COMPUTER SIMULATION OF A LIVING CELL: PART II*

R. WEINBERG† and M. BERKUS‡

Department of Computer and Communication Sciences, University of Michigan (USA)

(Received: 10 June, 1970)

SUMMARY

This paper, which is being presented in two parts, is concerned with the problem of a realistic and useful simulation of a living cell, the simple unicellular bacterium Escherichia coli, the program being written in Fortran IV for an IBM 360/67 computer. The object is to represent the cell in such a way as to simulate its growth in real environments of a living cell, as well as during changes from one chemical environment to another. This simulation could also supply useful information for answering current questions in biology.

In Part 2 of this paper the authors describe the method of evolving DNA and discuss cancer in relation to the computer simulation under consideration. Literature references cited in both parts of the paper, as well as the appendix to the work, are also presented.

SOMMAIRE

Cette étude s'occupe du problème d'une simulation réaliste et utile d'une cellule vivante: la bactérie simple et unicellulaire Escherichia coli, le programme étant écrit en Fortran IV pour un ordinateur IBM 360/67. Le but c'est de représenter la cellule de telle façon à simuler la croissance de celle-ci dans le milieu véritable d'une cellule vivante ainsi que pendant les changements d'un milieu chimique à un autre. Cette simulation pourrait aussi fournir des renseignements utiles en répondant à des questions courantes dans la biologie.

* Part I appeared in *Bio-Medical Computing*, **2** No. 2, April, 1971.

† Supported by the Department of Health, Education and Welfare, NIH, Bethesda, Maryland (USA).

‡ Present address: Dept of Statistics and Computer Science, Kansas State University, Manhattan, Kansas (USA).

Bio-Medical Computing (2) (1971)—

4. COMPUTER SIMULATION OF EVOLVING DNA

The computer simulation of a living cell adapts phenotypically to three different chemical environments (section 2, Part 1). I will extend the simulation so that the cell can adapt genetically as well as at the phenotypic level. I will represent DNA as an array in the computer in which are stored the indexes and values of various rate constants in the equations representing the simulated cell.

Four powerful genetic operators for evolution of populations are (1) crossover, (2) inversion, (3) mutation and (4) dominance.

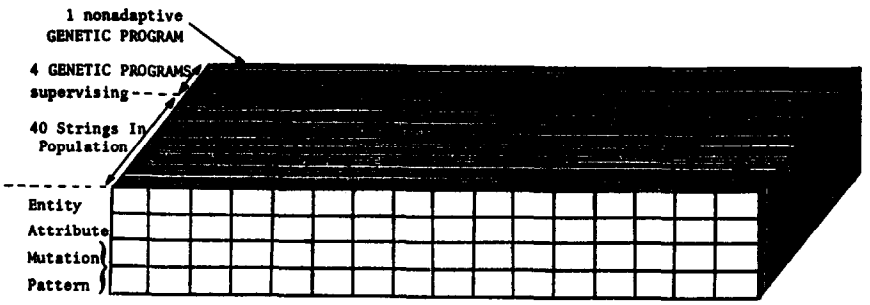

Fig. 4.1. The Chrom Array.

15 Entities Referenced by 1 String = 1 Individual
Column Number = Position of Entity Reference on String
= 2nd Dimension of Array
$1 \leq$ Column Number ≤ 15

Row Number = 1st Dimension of Array. $1 \leq$ Row Number ≤ 4
Row Number :1 = Index of Entity Referenced = Locus Referenced
2 = Attribute of Entity Referenced = Value of Locus
3 and 4 Describe Mutation Pattern for Entity Referenced in Row 1
3 = Number Controlling Interval over Which Random Number is Generated for Mutation
4 = Index of Probability Distribution over Increments of Mutation for Monte Carlo Method.

INDIVIDUAL IN POPULATION = STRING IN POPULATION IS INDICATED BY THE THIRD DIMENSION OF THE ARRAY. THE CONTENTS OF THE ROWS AND COLUMNS CONTAIN INFORMATION ABOUT ATTRIBUTES AND GENETICS OF THE INDIVIDUAL INDEXED BY THE THIRD SUBSCRIPT. e.g., *CHROM* (1,4,2) = 3 MEANS THAT THE THIRD ATTRIBUTE OF THE SECOND INDIVIDUAL IS INDEXED BY THE FOURTH COLUMN OF *CHROM* ARRAY INDICATED. $1 \leq$ INDIVIDUAL INDEXED ≤ 40.

THE LAST FOUR STRINGS INDEXED CONTAIN INFORMATION ABOUT THE GENETIC PROGRAM SUPERVISING EVOLUTION. $41 \leq$ GENETIC PROGRAM ≤ 44.

Crossing over permits preferential multiplication of groups of subroutines which interact well, giving coadaptation. Without crossing over all subroutines are equally linked on the string referencing an individual as a collection of subroutines, so that there is no such concept as 'close together on the linkage map'.

Inversion is necessary to rearrange the genetic location of different subroutines, so that those that should be close together on the genetic map get a chance to approach each other during evolution.

Mutation is necessary to explore a large genetic space, and also to regenerate attributes of functions lost through selection.

Dominance is necessary to preserve attributes of functions which are at a temporary disadvantage, but may be useful at some later time in evolution.

Duplication of individuals, as well as the genetic operators crossing over, inversion, and mutation will be simulated by operations on the contents of the arrays. The modified arrays will be used to calculate the modified rate constants by which the new populations of simulated cells grow. I will partially realise the function of dominance in my simulation by strongly directed mutation to restricted sets of mutant alleles, a mechanism not possible in real DNA, but one which realises one function of dominance, *i.e.* the conservation of genetic variability in a population.

Since the simulation is of a haploid bacterial population, I will attempt to simulate in a reasonable amount of computer storage populations of haploid bacteria, which store variability without extensive use of dominance and diploidy. This is to make easier the realisation of genetic mechanisms used by real bacterial populations, rather than because diploidy is unreasonable. Indeed diploidy offers a natural and straightforward way to realise the power of genetic operators and to store genetic variability for evolutionary demands put on the population by changing environments. Furthermore, diploidy permits storage in the form of valuable substrings of successful alleles, allowing many sampling advantages which will be diminished in my representation of a population by applying genetic operators to individuals who represent the means of probability density functions, defined by the formula for the density together with the mean and variance of the density. However, many haploid populations of bacteria exist in nature in environmental niches which are accessible to less successful dioloid competitors, *e.g.* protozoa, indicating that the haploid mechanisms are more successful than the diploid ones in certain circumstances.

An excellent feature of a general scheme like Holland's is its extreme flexibility. One can consider part of an organism as the string which forms an individual in the population, and the rest of the organism as part of the environment. Since the theoretical development is much easier for a stationary environment, I will consider all loci which are fixed during the whole evolution of the programs as the environment, and will consider only unfixed loci as strings. Since I am free to set linkage parameters as I wish, I can increase linkage to account for those fixed loci which do not explicitly appear. Most of the genetic characteristics of the individuals such as mutation rate and crossover will be represented as separate strings of adaptive or non-adaptive genetic programs, each of which will supervise the evolution of a population.

