.

The first part of this feature is blatantly reasonable, but the second one is a somewhat unnatural restriction. The only justifications for it are that it makes the programming essentially easier and that it has not often been found necessary to have it otherwise. It is true that in iterative circuits[7,10] each cell has a "primary input", but since it is usually kept constant during a computation it can be incorporated into the rules determining the behavior of cells. In the many biological applications of Lindenmayer models there is only one example where sensitivity to the environment by each cell was found necessary, that is in the work of Surapipith and Lindenmayer.[14]

(e) During a change of state a cell may divide into two or more cells or disapper altogether.

This is a feature which makes Lindenmayer models so suitable for simulating biological development. In other models[2,3] growth can only take place at the end of the array. This is very restrictive from a biological point of view. In computer terminology the difference between the Lindenmayer[11] and the von Neuman[2,3] conception is the same as between an array and a list. (This particular analogy becomes even more illuminating when we compare von Neuman's fixed two-dimensional array with Lindenmayer's branching filaments.)

The above description of Lindenmayer models should be sufficient for a good intuitive picture. A precise mathematical definition has been given by Herman.[8]

Our object was to simulate Lindenmayer models as described above, bearing in mind the kind of applications listed at the end of §1. This has resulted in several additional design aims.

(a) A large variety of notation should be acceptable for describing states of cells both in the input and in the output.

There are many reasons for this. One is when one wants to check the work of others, it greatly simplifies matters if one can use their notation. But this varies from person to person. Codd[3] uses numbers, Balzer[1] uses letters and Lindenmayer[12] uses both as well as symbols like /,), (. In many cases the use of different kinds of symbols is well justified, by making the output more readable. (E.g., / is used by Lindenmayer to denote an oblique cell wall, rather than a 'real' cell.) Another reasonable sort of notation is A1, A2, .., A9, B1, B2, ..., B9 to distinguish between the different states of the same cell type.

(b) The program should have the ability to test for cycles in development.

This is necessary in the electrical engineering applications in order to find out whether or not a network is regular. It is also important in biological applications, since it is our main tool for solving the problems

caused by the different time scales. For instance, in an example which will be described in detail later on we shall be interested in the concentration of a certain chemical in the biological cells. After a cell divides, there is a redistribution of this chemical in the whole filament which takes something in the order of 3 minutes. As opposed to this, the life cycle of a cell is about a day long. In order to simulate the redistribution of the chemical with any precision we need a time unit which is about a six thousandth of a day. In order to get statistically significant data, we need to observe the development for at least 10 days. That is 60000 units of time. If a long array is to be manipulated this many times by the program the simulation would take an extraordinarily great length of time. Also note, that during most of this time there would be no change in the simulating process as far as the distribution of the chemical is concerned. There are two possible ways of avoiding this difficulty. One is to assume that the redistribution of the chemical is instantaneous. (Such a simulation has been carried out for the same organism by Dr. Richard Gordon at the Center for Theoretical Biology at the University of Buffalo.) This would bring in an additional error in the simulation. The alternative is to check for a steady state. This will appear as cyclic behavior in the distribution of the chemical (with a small cycle length) due to the fact that our measure of the concentration is only an approximation. If a cycle is detected, time can be updated to the next time when the life cycle of a cell is at an end. This way it is possible to simulate reasonably cheaply biological situations in which events are occurring on two time scales.

(c) The program must have the ability to gather statistics and to present it to the user in a reasonable form.

This is essential for the biological hypothesis testing application. Although the user can write his own statistical subroutines, we have decided to incorporate as standard feature the ability to tabulate certain phenomena (in the same sort of fashion as a GPSS TABLE) and also to give a histogram (similar in appearance to the CSMP histogram) of the table. Examples of the kind of things that maybe tabulated are:

(i) Distance between certain kinds of cells.

(ii) Length of strings of consecutive appearance of certain kinds of cells.

(iii) Number of occurrences of certain kinds of cells.

(iv) Relative number of occurrences of certain kinds of cells (relative to all the cells in the array at the time the measurement is taken).

(d) There must be the ability to stop simulation upon the occurrence of certain kinds of cells.

