



12-2001

Using yeast to implement DNA-based algorithms

Colton Arlington Smith

Follow this and additional works at: https://trace.tennessee.edu/utk_gradthes

Recommended Citation

Smith, Colton Arlington, "Using yeast to implement DNA-based algorithms. " Master's Thesis, University of Tennessee, 2001.

https://trace.tennessee.edu/utk_gradthes/9746

This Thesis is brought to you for free and open access by the Graduate School at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Masters Theses by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a thesis written by Colton Arlington Smith entitled "Using yeast to implement DNA-based algorithms." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Michael W. Berry, Major Professor

We have read this thesis and recommend its acceptance:

Straight, Becker

Accepted for the Council:

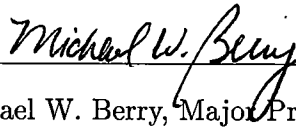
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

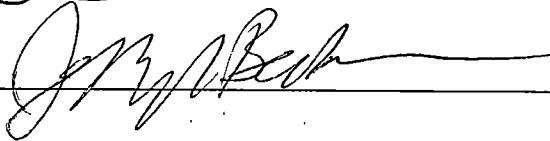

To the Graduate Council:

I am submitting herewith a thesis written by Colton Arlington Smith entitled "Using Yeast to Implement DNA-Based Algorithms". I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

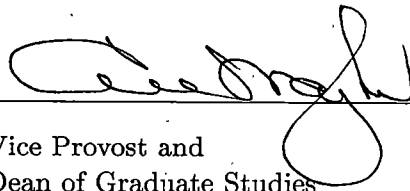


Dr. Michael W. Berry, Major Professor

We have read this thesis
and recommend its acceptance:



Accepted for the Council:



Vice Provost and
Dean of Graduate Studies

Using Yeast to Implement DNA-Based Algorithms

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Colton Arlington Smith

December 2001

Acknowledgements

I'd like to thank Drs. Berry and Becker for seeing this idea through. I'd also like to thank my committee for reviewing this project—Drs. Straight, Berry and Becker. I also appreciate the Innovative Technology Center (ITC) for allowing me to use their digital cameras and computers. Finally I'd like to acknowledge Fred Winston of Harvard Medical School for his generous gift of FY4, a critical strain used in this work.

Abstract

In 1993, Leonard Adleman showed that synthetic strands of deoxyribonucleic acid (DNA) can be made to compute in test tube reactions and thus invented the DNA computer. The DNA computer scales with remarkable efficiency when used to solve computationally hard problems. Here, we show that the DNA computer can be recast using the common yeast *Sacchomyces cerevisiae*. The yeast computer retains the efficiency of Adleman's DNA computer but is much easier and far less costly to implement.

Contents

1	Introduction	1
1.1	Background	1
1.2	Adleman's DNA computer	2
1.3	Motivation	4
2	A Novel Encoding for a Biological Computer	5
2.1	Introduction	5
2.2	The Encoding	6
2.3	Getting Ready To Compute	7
2.3.1	Choosing Genes	8
2.3.2	Physical Implementation of the Growth Matrix	9
3	Solving SAT Problems with the Yeast Computer	13
3.1	Introduction	13
3.2	Solid-Phase Computation	14
3.2.1	First Attempt	14

3.2.2	Second Attempt	15
3.3	Liquid-Phase Computation	17
3.3.1	Computing (x=1 or y=0) and (x=0 or y=1)	17
3.3.2	Computing (x=1 and y=0) or (x=0 and y=1)	20
3.4	Error-Rate	21
3.5	Scalability	22
4	Summary and Conclusions	24
	Bibliography	28
	Appendices	31
A	Figures and Tables: Chapter 2	32
B	Figures and Tables: Chapter 3	39
C	Glossary of Genetic Jargon and Abbreviations	62
D	Media Formulations	65
D.1	Condition A	65
D.1.1	agar plates	65
D.1.2	liquid	66
D.2	Condition B	66
D.2.1	agar plates	66
D.2.2	liquid	66

D.3	Condition C	67
D.3.1	agar plates	67
D.3.2	liquid	67
D.4	Condition D	68
D.4.1	agar plates	68
D.4.2	liquid	68
D.5	Condition I, alias "YEPD"	68
D.5.1	agar plates	68
D.5.2	liquid	68
D.6	Misc solutions	69
Vita		70