It is important to have adaptive genetic programs which can evolve, since selective procedures may lead to unexpected consequences, such as death for long-legged chickens, and should be amenable to modification. (The biological example, death for long-legged chickens, refers to an experiment in which longer shanks were selected in populations of chickens (Wallace, 1968, p. 455).) The populations so selected always became extremely unfit.

There are two detectors of phenotypic limitations on genetic evolution of the simulated cells which are particularly easy to observe. One is a wide disparity between simulated chemical concentrations of cell metabolites and the concentrations necessary for balanced growth. The second straight-forward detector of phenotypic imbalance is the inability to modify the simulated enzymes to account for the growth rates required in the three simulated environments. The impossibility of manipulating the enzymes to produce required growth rates immediately shows up as the inability to produce a solution in the solve routine of the computer program (section 3, Part 1). The inability to maintain biochemical equilibria necessary for life shows up in a departure of the ratios of the concentrations of biochemicals to the necessary concentrations. These ratios should be 1 if metabolic equilibrium is maintained, and departures of the ratios from 1 indicate instability. For this reason, the utility function which directs the rate of reproduction of each individual in the population contains the sum of (ratio + 1/ratio) in its denominator, so that the further the ratio departs from 1, the lower the value of the utility function, and the less the rate of reproduction of the individual under consideration. Inability to solve the equations for allosteric modification of the enzyme pools adds a 10 to the denominator of the utility function, so that individuals which can not use allosteric modification correctly can still be ranked as a function of how far off their ratios are.

Inducing sophisticated quantities by simple genetic operations on finite strings relates directly to Holland's description of the complex populations of schemata and operators on schemata, both conservative and nonconservative which are present in simple finite populations of evolving strings, and which confer upon these simple finite populations of strings powerful evolutionary capabilities. The complexity of calculation of average excess induced by elimination of forty percent of the population at each reproductive cycle illustrates the ease with which one can realise something which takes a good deal of effort to describe in quantitative terms, and points up the advantages of studying evolving schemata in the space of a 'successfully' evolving population of strings.

It is important to distinguish between selection induced on schemata by genetic operations on three-dimensional arrays referencing heuristic programs, and the criteria used directly on the programs themselves. To illustrate this perhaps subtle, and certainly profound, distinction, I am going to illustrate the genetic operation of selection by eliminating sixty percent of the strings each reproductive cycle, and fill in the missing sixty percent with new strings produced from the old strings not

eliminated by the genetic operators crossing over, inversion and mutation. A string's utility will be proportional to how well and how quickly the simulated cell which is the description of the string adjusts to changes in simulated environments. The environment for the evolving strings is all fixed loci represented by the equations simulating cell growth and adaptation, as well as the changing simulated environment for the cell. To return to the number judging performance by the string description as opposed to a sophisticated measure such as average excess for that string, how well and how quickly the simulated cell adjusts to changes in its chemical environment will be simply expressed as utility = 1/(a sum of ratios of current chemical concentrations compared to the desired chemical concentrations + the computer time it took the genetic operators to modify the string during evolution + 10 if the program was unable to correctly accomplish allosteric inhibition). Since the last three quantities are in the denominator of the formula for utility, the larger the deviation of the chemical ratios, the longer the time to evolve, and the greater the failure to solve for allosteric modification, the less the utility. Obviously computer time doesn't even exist in the real cell, and to destroy sixty percent of the programs is a clumsy and unsubtle procedure. However, given this environment, each string does have some average excess induced on it, which would have to be calculated over the run of the program. It is important to perceive that this average excess exists, but does not appear as a number in the running genetic program which effects the evolution of programs in the computer. If one were simulating the theory of evolution rather than the evolution of an effective program to accomplish a task, one would certainly want to calculate the average excess of each program rather than defining it implicitly by the genetic procedure for producing new programs from old ones.

It will be recalled that the utility of the best individual produced under direction of an adaptive genetic program, as judged by that adaptive genetic program is recalculated by the nonadaptive genetic program. The nonadaptive genetic program gives the utility it calculates for the description of the best string in its population to the adaptive genetic program directing evolution of that population of strings. This utility is then used by the nonadaptive genetic program to direct the evolution of the adaptive genetic programs. Elimination of unnecessary genetic operations is both a practical advantage to the programmer, and an experimental fact in competitive natural populations. Therefore the time it took the adaptive genetic program to manipulate its population to produce the best string is added to the denominator of the SUM which is the utility given to that adaptive genetic program. Much of the information contained in rows 3 and 4 of the strings CHROM column can be approximated by ignoring the cumulative frequency distribution indexed in the fourth row, and simply using the uniform distribution. An adaptive genetic program which does this will gain in utility. For example FREQ(3) may be set to a uniform frequency distribution, and effectively ignored for the jth entity during mutation since the random-number generator itself sets up a uniform

distribution by choice of the correct interval in which the random numbers are generated. If the loss in evolutionary power does not overbalance this gain in utility by economising on computer time, some of the information in the third and fourth rows of the CHROM array may be dropped from the program eventually. It was included to indicate the ease with which a general and powerful evolutionary program may be written. Since the whole growth and phenotypic adaptation procedure takes less than 3 seconds of IBM 360/67 computer time, and may be shortened, the program is not as time consuming as it might appear to be at first glance. Storage of large blocks of the program on disks or in files until needed would also economise on computer costs.

A brief consideration of the probability of replacement of a program in the population shows that the amount of utility judged to be associated with the program influences the probability that the program will be erased by its genetic supervisor. In order to be a member of the survival population, a newly generated program has to be in the best four in the population of running programs. The higher the value of one of the old surviving programs, the less likely it is to be supplanted by a newcomer in the next reproductive cycle. I do not want to discuss these calculations in detail, since my main point is to use Holland's formal theory of adaptive systems to support the validity of writing extremely simple, albeit evolutionarily powerful heuristic programs.

Sophisticated molecular interactions effecting negative feedback of metabolic processes at both the DNA and cytoplasmic level, as well as position controls of DNA and cell division enable a real cell to survive well, and to explore only a particularly productive subset of possible physiological states. The molecular mechanisms underlying many of these sophisticated relationships are often simple and direct from a molecular point of view. For example, looking to see whether DNA is in the process of replicating, and not replicating the cell unless all DNA which has begun replication has completed replication simply involves a replication site occurring at a potential site of cell division. Only when replication of the circular DNA molecule is completed can the new cell wall be laid down. This type of sophisticated limitation of possible actions also occurs in genetic modification through inertia, in that a chromosome only undergoes small changes compared to all possible changes which may occur. Which changes survive is very closely dependent on the structural and functional relationships existing in the cell, and sophisticated evolutionary schemes may well embody this information.

I will pick as unfixed variables from which to generate my population of strings (and schemata), 15 control parameters which the cell uses for phenotypic adaptation to changing environments (Fig. 3.4, Part 1). The five control constants corresponding to repression of enzyme production by DNA are $R(1), \ldots, R(5)$ (*see* appendix). The ten control constants corresponding to allosteric inhibition of enzyme activity after the enzyme has already been formed are $P1V(1), \ldots, P1V(5)$, $P3V(1), \ldots, P3V(5)$.