This is needed for instance when the occurrence of a certain kind of cell signifies the death of an organism.[8] In simulating the firing squad[1] the appearance of a firing state should signify the end of the simulation. Codd[3] uses the appearance of a certain state as a halt instruction for his cellular computer.

(e) The user must have the ability to simulate a number of similar Lindenmayer models within the same computer run with a minimal amount of programming effort on his part.

The situation when a number of similar models have to be tested arises in a variety of situations. For instance, when one has a fair idea of the developmental rules which will cause a particular kind of organism to develop, but wants to make the rules more precise, experiments with variants of the rules is the only way. Similarly, when the regularity of a network is to be checked, one has to test that equilibrium is reached for all possible combinations of primary input values. Since the primary input values are incorporated into the rules determining the behavior of the cells, again we have a number of similar runs.

(f) The program should be usable in a wide variety of situations with very little additional programming effort.

3. DESCRIPTION OF THE PROGRAM

We now have a program working which satisfies all the design aims described in the last section.

Apart from certain job control type information (e.g., length of run in simulated time) the user need only write one subroutine. This is to describe for any three states how the middle one changes if we have three cells in those states. Even

here there is an option to use a standard subroutine which will read in a table describing this function. However, in most cases the function will not be random and thus a large table can be described by relatively few logical instructions. We shall refer to this subroutine as the DELTA subroutine.

At present this subroutine and any other user written subroutines (for instance the one describing the history of the environment if it is not constant) are assumed to be in FORTRAN, although this can easily be changed. FORTRAN has been chosen, since it is generally available, it is probably the best known language to scientists and it does not appear to have any disadvantages as opposed to other high level languages for this kind of application. The main (control) program is also written in FORTRAN, but all of the internal subroutines have been written in COMPASS, the CDC 6400 assembler language. (We also have FORTRAN versions of them, but on the examples we tried, this caused approximately four times longer runs. This is because of the unsuitability of FORTRAN for list processing.)

The essence of the program is the maintainence of a list representing the array of cells. From this list the state of the cells are passed one by one to the DELTA subroutine, which at any time holds three consecutive elements of the old list and returns the string of elements by which the middle one must be replaced. This will usually result in relinking the list. A list of available elements is also maintained and available space is immediately relinked to this list. This way there is no garbage collection problem at all. During this process the program checks for the occurrence of a cell indicating termination of the program and also gathers statistics if this was requested. Cycles up to any user specified length are checked for by keeping copies of the state of the list at the last instances. A copy of the present list is also available to the user for manipulation in statistics gathering.

At the simplest level, the user will use numbers to describe the states of cells, will specify the number of states, the state which is the constant environment, the state which causes termination (if any), the maximum number of steps for which the simulation should run, the initial array, and a table taking place of the DELTA subroutine. For this he will get a list of all arrays from the initial till the final (determined either by an occurrence of the specified state or by the number of steps).

In more sophisticated uses he may use a full character set option for state description, he may write subroutines for testing termination on certain states or on cycle length, he may write a subroutine for varying the environment, and he may use the statistical survey option either with the standard subroutines or his own. At no time does he have to know the internal organization of CELIA or make any changes to it.

Upon termination of one simulation control is passed to the main routine which checks whether a new simulation is requested in the job control data. One part of this data is a variable which can be referred to in the DELTA subroutine. It is by altering this variable in the job control that the user can achieve consecutive runs of similar but not identical simulations.

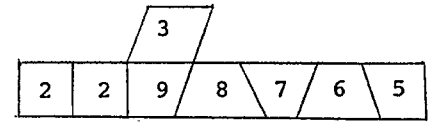
4. APPLICATIONS

We shall concentrate on the details of a biological hypothesis testing application. However, before doing that we wish to give a demonstration of our claim that the restriction to linear arrays does not preclude the study of more complicated organisms.