List of Tables

A.1	Yeast strains	32
A.2	Growth matrix	33
A.3	Growth matrix: an extension	35
B.1	Problem 1: growth spectra of condition B survivors	46
B.2	Problem 1: summary of condition B survivors	46
B.3	Problem 1: growth spectra of condition C survivors	48
B.4	Problem 1: summary of condition C survivors	48
B.5	Problem 1: growth spectra of condition A survivors	50
B.6	Problem 1: summary of condition A survivors	50
B.7	Problem 1: growth spectra of condition D survivors	52
B.8	Problem 1: summary of condition D survivors	52
B.9	Problem 2: growth spectra of condition B survivors	55
B.10	Problem 2: summary of condition B survivors	55
B.11	Problem 2: growth spectra of condition A survivors	57

B.12 Problem 2: summary of condition A survivors	57
B.13 Problem 2: growth spectra of condition C survivors	59
B.14 Problem 2: summary of condition C survivors	59
B.15 Problem 2: growth spectra of condition D survivors	61
B.16 Problem 2: summary of condition D survivors	61

List of Figures

A.1	Implementation of the growth matrix	34
A.2	Implementation of an extension of the growth matrix designed to select for singletons	36
A.3	Additional attempts to achieve condition I	37
A.4	Implementation of growth matrix in liquid medium	38
B.1	Computing using agar plates	40
B.2	Device used to make replica plates	41
B.3	Computing using replica-plating	42
B.4	Genotyping of replica-plating survivors	43
B.5	A liquid-phase algorithm.	44
B.6	Problem 1: genotyping of condition B survivors	45
B.7	Problem 1: genotyping of condition C survivors	47
B.8	Problem 1: genotyping of condition A survivors	49
B.9	Problem 1: genotyping of condition D survivors	51

B.10 Problem 2	53
B.11 Problem 2: genotyping of the condition B survivors	54
B.12 Problem 2: genotyping of the condition A survivors	56
B.13 Problem 2: genotyping of the condition C survivors	58
B.14 Problem 2: genotyping of the condition D survivors	60

Chapter 1

Introduction

1.1 Background

Computer scientists and engineers are now looking beyond semiconductor-based solid-state microelectronics and searching for new, more powerful ways of doing computations. Some scientists are trying to build logic circuitry out of individual organic molecules [RT00]. Others are taking things even further and trying to do computations using electrons as bits [BCLM98]. The latter approach has particular promise because electrons can apparently exist in two states at once and thus represent 1 and 0 simultaneously. This quantum effect can be exploited to achieve (theoretically at least) blazing fast computations.

But theoretical promise is one thing, concrete realization is another. Only the crudest sorts of molecular circuitry have been built so far and quantum computing is still just a theoretical possibility.

1.2 Adleman's DNA computer

In 1993, however, another alternative was conceived. It was the brain-child of Leonard Adleman, then a professor of computer science at the University of Southern California (USC). As he tells it [Adl98], he decided to take a break from computer science and spend some time in a biology laboratory learning the nuts and bolts of how to work with deoxyribonucleic acid—DNA. Ultimately, it was his intention to study the molecular biology of HIV, the virus that causes AIDS. Instead, late one night while nodding off to sleep, he had an epiphany—why not make a computer out of DNA? In other words, a DNA computer.

He was inspired by two facts [Adl98]. First, DNA chemistry follows a strict set of rules. The most famous of these is the Watson-Crick pairing rule which can be stated as follows: a piece of single-stranded DNA will form a stable double-helix when presented with its complement (see Appendix A for explanations of biological terms). From such simple rules, Adleman noted, computations can be done. Second, he was impressed with the fact that DNA segments can be artificially synthesized and manipulated almost at will in test tubes. With such capability, he reasoned, a DNA computer can actually be built.