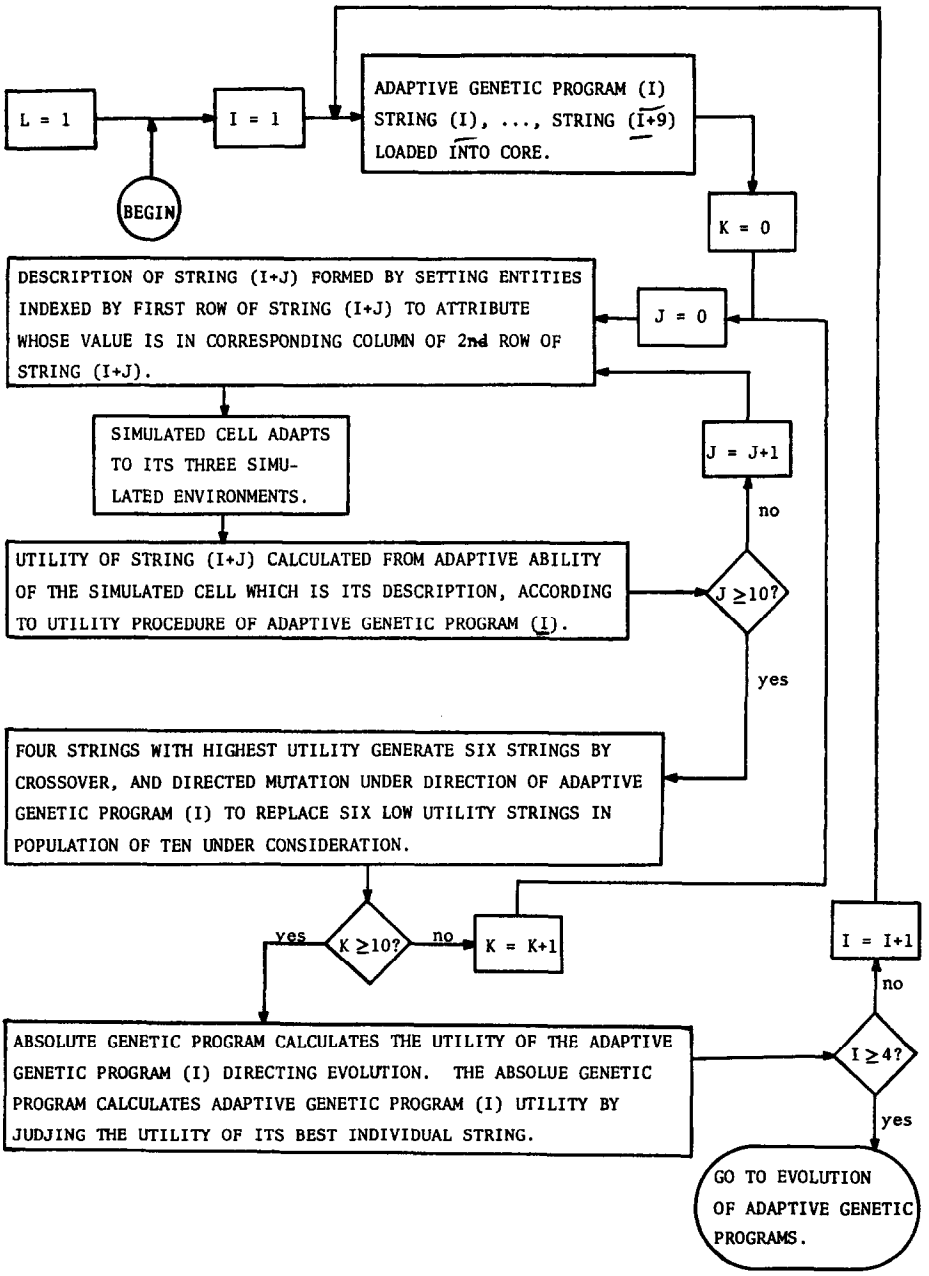

Fig. 4.2. Evolution of strings.

I will keep the simulated cell program and all variables in program common, and call in the simulated environments, adaptive genetic program, and the ten string population from disk storage. The program for the simulated cell will receive the values of its variables according to the information of a string representing one individual, the simulated cell program will be run, a utility will be calculated according to the adaptive genetic program, values of the variables will be filled in according to the next string in the population, until all of the strings in the ten string population have obtained a utility for that run. The adaptive genetic program

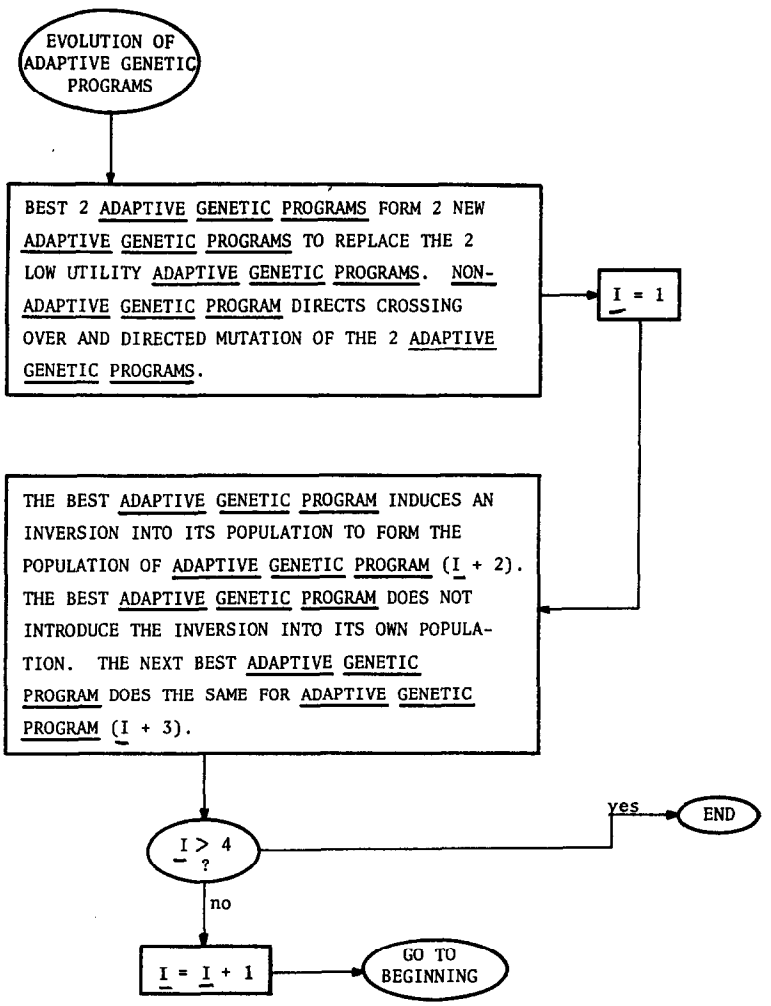

Fig. 4.3. Evolution of adaptive genetic programs.

will then operate on the ten string population to form a new population according to its genetic information and the utilities awarded to each string when its values were loaded into the simulated cell program and run. The adaptive genetic program and its ten string population will then be returned to disk storage, and the next adaptive genetic program and population will be brought into core (Figs. 3.2 and 3.3, Part 1). Each adaptive genetic program will be awarded a utility dependent on information in a non-adaptive genetic program. The four adaptive genetic programs and the non-adaptive genetic program will then be loaded into core, and the two worst adaptive genetic programs will be replaced by programs generated from the two best adaptive genetic programs. I will introduce inversions in the populations of strings supervised by the two best adaptive genetic programs, and use these modified strings to form the new populations of the new adaptive genetic programs just formed. The inversions will only appear in the populations of the new adaptive genetic programs, and will be homozygous in these populations for ease of genetic operations on the populations (Figs. 4.2 and 4.3).