Lindenmayer [11,12] has discussed developmental rules for the red alga Callithamnion Roseum Harvey. This is an organism which has branches growing out from the sides of some of its cells and also it may have transverse or oblique cell walls between cells. Yet such an organism can be described by a linear array. This array contains extra, non-biological cells, namely: to represent a transverse cell wall, / to represent an oblique cell wall, left parenthesis to represent the beginning of a branch and right parenthesis to represent the end of a branch. Numbers will represent actual cells. If we further agree that branches grow on alternative sides and oblique cell walls are tilted in alternative directions, then,

(2 : 2 : 9 (3) / 8 / 7 / 6 / 5)

represents the organism



Lindenmayer[12] describes a set of developmental rules for this organism and the corresponding DELTA subroutine is given in Figure 1. Figure 2 is the output we got for a simulation which stopped after 16 steps and Figure 3, is the corresponding branching array. There are two particular interesting aspects of this example. One is that the next state of each cell depends on the cell only and not on its neighbors. Second is that it would be impossible to embed such branching pattern into a von Neumann[2,3] type fixed two dimensional space.

We have used one program for testing and developing theorems in many of the references[1,7,8,9,10,11,12] as well as in other works. We shall forego detailed description of these applications.

Our most interesting application was in the area of biological hypothesis testing. What we describe now is more for the purpose of demonstrating how our program can be

applied, than for giving the final answer to an open problem of biology. For the latter purpose a lot more experimental and simulation work is still to be carried out.[4]

The problem we are dealing with is the problem of heterocyst formation in blue-green algae. Certain blue-green algae are linear arrays (filaments) of three kinds of cells: vegetative cells, heterocysts and spores. Under certain circumstances vegetative cells turn into heterocysts. Fritsch[6] called heterocysts a "botanical enigma". As Wolk[15] says "although the observations of a number of investigators have supported various suggestions as to the non-reproductive functions of these cells, no definite elucidation of their function has yet been presented." The problem is that there has been very little work done in studying all but the immediate consequences of any of the rather numerous suggestions and so there is hardly any evidence to justify or to reject any of them. We shall show how our program can help in this direction. We shall concentrate on one particular hypothesis, one which has been based on the work of Fritsch[6] and Fogg[5]. Again we quote Wolk[15].

"Fritsch stated as working hypothesis that the primary function of heterocysts is secretion of growth stimulatory substances. He further proposed that when these substances decrease sufficiently in concentration in the region between two heterocysts which are becoming increasingly widely spaced because of vegetative growth, a new heterocyst forms."

"That nitrogen acts to inhibit the formation of heterocysts might mean, Fogg suggested, that a vegetative cell becomes a heterocyst when its intercellular concentration of a specific nitrogenous inhibitory substance, probably ammonia or simple derivative, falls below a critical level. He further suggested that periodic concentration gradients of such inhibitory substance exist along the length of a filament, and that heterocysts form at points where inhibition is minimal."

On this basis we built a model for heterocyst formation in blue-green algae. (Here the authors received continuous advice and help from Dr. Richard Gordon. Many of the basic biological ideas are due to him. However, whatever faults the model may have from the biological point of view is most likely due to simplifications by the authors of the fairly complex situation presented to them by Dr. Gordon.)

We ignored spores altogether, since under certain conditions they do not appear. So our filaments consist of nothing but vegetative cells and heterocysts. Heterocysts do not divide, concentration of the inhibiting chemical in them does not vary with time or from heterocysts to heterocyst. The concentration, c, of the inhibitory chemical in a cell varies proportionately to the difference between the concentration in that cell, the concentration in the cell on its left (l), the concentration on the cell on its right (r) and the concentration in the environment (e). Thus

$$\frac{dc}{dt} = k_1(l-c) + k_1(r-c) + k_2(e-c)$$

$$= k.(l + r + e - 3c)$$

where we have assumed $k = k_1 = k_2$. (We can think of this as the chemical flowing across the walls of a cell at a speed propotional to the difference in concentration on either side.)

In a short length of time Δt, the change in concentration will be Δc, where

$$c = k.\Delta t.(l + r + e - 3c).$$

(Note the implicit assumption: the rate of flow across the external cell wall is the same as across walls between cells. An alternative interpretation is that we have assumed that the permeability coefficient of the intercellular walls, k_1, equals the specific rate of destruction of the chemical, k_2. e is then proportional to instead of equal to, the concentration in the environment.)