What he did next was almost diametrically opposite to what biologists normally do (but very much in keeping with what theoretical computer scientists do) [Adl94]. Instead of looking at naturally occurring pieces of DNA and trying to divine their information content (e.g., the Human Genome Project), he made up his own information content and

logically projected it onto DNA. In other words, he realized that computational problems can be encoded as strands of synthetic DNA and that these strands can be operated upon in a logically meaningful way using the modern-day tools of molecular biology. To prove his point, he went into the lab and did a DNA-based computation. Specifically, he used combinatorial chemistry to synthesize a mixture of DNA strands that logically represented all possible paths through seven nodes and then used the modern tools of molecular biology to winnow through the mixture to extract the strands (i.e., the paths) that qualified as Hamiltonian. The entire experiment took him seven days—a long time considering that a modern electronic problem could solve the problem in a nanosecond. But it worked—he managed to extract the strands representing correct answers.

While the DNA computer proved to be disappointingly slow, Adleman realized that it could perform a huge number of instructions simultaneously. In other words, it had the potential of being massively parallel. His reasoning was as follows. DNA strands are molecules and hence very small. A test-tube can hold trillions of individual strands. Likewise, the enzymes used to operate on DNA are very small as well. Consequently, it's possible to operate on a test-tube's worth of DNA all at once by simply adding the appropriate enzymes. If each of these strands in this hypothetical test tube has a unique sequence and thus logically represents a different *guess* to a particular problem, then what we have in effect is a computer executing trillions of instructions simultaneously. In other words, a massively parallel computer unmatched by anything now currently available. And although the DNA computer may take a long time to complete an

operation, it compensates in effect by executing trillions of instructions per operation. This insight ignited an explosion of interest. Today, there are a dozen research groups, both in industry and academe, trying to speed up DNA operations to exploit this inherent parallelism exhibited by the DNA computer. The hope is that if the operations can be sped up sufficiently a nexus will be reached that will propel DNA computers into contention with today's most powerful supercomputers.

1.3 Motivation

Herein, a different tack was taken. Instead of trying to speed up DNA operations, an entirely new biological computer was developed, one which retains much of the parallelism of Adleman's DNA computer but which is easier and far less costly to implement. The encoding utilizes *Sacchromyces cerevisiae*, a common yeast strain. The yeast computer doesn't require costly DNA synthesizers or reagents and doesn't involve labor-intensive chemical extractions or analysis. Rather, the yeast computer relies on yeast to do the heavy-lifting required in the computations, not the human operator. As a consequence, the yeast computer serves as a prototypical example of how autonomous biological organisms can be used to compute and underscores some of the advantages inherent in this approach.

Chapter 2

A Novel Encoding for a Biological Computer

2.1 Introduction

Adleman used DNA as a medium for computation. DNA has a lot of features which recommend it. First of all, it's a very simple molecule. Whether very long or very short, it's always a linear array of four nucleotides (A, G, C, T) arranged one after the other in some sequence. These nucleotides can be thought of as an alphabet and the sequences they form can be thought of as strings conforming to the regular expression:

$(A + G + C + T)^*$.

Without the ability to operate on these sequences, however, there would be no such thing as a DNA computer. For the past thirty years, scientists have been developing

techniques to do just that. Today, DNA strands of any arbitrary sequence and length can be synthesized using automated machinery. Other machines are available which can read the nucleotide ordering of any arbitrary strand. To perform operations such as concatenation, deletion, selection, recombination or circularization, modern-day biologists have whole catalogues full of specialized enzymes they can buy to perform these tasks.

So why bother building a yeast computer when the DNA computer is already available? First of all, realize that the two aren't unrelated. Yeast have DNA. Yeast (or any living organism for that matter) are manifestations of their genetic content—i.e., their DNA. The physical manifestation of a genome's content is called a phenotype. If a yeast's phenotype can be mapped to its DNA, then its DNA can be logically operated upon through its phenotype. Such mappings are what the science of genetics is all about. And, as demonstrated in this thesis, it turns out that encoding problems and executing operations at the phenotypic level is much cheaper and far easier to accomplish (and to understand) than directly operating on the level of DNA.

Think of it this way. Encoding problems at the DNA level is like writing assembler code—laborious, tedious, slow and exhausting. Encoding and computing at the phenotypic level is like writing C code—much easier and certainly more fun.