Information concerning genetic manipulation of the evolving strings will be stored in the references to the adaptive genetic programs (Fig. 4.5). This information determines crossover, inversion, mutation and selection. The pattern of mutation, once the locus to mutate has been determined by the adaptive genetic program, will be referenced by the third and fourth rows of the array containing the string which will mutate. The non-adaptive genetic program directs evolution of the adaptive

INDEX OF ENTITY	ENTITY EQUALS LOCUS	IOTA = RANGE FOR ATTRIBUTE OF ENTITY INDEXED	MUTATION PATTERN EQUALS INCREMENT IN ATTRIBUTE
1	$P1V(1)$	$(-10^{-6}, +10^{6})$	+Normal: mean $P1V(1)/10$ = variance
2	$P1V(2)$	,, ,,	2
3	$P1V(3)$	,, ,,	3
4	$P1V(4)$	,, ,,	4
5	$P1V(5)$	,, ,,	5
6	$P3V(1)$	,, ,,	+Normal: mean $P3V(1)/10$ = variance
7	$P3V(2)$	,, ,,	2
8	$P3V(3)$	,, ,,	3
9	$P3V(4)$	,, ,,	4
10	$P3V(5)$	,, ,,	5
11	$R(1)$	(0, 1)	+Uniform: -1, $1.R \geq 0$. If $R < 0$, set to 0.
12	$R(2)$		
13	$R(3)$		
14	$R(4)$		
15	Utility of individual referenced by third dimension of array. Utility is calculated by the genetic program supervising evolution, so there are some empty spaces here.		

Fig. 4.4. Description of Strings References when Third Dimension of CHROM Array Ranges from 1 to 40.

INDEX OF ENTITY	ENTITY	ATTRIBUTE RANGE	RANDOM NUMBER INTERVAL AND CUMULATIVE FREQUENCY DISTRIBUTION USED TO EFFECT GENETIC OPERATION OF INDEXED ENTITY
1	crossover	(1,14)	Random (1,N(1)), Freq. (1)
2	inversion	(1,14)	Random (1,N(2)), Freq. (2) length = 2★Random (1,5)
3	mutation	(1,15)	Random (1,N(3)), Freq. (3)
4 5 6 7 8 9 10 11 12 13 14	coefficients for utility polynomial for string being operated on.	(0,1)	Random (1,N(4)), Freq. (4) etc.
15	Utility of this adaptive genetic program, calculated by non-adaptive genetic program.		

Fig. 4.5. Adaptive Genetic Programs. Adaptive genetic programs: $N(I)$ and FREQ(I) are determined by a non-adaptive GENETIC PROGRAM, which also operates on the adaptive genetic programs as evolving strings. The nonadaptive GENETIC PROGRAM determines utility by a polynomial which evaluates the best individual the adaptive genetic program offers it from its population. The attribute of the adaptive genetic program references the column in the string(s) upon which the adaptive genetic program is currently operating.

INDEX OF ENTITY	ENTITY	ATTRIBUTE RANGE	RANDOM NUMBER INTERVAL AND CUMULATIVE FREQUENCY DISTRIBUTION USED TO EFFECT GENETIC OPERATION ON CURRENT ADAPTIVE GENETIC PROGRAM
1	crossover	(1,14)	Random (1,14)
2	inversion	(1,14)	Random (1,14) length 2★ Random (1,5) or less.
3	mutation	(1,15)	Random (1,15) for entity attribute/2 for magnitude.
4 5 6 7 8 9 10 11 12 13 14	coefficients for utility polynomial for best individual produced by adaptive genetic program being operated on	(0,1)	all coefficients = 1.
15			

Fig. 4.6. Non-adaptive Genetic Program Directing the Evolution of Adaptive Genetic Programs as they Operate on Populations of Strings.

genetic programs in much the same way that the adaptive genetic programs direct the evolution of the strings of individuals (Fig. 4.3).

The actual mechanics of programming are straightforward. The description of the $(I + J)$th string is easily obtained by a DO loop which loads the attribute of the entity into the entity for ENTITY(1) through ENTITY(14). The program for the simulated cell then has the proper values for its variables so that it can make a simulated run and receive a utility rating. The indexes for the respective entities, together with the attribute associated with the entity are stored in the CHROM array with a value of the third dimension of the array equal to $I + J$. The program statement is

```
      DO 1, K = 1,14,1
1     ENTITY(CHROM(1,K,I + J) = CHROM(2,K,I + J)
```

Crossing over is very easy to program since all individuals which form crossover pairs have the same linkage map. Inversions do occur, but when an inversion is produced, it is used to generate a population which evolves as a group. The unequal probabilities of crossing over for different regions of the linkage map are realised by associating a cumulative frequency distribution with the crossover operator indexed in the genetic program (adaptive or non-adaptive). This probability is an attempt to simulate the inequalities in probability of crossing over for different regions of real chromosomes induced by the presence of inversion heterozygotes during real crossing over, since such heterozygotes are not simulated because of the complications introduced into the programming procedure for crossing over. Inversions are simulated, however, since they are a powerful permutation operator allowing the evolving populations to experiment with various linkage maps.

An example of crossing over will be programmed for the individuals with the highest and next highest utilities in the population of ADAPTIVE GENETIC PROGRAM (1). These strings will be ordered so that they occupy the first two positions in the population of ADAPTIVE GENETIC PROGRAM (1) *i.e.*, CHROM $(I,J,1)$ refers to the Ith row, and Jth column of the individual with highest utility in the population supervised by ADAPTIVE GENETIC PROGRAM (1). CHROM $(I,J,2)$ refers to the string with second highest utility in an analogous manner. To obtain the crossing-over parameter, a random number is generated in the range 1 to N, where N is stored in the location CHROM(41,2), *i.e.* N is the attribute of the 41st entry. The crossover will then occur at the right of the column in the CHROM vector designated by the random number if the random number is less than the number of columns in the CHROM vector; otherwise no crossover will take place. The larger the value of N, the less the probability of a crossover. To generate a probability curve other than uniform for crossing over, a cumulative probability distribution could be used, and one could pick the point on the chromosome whose cumulative distribution function is less than but closest to a random number which was generated between 0 and 1 (Mize and Cox, 1968). The

same Monte Carlo technique can be used to obtain probability distributions for mutational increments for any desired probability distribution.