Vegetative cells have a life cycle of approximately a day. At the end of their life cycle they divide into two daughter cells each of the same concentration as the mother cell. However, if the concentration of the inhibitory chemical is below a certain threshold, say t, instead of dividing, the cell will turn into a heterocyst. We will assume that this is the mechanism by which heterocysts form.

Wolk[15] has tabulated the length of inter-heterocyst intervals following growth in the presence and absence of ammonium. We shall investigate whether his experimental results are sufficient to refute the validity of model as outlined above. We shall restrict our attention to the case when ammonium is absent (i.e., e=0).

In our simulation the state of a cell had two components. One was the concentration of the inhibitory substance within it. This had a range from 0 to 999, with 999 being the concentration in heterocysts, which we assume is held constant by the heterocyst. The other was the number of units of time it still had to go before the end of its life cycle. Because of the relatively great speed of the passage of the inhibitory chemical as compared to a life cycle, we gave this the range from 0 to 9999. (0 for heterocysts.)

A cell whose state is denoted by the number 3875521, is a cell with a concentration 387 which still has 5521 units of time to live.

It would be unreasonable to assume that two daughter cells of the same cell will have exactly the same length of time to live. In our program we have assumed that

the life cycle of cells is normally distributed with standard deviation being 10% of the mean. The mean we took to be 6000 time units. Thus we use a probabilistic next state function.

In each step the time component of the state is reduced by 1 (except when it is already 0) and the concentration c changes into

$$c + \Delta c = c + k.\Delta t.(1 + r - 3c).$$

First we had to decide on the value of $k.\Delta t$. By comparing different values it was found that the value which caused steady state to be reached after cell division in about the same time what has been estimated[4] on the basis of diffusion theory is 1/4. Figure 4 shows a part of the history of the development under these assumptions. Note that steady state situations have been skipped.

Next we had to decide what the value of the threshold t should be. This we have done by running a number of similar shorter experiments with various thresholds. The one which gave the most approximate mean inter-heterocyst interval was t=3.

Using this assumption we made a longer simulation run giving us a table of a total of 43 inter-heterocyst distances. (See Figures 5 and 6). This table we have compared with the table of Wolk[15] based on experiments. Using a χ^2-test we found that the hypothesis that the two tables report on two samples from the same distribution can only be rejected with about 80% confidence. This means that even if our model was perfectly valid, worse discrepancies between experiment and observation than what we have found would occur on average one out of five times. This is not strong enough evidence to reject the model and the underlying biological hypothesis.

So the Fritsch-Fogg hypothesis together with the details we have filled in passed at least one test of validity. Naturally, before it is accepted as authoritive much more experimental and simulation work has to be done. We hope that we demonstrated that CELIA can be a great help in such work.

5. PROJECTED FUTURE WORK

The program CELIA has already been found useful in a wide variety of situations. However, it has not been used very long and it may have some weak points which have so far remained hidden. Our intention is to keep applying it to genuine biological problems (the development of hydra is a possible next line of investigation) and thereby discover any weakness it may have.

As far as adding to the program is concerned, there is only one projected addition in the near future. This is to make the program suitable to deal with the branching patterns of Lindenmayer[11]. Since our internal list structures have already been built doubly-linked with this extension in mind, no difficulty of implementation is foreseen at this stage.

Extension to genuinely many dimensional arrays is not envisaged at this stage. Some simulation work is being done in this direction by many people on particular organisms, but the underlying model is not sufficiently uniform to justify a nonlinear version of CELIA.

ACKNOWLEDGEMENTS

The authors wish to thank Dr. R. Gordon for his help and advice on biological matters and Mr. E. Artzy who wrote our COMPASS list processing subroutines.

REFERENCES

1. BALZER, R.
An 8-state minimal time solution to the firing squad synchronization problem, Information and Control, v. 10, 1967, pp. 22-42.

2. BURKS, A. W. (ed.)
The Theory of Self-Reproducing Automata, by John von Neumann, University of Illinois Press, Urbana, Ill., 1966.