2.2 The Encoding

What follows is our phenotypic encoding:

Let any gene in the yeast genome represent a variable. If that gene is *wild-type* (i.e., it does not have any mutations), then the variable it represents has a value of 1. On the other hand, if that gene harbors a mutation, then the variable it represents has a value of 0. Furthermore, for every variable (i.e., for every gene), suppose there is at our disposal a set of two growth conditions, one of which selects for yeast that have the wild-type allele of that gene while the second selects for yeast harboring the corresponding mutant allele. Now, to do a computation, encode the problem such that its answer is a bit string(s) of length n . Obtain a mixture of 2^n strains representing all possible bit patterns (i.e., all possible solutions). Algorithmically operate on this mixture with the different growth conditions to winnow out the strains that represent the correct solutions to the given problem.

2.3 Getting Ready To Compute

DNA computers were first used to solve problems in boolean logic, the basis of all modern-day computation. To illustrate the yeast computer, the same will be attempted here. Suppose we wanted to find out all values of x and y that render the following formula true: $(x=1 \text{ or } y=0)$ and $(x=0 \text{ or } y=1)$. This is an example of a class of boolean logic problems called *satisfiability* (or, SAT, for short) problems [HMU01]. SAT problems are notorious in computer science because they are considered *intractable*—i.e., unable to be solved in a linear number of steps.

This particular problem involves two variables. According to the yeast encoding, it

thus requires four different strains (note: $4 = 2^2$) that are variously mutant or wild-type in two different genes but are otherwise isogenic (i.e., the same). At this point, an appropriate set of two genes has to be chosen.

2.3.1 Choosing Genes

Perusal of the literature covering yeast genetics does not reveal a bounty of candidate genes. Considering the fact that yeast have about 6,200 genes, this seems odd. The problem is that geneticists haven't bothered to work out both negative and positive growth selections that specifically operate on each of these genes. Up till now, there hasn't been much need to. Despite this, there are some prospects. *URA3* and *LYS2* are two in particular.

S. cerevisiae requires the *URA3* gene to make uracil, an essential component of RNA. *URA3* mutants need exogenously provided uracil in order to grow. A toxic analog of uracil, 5-FOA, poisons yeast cells that contain the wild-type allele, leaving mutants in the *URA3* gene untouched [BLF84].

Likewise, *S. cerevisiae* requires the *LYS2* gene to make lysine, an essential amino acid. *LYS2* mutants need exogenously provided lysine in order to grow. A toxin called aminoadipate kills yeast cells that harbor the wild-type allele; the toxin leaves *LYS2* mutants unharmed [CS79].

Given these facts, *URA3* and *LYS2* should function appropriately in the context of the yeast encoding, at least theoretically. Let *LYS2* represent the variable x in our problem and let *URA3* represent y . Four strains are now required which are variously

mutant or wild-type in these two genes. Luckily, these are already available. Their names, genotypes and logical equivalences are shown in Table A.1.¹ Note that all figures and tables for Chapter 2 are included in Appendix A.

Given Table A.1, a matrix of growth conditions can be constructed that operate on these strains according to their genotypes. This matrix is shown in Table A.2.²

2.3.2 Physical Implementation of the Growth Matrix

At the time Table A.2 was conceived, it was not known whether the growth conditions it described would work in actual practice. Yeast geneticists had been working with *URA3* and *LYS2* for years but never in concert. Consequently, uncertainties abounded. For example, there was a very real fear that the presence of exogenous uracil would ameliorate the toxic effects of aminoadipate and thus subvert the purpose of condition A (namely, to kill off any strains wild-type for *LYS2*). Successful implementation of the growth matrix was predicated on the assumption that *LYS2* was indifferent to the status of *URA3* and vice versa. This was a very dangerous assumption indeed considering the extensive network of interdependencies between genes in biological systems.

¹Genotype designations in all capital letters indicate the wild-type allele. Designations in all small case letters indicate the mutant allele. All of the mutant alleles used in this study are deletions and hence do not revert to wild-type. Remember, a wild-type allele represents the value 1 while a mutant allele represents the value 0. Also remember, *URA3* represents the variable x while *LYS2* represents the variable y . Logical equivalence is expressed arbitrarily as an ordered pair $(0 + 1)(0 + 1)$ where the first number represents the value of x and the second represents the value of y .