To program a crossover between individuals CHROM (*I,J,*1) and CHROM (*I,J,*2), the following sequence of instructions can be used, where *I* and *J* are variable, and denote individual strings, the crossover takes place at position *X*, and the crossover products are loaded into strings CHROM (*I,J,*5) and CHROM (*I,J,*6).

```
      K = 1
      DO 1 I = 1,4,1
      DO 1 J = 1,X,1
      CHROM (I,J,K + 4) = CHROM (I,J,K)
1     CHROM (I,J,K + 5) = CHROM (I,J,K + 1)
      DO 2 I = 1,4,1
      DO 2 J = X,15,1
      CHROM (I,J,K + 5) = CHROM (I,J,K)
2     CHROM (I,J,K + 4) = CHROM (I,J,K + 1)
```

KEY

K is the base for denoting the individual.
I denotes the row of the CHROM array.
J denotes the column of the CHROM array.
the following six lines effect crossing over.

Since the best four strings are saved after a round of phenotypic adaptation and evaluation by the ADAPTIVE GENETIC PROGRAM (1), the recombinant will be loaded into CHROM (*I,J,*5), thus destroying the fifth individual in ranking with respect to utility. If both products of the crossover are saved, the second recombinant will be loaded into CHROM (*I,J,*6).

The column for crossover will be obtained by Monte Carlo techniques, using the random-number interval and cumulative frequency distribution located in the column of the adaptive genetic program which indexes the crossover entity. (1 happens to index crossing over, so the third and fourth rows of the column containing a 1 in its first row will contain, respectively, the random number interval and cumulative distribution used to generate the point of crossover point.) For a simplified example of Monte Carlo techniques let the random number be generated over (1,14). Let x be the column to the left of the crossover point. Let $F(x)$ be the cumulative distribution for the probability of a crossover occurring to the right of x. The crossover point $x = F^{-1}(u)$ (Mize and Cox, 1968, p. 74). The crossover does not occur if a random number is generated outside of the range of $F^{-1}(x)$, allowing mutation to increase or decrease the probability of crossing over for the whole chromosome by changing the length of the interval over which the random number is generated if it is wider than the range of $F^{-1}(x)$. If this is not useful, it can be discarded by the genetic program, and a random-number interval (1,14) used.

Not only inversions, but also chemical differences on different parts of real chromosomes alter biological crossover frequency. There are complex interactions between different parts along the length of chromosomes undergoing crossing over. The assumption of constant crossover frequency per unit string length is a close approximation to the relation between linkage and percent recombination for real chromosomes. The linkage map of the real chromosome approximates a linear function of percent recombination for map distances less than forty percent. Constant probability of crossover per unit length is therefore a reasonable first approximation for Holland's theoretical development. However, the probability distributions I use to simulate the action of inversion heterozygotes will also take care of much of the genetic control exercised by real cells on different rates of crossover for different regions of their chromosomes. Since the probability distribution used is under genetic selection, and the probability distributions stored in the computer may be mixed for Monte Carlo simulations (Mize and Cox, 1968, Chapter 6), this simple expedient realises many complex genetic functions, and therefore generating probability distributions for crossover actuation seems a practical utilisation of computer facilities in effecting simulated evolution. Experimental observations upon relationships between linkage, percent recombination and cytological observations are available in the literature (Strickberger, 1968).

Inversions are somewhat artificially simulated for programming simplicity. The best adaptive genetic program as judged by the non-adaptive genetic program, selects the sites of inversion using the random-number interval and frequency distribution in its column which indexes the inversion entity (column with a 2 in row 1). It then inverts this segment of the strings of its population for loading into the population of the third best adaptive genetic program. Similarly the second best adaptive genetic program introduces an inversion into its population for loading into the population of the worst adaptive genetic program. The inversions are not introduced into the populations of the two best genetic programs. However, the two worst adaptive genetic programs are replaced by genetic combinations of the two best genetic programs, so there is some possibility for good genetic procedures to evolve and interact with improved linkage maps effected by inversion. By appropriate choice of random-number interval and cumulative density, one can easily manipulate the probability of obtaining any particular number. This is particularly useful in directed mutation, where the increment in an attribute becomes easy to control, implicitly defining the recessives stored by a string as its high-probability mutants. The whole population of potential mutants changes when an attribute changes, partially simulating a change in dominance. This correspondence is so indirect, however, that I would consider it an experimental part of the program. Since the random-number interval and frequency distribution referenced are also subject to mutation and selection along with the rest of the string, the population may evolve an efficient simulation of natural dominance since dominance is useful.

The locus to undergo mutation is obtained by the genetic program, and the mutational increment is then obtained by using the random-number interval stored under the locus (locus = entity) undergoing mutation. Mutation in the adaptive genetic programs is analogously effected by the non-adaptive genetic program. The non-adaptive genetic program only mutates under direct manipulation by the programmer. An example of the kinds of values used in directed mutation follows. The entity to mutate is $k(5)$, which has a current value of 5. The mutational increment is set at $n \cdot 5$ by the information stored along with the index of the entity in the CHROM array. The value for n is obtained by calling on the random-number generator, and generating a number between -20 and $+20$ as directed by the density distribution specified along with the index of the entity. The mutational increment thus ranges from $-20 \cdot 5$ to $+20 \cdot 5$, in intervals of 5. The next value of $k(5)$ will be one of the numbers in the range $-20 \cdot 5 + 5$ to $+20 \cdot 5 + 5$, obtained by adding the value of the mutational increment ($-20 \cdot 5$ to $+20 \cdot 5$) to the current value for $k(5)$ which is 5. This procedure differs from natural mutation where most mutants are random alterations, and therefore useless, so that saving recessive alleles through protection by dominance in diploids becomes necessary. Directed mutation saves time and storage in a computer simulation since much unnecessary mutation is ruled out, recessive alleles do not have to be stored, and complex calculations for dominant alleles in the canonical realisation of the string are eliminated. The directed mutation procedure can be quite simple as outlined above for the entity $k(5)$.

My motivation for only one representation of any particular string in the program is that I would like to preserve maximum variability with minimal computation and storage, since variability is equal to the rate in change of fitness of the population by Fisher's Fundamental Theorem. The value of the utility of an individual, rather than a number of copies of that individual, determines the contribution of that individual to the next generation. There might also be a population of the best unused string from each population to be saved but not used except for recombination, as well as the directed mutation scheme, which permits nonrandom mutation to alleles likely to be useful, in order to permit realisation of dominance without lengthy computation. This may lead to lack of fixation of fit individuals, but will enable the evolving strings to try out more combinations. Since old strings are preserved each generation both as members of the next generation (40% of the old population is saved intact) and as potential population members through directed mutation which is rigged to produce useful alleles, the population is unlikely to 'forget' a good set of parameters once they are obtained.