3. CODD, E. F.
Cellular Automata, Academic Press, New York, New York, 1968.

4. GORDON, R. & WALSHBY, A. E.
Development of the spacing pattern of heterocysts in Anabaena, in preparation.

5. FOGG, G. E.
Growth and heterocyst production in Anabaena Cylindrica Lemm. II. In relation to carbon and nitrogen metabolism, Annals of Botany, N.S., v. 13, 1949, pp. 241-259.

6. FRITSCH, F. E.
The heterocyst: a botanical enigma, Proceedings of the Linaean Society, London, v. 162, 1951, pp. 194-211.

7. HENNIE III, F. C.
Iterative Arrays of Logical Circuits, MIT Press, Cambridge, Mass., 1961.

8. HERMAN, G. T.
The computing . ability of a developmental model for filamentous organisms, Journal of Theoretical Biology, v. 25, 1969, pp. 421-435.

9. HERMAN, G. T.
The role of environment in developmental models, Journal of Theoretical Biology, to appear.

10. KILMER, W. L.
On dynamic switching in one-dimensional iterative logic

networks, Information and Control, v. 6, 1963, pp. 399-415.

11. LINDENMAYER, A. Mathematical models for cellular interactions in development, I. Filaments with one-sided inputs, II. Simple and branching filaments with two-sided input. Journal of Theoretical Biology, v. 18, 1968, pp. 280-293 & 300-315.

12. LINDENMAYER, A. Developmental systems without cellular interactions, their languages and grammars, to appear.

13. SHOUP, R. G. Programmable Cellular Logic Arrays, Ph.D. dissertation, Carnegie-Mellon University, 1970.

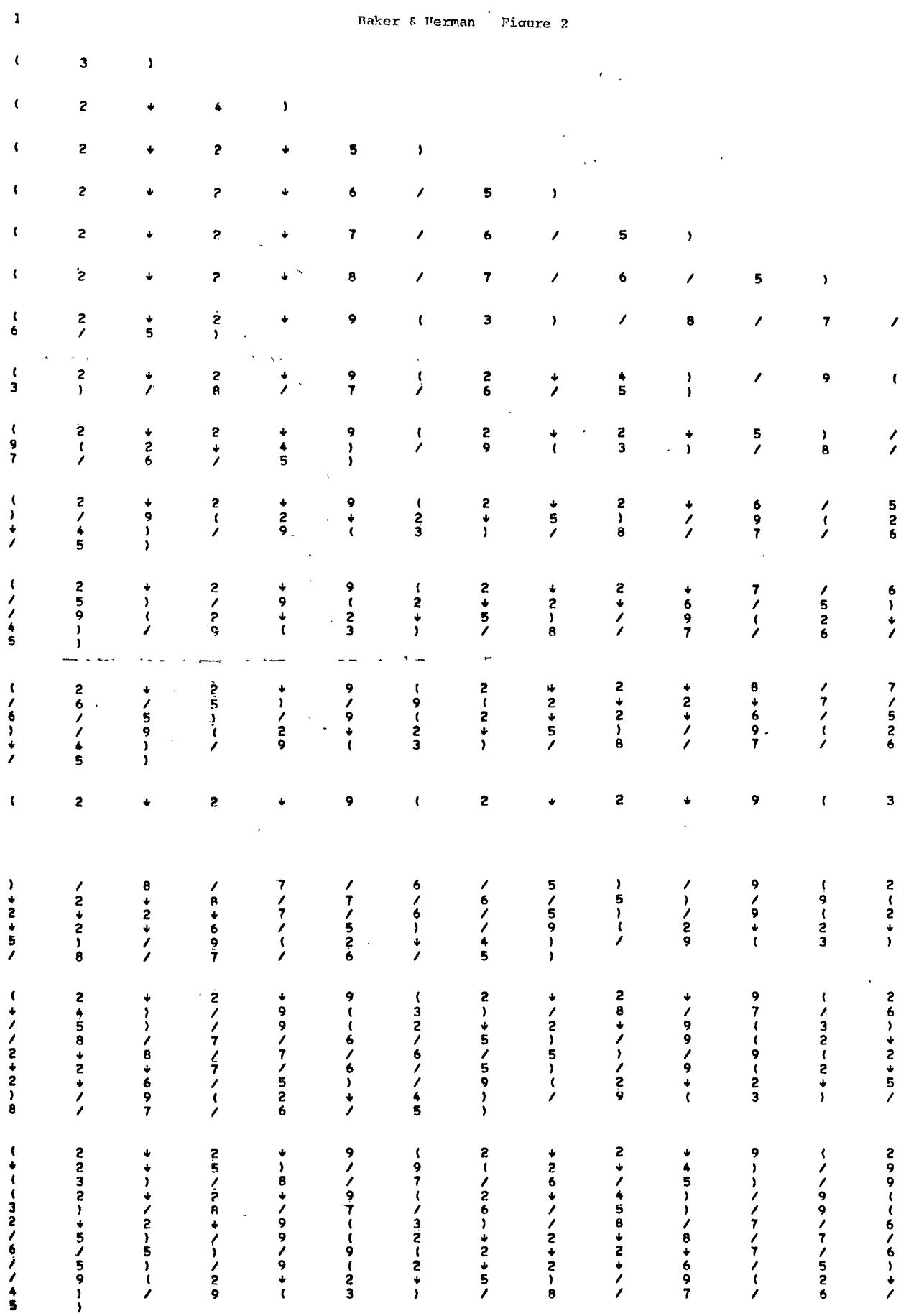
14. SURAPIPITH, V. & LINDENMAYER, A. Thioguanine-dependent light sensitivity of perithecial initiation in Sordaria fimicola, Journal of General Microbiology, v. 59, 1967, pp. 227-237.