²"+" indicates growth; "-" indicates no growth. Condition A contains supplemental uracil, aminoadipate and lysine. Condition B contains uracil. Condition C contains 5-FOA, uracil and lysine. Condition D contains lysine. Condition E contains uracil and lysine. Exact details of their composition can be found in Appendix B. FY4 and S288c are essentially identical except for the fact that FY4 is of the same mating type as the rest of the strains while S288c is of the opposite type. S288c was later abandoned for this reason.

Nonetheless, the risk was taken. First, agar plates embedded with each of the growth conditions were made (see Appendix B for details). Using a toothpick, small numbers of each strain were then struck out unto the plates. After three or four days of incubation at 30 degrees celsius, the plates were then observed for any consequent growth. The results are shown in Figure A.1.³

First, note that we observe robust growth only where expected: only two out four strains vigorously grow on any one plate. Which two is a function of what plate is used. Observe, however, this effect isn't absolute. For example, note how BY4724 exhibits some residual growth on condition B plates where none is expected. This effect is called *carry-over* and occurs because yeast are able to manage a few additional rounds of multiplication using internal sources of nutrients stored up when growing under permissive conditions. This observation is important because carry-over has the potential of introducing errors in yeast-based computations.

As an aside, another set of plates were made whose formulations were designed to allow for only *one* of the strains to grow up at a time. Such plates aren't really necessary for the yeast computer, because their results can be achieved by simply applying two of our original conditions in serial fashion (which two depends on which strain is ultimately being selected for). Nevertheless, these plates would have been useful as *short-cuts*. So, an attempt was made to extend the growth matrix with the additional conditions described in Table A.3.⁴

³In this figure as well as in many of the subsequent ones, condition E is often referred to as YEPD. This is in deference to yeast geneticists who normally see it referred to as such.

⁴Again, "+" indicates growth; "-" indicates no growth. Condition F contains no supplements.

As shown in Figure A.2, conditions F, G and H worked as intended but condition I failed. As suggested in [MD91], we tried systemically varying the concentration of 5-FOA against a constant amount of aminoadipate and lysine (see Figure A.3). As shown, at relatively low concentrations of 5-FOA, good selection against S288c and BY4700 was observed but unacceptably high levels of growth by BY4715 occurred. At slightly higher levels of 5-FOA (8-fold), BY4715 no longer grew but neither did BY4724. This suggested that, at best, there was an 8-fold range in which to work. This was too narrow and the effort was abandoned. While this effort didn't contribute anything to the work outlined in this thesis, it does underscore the vagaries one often encounters when trying to obtain discrete behavior from biological systems.

At this point, it was decided to test whether formulating the growth conditions in liquid medium might yield superior performance. The reasoning was that streaking yeast by toothpick is a very uncontrolled procedure (at least when done by a human). One can't really control the amount of inoculum, nor can the actual pressure and geometry of the stroke be reliably reproduced. In contrast, when grown in liquid, yeast are easier to work with. Their concentrations can be reliably determined using turbidity measurements and micropipettors can be used to transfer exactly measured-out aliquots from flask to flask. In view of this, we tested the matrix shown in Table A.2 in liquid. The results can be seen in Figure A.4. Each of the pictured flasks were inoculated several days before with 10^4 individuals of the indicated strain. As shown, no perceptible

Condition G contains 5-FOA and uracil. Condition H contains aminoadipate and lysine. Condition I contains 5-FOA, uracil, aminoadipate and lysine.

carry-over was observable with the naked eye. Nor was there any indication of the phenomenon using turbidity measurements (data not shown). Consequently, growth in this case seems truly *discrete*. Of course, this probably really isn't really the case. But the combination of using a small inoculum, a relatively large volume of liquid and allowing for a long period of biological amplification (i.e., growth) probably swamps out any chance of perceptible carry-over.

Chapter 3

Solving SAT Problems with the Yeast Computer

3.1 Introduction

In the last chapter, a problem from boolean algebra was chosen and then reduced to a corresponding problem in yeast genetics. Experiments were then done to make certain that the basic operations required by the encoding were actually realizable in the laboratory. Now, it's time to test whether it's possible to link these operations (both in series and in parallel) to achieve a meaningful problem-solving capability.