The action of the operon in the simulated cell is particularly interesting, since the same repression technique can be applied to replication of portions of DNA by examining concentration of quantities in the program produced under the direction of one of the loci in the string. If the evolving string references simple subroutines, the genetic program may generate duplicates of the locus which needs

modification when the threshold for the quantity reaches a danger level (either too high or too low), or pick alternate subroutines from a list to add to the string, or call the operator for man machine interaction in production of a new subroutine to add to the string, the new subroutine being designed to supplement the offending subroutine already present which was not doing its job.

In a program as complex as the simulated cell, the duplication operator would be a signal for man machine interaction, with the man modifying the subroutine not predicting correctly, or adding a new subroutine to extend one already present. The predictors, however, are easy to define for the simulated cell, since each parameter is closely associated with a simulated activity. $P1(k)$ and $P3(k)$ would need modification when the catalytic activity of enzyme $EK(k)$ increased in spite of the fact that too much $PRDC(k)$ was already present. $R(k)$ should be examined if MRNA(k) increased when $PRDC(k)$ was already too high, or if MRNA(k) decreased when $PRDC(k)$ was too low, for either of these actions would indicate that $R(k)$ is not doing the job it is predicted that it will do.

In a program with simpler subroutines, like Cavicchio's pattern-recognition program, the signal for duplication of a locus might well enable the genetic program directing evolution to modify an old locus to produce the needed function.

A lumping operator on strings would be related to schemata (Holland). A gene could be merged with another gene by the lumping operator which would convert the references to two parameters to a reference to one parameter. The lumped genes are closely related to successful schemata, since their survival indicates that the genes which constitute them are successful in combination with each other. By allowing directed mutation within the lumped group of genes, one may reap the reward of hidden recessives becoming dominant without sacrificing the advantage of a coadapted set of genes remaining linked through evolution.

Since the parameters indexed by the chromosome arrays are not limited to biochemical rate constants, the realisation of Holland's reproductive scheme as a computer simulation may be used to do a genetic search of many different spaces, thereby realising heuristic programs. Examples are the kinds of subroutines useful in pattern recognition (Cavicchio, 1968) or production of English sentences using a generative grammar (Bono, 1968). Both of these tasks have been written in preliminary form as populations of computer programs which evolve over time as a function of how well they do the specific task assigned to them. Heuristic programming may have interesting applications in obtaining programs to accomplish many ill-defined algorithms for tasks with a well-defined goal and reward scheme.

5. CANCER IN RELATION TO THE COMPUTER SIMULATION OF A LIVING CELL

Curing cancer is an example of the type of extension of the simulation of a living cell which I would like to eventually make (Heinmetz, 1966). Modifications of the

computer simulation might help to screen for cancer curing environments. Cancer is caused by uncontrolled growth of cells in the body. Normally many human cells stop growing in adults, or grow slowly. A human liver cell, for example, is inhibited from growing and dividing by contact inhibition, that is by contact with other liver cells. In a cancerous cell, this inhibition is ineffective. The cancer cells grow without proper controls, crowding out other cells. Another type of control of normal cells is determined by the tissue in which they may grow. Normal liver cells will not grow at all in the lungs, while cancer cells derived from liver may grow in the lungs and crowd out lung cells. Invasiveness and uncontrolled growth of cancer cells makes them difficult to remove by surgery, and may kill the man afflicted. Understanding the types of changes in cellular control mechanisms giving rise to uncontrolled growth may help in curing cancer by suggesting rational approaches to the prevention of changes leading to cancer, and to plans of attack against cancer cells.

Two types of changes in cellular control mechanisms may lead to uncontrolled, cancer-like growth: (1) a mutation in the cell's genes may alter the cell's control genes; (2) virus genes may enter the cell's chrosome, and subsequently alter the cellular control systems (Davis *et al.*, 1968). Simulation of cancer caused by mutation and by hidden viruses may enable one to simulate the effect of various environments on normal and cancer cells, and thereby help to find chemical environments which are likely candidates to test as cancer cures. The rapidity with which one can simulate the effect of different environments may enable one to 'test' in far greater amount and detail than one could investigate in actual experiments where limitations of time, space and experimental organisms are strong constraints. The control of DNA replication in bacteria and in humans is intimately related to the cell membrane, suggesting that extension of the microbial model of DNA replication to human cells may prove feasible (Clark, 1968; Comings and Kakefuda, 1968).

ACKNOWLEDGEMENTS

Dr J. A. Jacquez and the Biomedical Data Processing group provided me with a milieu in which I could begin studies at The University of Michigan. Dr John H. Holland encouraged me to enter the strange new field of computer and communication sciences from genetics, and taught at a level of excellence which permitted me to remain a student for the requisite initiation period. Dr Bernard P. Zeigler, E. Stewart Bainbridge, and Daniel J. Cavicchio tutored me with patience and skill through many perilous passages. Ronald Brender, Thomas Schunior and John Foy gave willingly of their computing knowledge. Dr Robert B. Helling and Dr Prasanta Datta kept me abreast of key happenings in biochemistry and genetics. Dr Arthur W. Burks understood my academic problems at all times, and inspired me by his

leadership of the Logic of Computers Group and the Department of Computer and Communication Sciences at The University of Michigan. Dr H. H. Swain and Richard Laing catalysed my work with their editorial comments and creative writing. Mr Thomas Dawson administered the affairs of the Logic of Computers Group, and the Department of Computer and Communication Sciences with morale boosting zeal. Miss Linda Beattie typed and drew with skill and dedication. I have erred at points in spite of all of these accomplished and generous helpers because I am human.

REFERENCES

ATKINSON, D. E., Regulation of enzyme activity, *Annual Rev. Biochem.*, **35** (1966) pp. 85–123.

BONO, P. R., A heuristic program which produces generative grammars, Ann Arbor, Mich.: Project for Course in Simulation of Biological Systems, CCS 680, The University of Michigan, 1968.

CAVICCHIO, D. J., Jnr., A heuristic program which recognises patterns, Ann Arbor, Mich.: Project for Course in Simulation of Biological Systems, CCS 680, The University of Michigan, 1968.

COMINGS, D. E. and KAKEFUDA, T., Initiation of deoxyribonucleic acid replication at the nuclear membrane in human cells, *J. Mol. Biol.*, **33** (1968) pp. 225–30.

CLARK, J. D., Regulation of deoxyribonucleic acid replication and cell division in *Escherichia coli* B/r, *J. Bacteriol.*, **96** (1968) pp. 1214–24.

DAVIS, B. D., DULBECCO, R., EISEN, H. N., GINSBERG, H. S. and WOOD, W. B., *Microbiology*, Harper & Row, New York, 1968.

FISHER, R. A., *The Genetical Theory of Natural Selection*, Dover, New York, 1958.

GALE, D., A geometric duality theorem with economic applications, *Review of Economic Studies*, **32** (1967) pp. 19–24.

GARFINKEL, D., A simulation study of mammalian phosphofructokinase, *J. Biol. Chem.*, **241** (1966) pp. 286–94.