15. WOLK, P. C. Experimental Studies on the Development of a Blue-green Alga, Ph.D. dissertation, Rockefeller Institute, 1964.

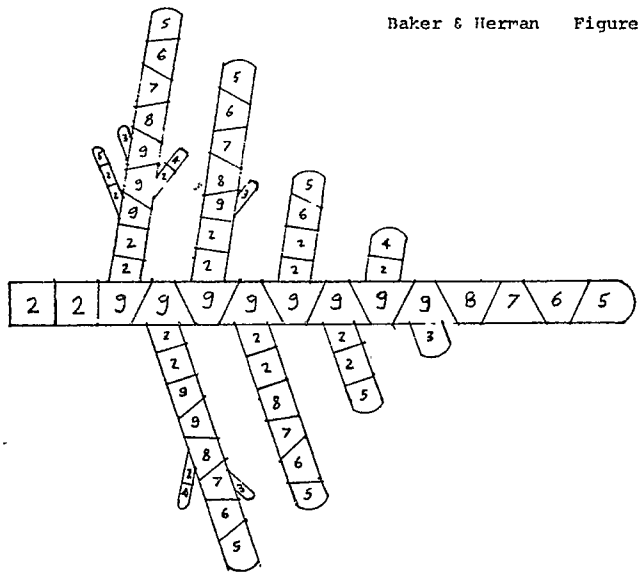
Present address of R. W. Baker is:
Engineering Department
Pfaudler Company
Division of Sydron Corporation
Rochester, New York

Baker & Herman Figure 1

```
000010          SUBROUTINE CELTA(L,M,R,N,MM)
                INTEGER R,MM(1),BLANK,RP,TCW,OCW
000010          DATA BLANK/4R          /,LP/8R(       /,RP/8R)
000010          DATA TCW/8R+       /,OCW/8R/       /
000010          IF(M.NE.8R1       ) GO TO 30
000012             N = 3
000012             MM(1) = LP
000014             MM(2) = 8R3
000015             MM(3) = RP
000017             RETURN
000017       20 N = 1
000020             MM(1) = M
000021             RETURN
000022       30 IF(M.NE.8R3       ) GO TO 40
000024             N = 3
000024             MM(1) = 8R2
000026             MM(2) = TCW
000027             MM(3) = 8R4
000031             RETURN
000031       40 IF(M.NE.8R4       ) GO TO 50
000033             N = 3
000033             MM(1) = 8R2
000035             MM(2) = TCW
000036             MM(3) = 8R5
000040             RETURN
000040       50 IF(M.NE.8R5       ) GO TO 60
000042             N = 3
000042             MM(1) = 8R6
000044             MM(2) = OCW
000045             MM(3) = 8R5
000047             RETURN
000047       60 IF(M.NE.8R6       ) GO TO 70
000051             N = 1
000051             MM(1) = 8R7
000053             RETURN
000053       70 IF(M.NE.8R7       ) GO TO 80
000055             N = 1
000055             MM(1) = 8R8
000057             RETURN
000057       80 IF(M.NE.8R8       ) GO TO 20
000061             N = 4
000061             MM(1) = 8R9
000063             MM(2) = LP
000064             MM(3) = 8R3
000066             MM(4) = RP
000067             RETURN
000070             END

SUBPROGRAM LENGTH
000152

FUNCTION ASSIGNMENTS

STATEMENT ASSIGNMENTS
```



Baker & Herman Figure 3



Baker & Herman Figure 4

```
1  9990000 4035274 2044975 2046026 4035703 9990000
1  9990000 4025273 2044974 2046025 4025702 9990000
1  9990000 4025272 2034973 2036024 4025701 9990000
1  9990000 4025271 2034972 2036023 4025700 9990000
1  9990000 4020299 2035607 2036153 2031051 4020728 9990000
1  9990000 4020298 2035606 1536152 2031050 4020727 9990000
1  9990000 4020297 1905605 1416151 1901049 4020726 9990000
1  9990000 3990296 1845604 1316150 1841048 3990725 9990000
1  9990000 3960295 1795603 1266149 1791047 3960724 9990000
1  9990000 3940294 1765602 1226148 1761046 3940723 9990000
1  9990000 3930293 1745601 1196147 1741045 3930722 9990000
1  9990000 3920292 1725600 1186146 1721044 3920721 9990000
1  9990000 3920291 1715599 1166145 1711043 3920720 9990000
1  9990000 3910290 1715598 1156144 1711042 3910719 9990000
1  9990000 39102R9 1705597 1156143 1701041 3910718 9990000
1  9990000 39102R8 1705596 1156142 1701040 3910717 9990000
1  9990000 3916142 3916393 1705308 1155854 1700752 3910429 9990000
1  9990000 4456141 2396392 1705307 1155853 1700751 3910428 9990000
1  9990000 4226140 2146391 1325306 1155852 1700750 3910427 9990000
1  9990000 4106139 1936390 1165305 1055851 1700749 3910426 9990000
```

Baker & Herman Figure 5

ENTRIES IN TABLE 43 MEAN ARGUMENT 12.86 STANDARD DEVIATION 3.77

INTERVAL COUNT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER
0	0	0.00	0.00	100.00
1	0	0.00	0.00	100.00
2	0	0.00	0.00	100.00
3	0	0.00	0.00	100.00
4	0	0.00	0.00	100.00
5	0	0.00	0.00	100.00
6	0	0.00	0.00	100.00
7	1	2.33	2.33	97.67
8	5	11.63	13.95	86.05
9	6	13.95	27.91	72.09
10	1	2.33	30.23	69.77
11	3	6.98	37.21	62.79
12	5	11.63	48.84	51.16
13	3	6.98	55.81	44.19
14	6	13.95	69.77	30.23
15	3	6.98	76.74	23.26
16	2	4.65	81.40	18.60
17	2	4.65	86.05	13.95
18	2	4.65	90.70	9.30
19	1	2.33	93.02	6.98
20	3	6.98	100.00	.00

Baker & Herman Figure 6

```
  ----------------------------------------------------------------------
 0 I
 1 I
 2 I
 3 I
 4 I
 5 I
 6 I
 7 I*
 8 I*****
 9 I******
10 I*
11 I***
12 I*****
13 I***
14 I******
15 I***
16 I**
17 I**
18 I**
19 I*
20 I***
```