GRIFFITH, J. S., Mathematics of cellular control processes, *J. Theoret. Biol.*, **20** (1968) pp. 202–16.

HEINMETS, F., Analog computer analysis of a model-system for the induced enzyme synthesis, *J. Theoret. Biol.*, **6** (1964) pp. 60–75.

HEINMETS, F., *Analysis of Normal and Abnormal Cell Growth*, Plenum Press, New York, 1966.

HILDEBRAND, F. B., *Introduction to Numerical Analysis*, McGraw-Hill, New York, 1956.

HOLLAND, J. H., Hierarchical Descriptions, Universal Spaces and Adaptive Systems, University of Michigan Technical Report 08226-4-T, Ann Arbor, Michigan, 1968a.

HOLLAND, J. H., Theory of Adaptive Systems, Course in Department of Computer and Communication Sciences, The University of Michigan, 1968b.

HOLLAND, J. H., Hierarchical descriptions, universal spaces and adaptive systems, in *Collection of Papers on Cellular Automata* (ed. A. W. BURKS), Urbana: University of Illinois Press, 1969a.

HOLLAND, J. H., Adaptive plans optimal for payoff by environments, in *Proceedings of the Second Hawaii Conference on System Sciences*, 1969b.

KIMURA, M., Changes of mean fitness in random mating populations when epistasis and linkage are present, *Genetics*, **51** (1965) pp. 349–63.

KOCH, A. L., Metabolic control through reflexive enzyme action, *J. Theoret. Biol.*, **15** (1967) pp. 75–102.

LARK, K. G., Regulation of chromosome replication and segregation in bacteria, *Bacteriol. Rev.*, **30** (1966) pp. 3–32.

MIZE, J. H. and COX, J. G., *Essentials of Simulation*, Englewood Cliffs, N.J.: Prentice-Hall, 1968.

MURRAY, A. W. and ATKINSON, M. R., Adenosine 5′ phosphorothioate. A nucleotide analog that is a substrate, competitive inhibitor, or regulator of some enzymes that interact with adenosine 5′-phosphate, *Biochemistry*, **7** (1968) pp. 4023–9.

STAHL, W. R., A computer model of cellular self-reproduction, *J. Theoret. Biol.*, **14** (1967) pp. 187–205.

STRICKBERGER, M. W., *Genetics*, Macmillan, New York, 1968.

SUGITA, M. and FUKUDA, N., Functional analysis of chemical systems *in vivo* using a logical circuit equivalent, *J. Theoret. Biol.*, **5** (1963) pp. 412–25.
TSANEV, R. and SENDOV, B., A model of the regulatory mechanism of cellular multiplication, *J. Theoret. Biol.*, **12** (1966) pp. 327–41.
WALLACE, B., *Topics in Population Genetics*, W. W. Norton, New York, 1968.
WEINBERG, R., Analytic and logical equations in a computer simulation of cell metabolism and replication, *Sixth Annual Symposium on Biomathematics and Computer Science in the Life Sciences*, The University of Texas, pp. 102–3, 1968a.
WEINBERG, R., Computer simulation of a living cell, *Bacteriological Proceedings*, (1968b) G114.
WEINBERG, R., Computer simulation of self-reproduction by a living cell, *Genetics*, **60** (1968c) p. 235.
WEINBERG, R. and BERKUS, M., Computer simulation of evolving DNA, *Biometrics*, **25** (1969) p. 447.
YEISLEY, W. G. and POLLARD, E. C., An analog computer study of differential equations concerned with bacterial cell synthesis, *J. Theoret. Biol.*, **7** (1964) pp. 485–501.

APPENDIX

Variables in program: A = array
0 = floating point
1 = integer

A2	A	0 arrays used in solve function to obtain rate constants used
A3	A	0 in allosteric inhibition
AA0		0 amino acid concentration at time zero
AAP		0 ATP molecules used to make 1 amino acid molecule
AA0		0 amino acid concentration
ADJST		0 adjustment factor for concentrations from volume increase
ADP0		0 ADP concentration at time zero
ADP		0 ADP concentration
ATP0		0 ATP concentration at time zero
ATP		0 ATP concentration
ATPSB	A	0 array to store ATP concentrations in different environments
BROTH		1 equals 1 if cell growing in broth
CAA		1 equals 1 if cell growing in casamino acid
CHRM0		0 number of chromosomes at time zero
CNTRL		1 equals 1 if cell using metabolic controls to adjust growth rate
COUNT		0 number of growth cycles made
CRAZY		1 used as a logical variable
C(I)	A	0 enzyme rate constants
DAA		0 change in amino acid concentration
DADP		0 change in ADP concentration
DAPO2		0 change in ATP concentration from literature
DATP		0 change in ATP concentration calculated from rate constants in one time step
DDNA1		0 change in amount of chromosome 1 in one time increment
DDNA2		0 change in amount of chromosome 2 in one time increment
DDNA3		0 change in amount of chromosome 3 in one time increment
DDNA		0 change in total DNA in one time increment
DEK(1)		0 change in enzymes for nucleotide production in one time increment
DEK(2)		0 change in enzymes for amino acid production in one time increment
DEK(3)		0 change in enzymes for glycolysis production in one time increment
DEK(4)		0 change in enzymes for wall production in one time increment
DEK(5)		0 change in enzymes for ADP, ATP synthesis in one time increment
DEK(6)		0 change in enzymes for DNA synthesis in one time increment
DEK(7)		0 change in enzymes for protein production in one time increment

Appendix (Cont.)

DEK(8)		0 change in enzymes for MRNA synthesis in one time increment
DEK(9)		0 change in enzymes for ribosome synthesis in one time increment
DEK(10)		0 change in enzymes for TRNA production in one time increment
DIN		0 change in initiator concentration in one time increment
DNA0		0 DNA at time zero
DNA1		0 chromosome 1 'concentration', *i.e.*, amount/volume of cell
DNA1Z		0 chromosome 1 at zero time
DNA1T		0 total chromosome 1
DNA2		0 chromosome 2 'concentration'
DNA2T		0 total chromosome 2
DNA2Z		0 chromosome 2 at zero time
DNA3		0 chromosome 3 'concentration'
DNA3T		0 total chromosome 3
DNA3Z		0 chromosome 3 at zero time
DNAP		0 ATP used per DNA molecule synthesized
DNA		0 DNA
DNASB	A	0 array to save concentrations of DNA in different environments
DNUC		0 change in nucleotide concentration
DBLE		0 time for cell to go through one reproductive cycle
DPRTN		0 change in protein in one time increment
DRIB		0 change in ribosome in one time increment
DMRNA		0 change in MRNA in one time increment
DRNA		0 change in total RNA in one time increment
DRNK(i)	A	0 change in MRNA for enzyme EK(i) in one time increment. i ranges from 1 to 10.
DT		0 length of one time increment, = differential
DUM1		0 dummy variable in solve function
DUM2		0 dummy variable in solve function
DUM3		0 dummy variable in solve function
DVOL		0 change in cell volume in one time increment
DWALL		0 change in cell membrane and cell wall in one time increment
DPRDK	A	0 array of change in product concentration in one time increment
PRD	A	0 the stored array of the previous four product values, for predictor corrector
DPRD	A	0 array of the four previous D(product) values for the predictor corrector
PPRD	A	0 current array of the predictor values of products
CPRD	A	0 current array of corrector values of products
EK(1)		0 concentration of enzymes for nucleotide production
EKZ(1)		0 concentration of enzymes for nucleotide production at zero time
EK(2)		0 concentration of enzymes for amino acid production
EKZ(2)		0 concentration of enzymes for amino acid production at zero time

where 3 indicates glycolysis
4 indicates cell wall production
5 indicates ADP, ATP production
6 indicates DNA production
7 indicates protein production
8 indicates MRNA production
9 indicates ribosome production
10 indicates TRNA production

FACTR		0 factor by which chromosomes multiply in one reproductive cycle
GLUC0		0 glucose concentration at zero time
GLUC		0 glucose concentration
ID		1 integer variable in RPLACE routine
IN11		0 site for replication of chromosome 11, = 1 if it is present

Appendix (Cont.)

IN11Z		0 site for replication of chromosome 11 at zero time
IN1		0 site for replication of chromosome 1, = 1 if it is present
IN1Z		0 site for replication of chromosome 1 at zero time
IN21		0 site for replication of chromosome 21
IN21Z		0 site for replication of chromosome 21 at zero time
IN2		0 site for replication of chromosome 2
IN2Z		0 site for replication of chromosome 2 at zero time
IN31		0 site for replication of chromosome 31
IN31Z		0 site for replication of chromosome 31 at zero time
IN3		0 site for replication of chromosome 3
IN3Z		0 site for replication of chromosome 3 at zero time
IN		0 concentration of initiator in cytoplasm
II		1 an integer variable
INZ		0 initiator concentration at zero time
K(1)		0 preliminary rate constant for nucleotide production
K(2)		0 preliminary rate constant for amino acid production
K(3)		0 preliminary rate constant for glycolysis
K(4)		0 preliminary rate constant for cell wall production
K(5)		0 preliminary rate constant for ADP production
K(6)		0 preliminary rate constant for DNA production
K(7)		0 preliminary rate constant for protein production
K(8)		0 preliminary rate constant for MRNA production
K(9)		0 preliminary rate constant for ribosome production
K(10)		0 preliminary rate constant for TRNA production
K(14)		0 preliminary rate constant for volume increase as a function of wall
KDRNK		0 rate constant for MRNA decay
K8K(i)	A	0 rate constant for MRNA EK(i)
K8KZ(i)	A	0 rate constant for MRNA for EKZ(i)
KBB(i)	A	0 rate constant for allosterically inhibited enzyme EK(i) with two molecules of product attached to the enzyme
KB	A	0 array of rate constants of allosterically inhibited enzymes with one molecule of product attached to the enzyme
KIN		0 preliminary rate constant for initiator production
K8K(i)	A	0 rate constant for production of EK(i)
KK(i)	A	0 rate constant for uninhibited enzyme EK(i)
K(i)	A	0 array to store preliminary rate constants, used for each environment
LN2		0 natural logarithm of 2
L		1 integer variable for calling on solve function
M		1 integer variable for printing loop
MRNA0		0 MRNA concentration at time zero
MRNAP		0 ATP per MRNA molecule produced
MRNA		0 MRNA concentration
MULT		0 number of genes producing initiator
NO		0 number of cell in population (doubles when cell divides)
NUC0		0 molecules of nucleotide at zero time
NUCP		0 molecules of ATP to make one nucleotide
NUC		0 concentration of nucleotide
P1		0 rate constant
P1V	A	0 array of equilibrium rate constants for enzymes
P3		0 equilibrium rate constant for two molecule allosteric inhibition
P3V	A	0 array of equilibrium rate constants for two molecule allosteric inhibition
PRDC0	A	0 array equivalenced to products at zero time

Appendix (Cont.)

PRDCK	A	0 array equivalenced to products
PRDC	A	0 array for storing concentrations of products in different environments
PRDC(1)		0 NUC
PRDC(2)		0 AA
PRDC(3)		0 ATP
PRDC(4)		0 WALL
PRDC(5)		0 ADP
PRDC(6)		0 DNA
PRDC(7)		0 PRTN
PRDC(8)		0 MRNA
PRDC(9)		0 RIB
PRDC(10)		0 TRNA
PRDC(11)		0 GLUC
PRDC(14)		0 VOL
PRTN0		0 protein concentration at zero time
PRTNP		0 ATP molecules used per protein molecule formed
PRTN		0 protein concentration
RAA		0 ratio of amino acid concentration to a base level
RADP		0 ratio of ADP concentration to a base level
RATP		0 ratio of ATP concentration to a base level
RC	A	0 array of repression constants for MRNA repression
RDNA1		0 ratio of chromosome 1 concentration to a base level
RDNA2		0 ratio of chromosome 2 concentration to a base level
RDNA		0 ratio of DNA concentration to a base level
REK(i)	A	0 ratio of EK(i) concentration to a base level, $i = 1, \ldots, 10$
RIB0		0 ribosome concentration at time zero
RIBP		0 ATP used per ribosome made
RIB		0 ribosome concentration
RNA0		0 RNA concentration at time zero
RNA		0 RNA concentration
TRNA0		0 transfer RNA concentration at time zero
TRNAP		0 ATP per transfer RNA molecule made
TRNA		0 transfer RNA concentration
RNK(i)	A	0 concentration of MRNA for enzyme EK(i), $i = 1, \ldots, 10$
RNKZ(i)	A	0 concentration at zero time of MRNA for EKZ(i), $i = 1, \ldots, 10$
RNUC		0 ratio of nucleotide concentration to a base level
RON		1 used as a logical variable turning repression on
RPRTN		0 ratio of protein concentration to a base level
RRIB		0 ratio of ribosome concentration to a base level
RMRNA		0 ratio of MRNA concentration to a base level
RRNA		0 ratio of RNA concentration to a base level
RTRNA		0 ratio of TRNA concentration to a base level
RRNK(i)	A	0 ratio of RNK(i) concentration to a base level, $i = 1, \ldots, 10$
R	A	0 array for repression constants
RVOL		0 ratio of new volume to old volume at end of one time increment
RWALL		0 ratio of pool for wall to a base level in terms of concentration
SUM	A	0 array used in solve function
T		0 generation time in seconds
VOL0		0 volume of cell at time zero
VOLN		0 volume at end of one time increment
VOL		0 volume
WALL0		0 concentration of pool for wall production at time zero

Appendix (*Cont.*)

WALLP		0 ATP molecules used per molecule of cell wall produced
WALL		0 concentration of pool for wall production
X		0 variable used in repression routine
XK(i,j)	A	0 value of K(i) in environment (j)
XEK(i,j)	A	0 value of EK(k) in environment (j)
XK8(i,j)	A	0 value of K8K(i) in environment (j)