3.2 Solid-Phase Computation

3.2.1 First Attempt

Figure B.1 shows our first attempt to solve $(x = 1 \text{ or } y = 0)$ and $(x = 0 \text{ or } y = 1)$ using agar plates. (Note that all figures and tables referred to in Chapter 3 are found in Appendix B). We call this *solid-phase, yeast-based* computing. The cartoon shows the operations and a stylized representation of the results. The associated photographs show the actual results.

The algorithm goes like this. First, streak out each strain in one of the four quadrants of a single condition E plate (labeled YEPD in the figure). All strains should grow into macroscopic colonies and indeed they did (see corresponding photo). Now, take a dab of each of the strains and streak them unto condition B plates, again in four quadrants. Take another dab and streak them unto condition C plates. Wait for a couple of days and observe for growth. Those strains that grow (i.e., the *survivors*) should represent solutions that satisfy the first clause, namely $(x = 1 \text{ or } y = 0)$. Note that, in practice, the strains denoting 11, 10 and 00 grew whereas the strain representing 01 did not.

Now, take a dab from each of the survivors and plate them on condition A plates. Take another dab and do the same on condition D plates. Observe for growth. Whichever strains emerge represent solutions that satisfy both clauses. Note that at this point in the algorithm the strains representative of 00 and 11 grew again whereas the strain equivalent to 10 failed. By inspection, 00 and 11 are both correct solutions to the original problem.

While this algorithm was successful, it seemed singularly unsatisfying. It was too labor-intensive in the sense that the human operator seemed to be called upon to make too many subjective judgements along the way. For example, how does the operator score growth versus carry-over? Herein, it was done by judging robust-looking growth as being bona fide and anemic-looking growth as being merely carry-over. Correct answers resulted, but this approach was abandoned nonetheless.

3.2.2 Second Attempt

In an attempt to salvage the use of plates in yeast computing, we tried a technique called *replica-plating*. This technique imprints the yeast from one agar plate unto another using a velvet surface mounted on a heavy cylinder (see Figure B.2 for a picture of the device). The procedure involves lightly pressing a virgin piece of velvet unto the first plate to make the imprint and then pressing the imprint on the second to accomplish the transfer, thereby making a replica. This procedure is not unlike old-fashioned printing but instead of ink, yeast is used instead.

The use of replica-plating fundamentally changes the algorithm to compute ($x = 1$ or $y = 0$) and ($x = 0$ or $y = 1$). Instead of toothpicking each of the strains individually, they can now be manipulated in parallel. See Figure B.3 for the algorithm and the results. In the figure, the directionality of the arrows indicate which plates served as templates and which were born of the procedure. The letters indicate the growth conditions embedded in the plate and the numbers indicate colonies selected for genotypic analysis. The strategy can be summarized as follows. First, make a mixture of the four strains in

liquid in a test-tube (ensure all four strains are represented equally) and spread (or *plate*, to use the parlance of geneticists) an aliquot onto a YEPD plate so that about 100 colonies develop upon incubation. The result of this step is shown as the top-most photograph in the figure. Now, replica-plate this plate unto B, YEPD and C plates and allow colonies to grow. The result of this is shown as the three photographs occupying the second row of the figure. Finally, replica-plate the B plate unto A, B and YEPD plates. Do the same with the C plate. The results are shown. Now, select colonies for genotypic analysis. In the figure, the numbered colonies were selected. Their analysis is shown in Figure B.4. Colonies 1 and 2 proved to be FY4 (i.e., 11). Colonies 3 and 4 proved to be BY4724 (i.e., 00). These are correct solutions to the original problem.

Despite this success, the algorithm still falls short. First of all, carry-over is still a problem. In addition, the procedure still doesn't scale well. One reason is evident from inspection of the plates shown in Figure B.3. As the replica-plateing is reiterated, the colonies get progressively more squashed until they finally lose all definition. Another more subtle reason is that the number of plates required scales according to

$$\sum_{i=1}^m k^i,$$

where k is the number of variables and m is the number of clauses. In other words, the number of plates scales exponentially.

3.3 Liquid-Phase Computation

At this point, all efforts to use plates were abandoned. The decision was made to use liquid cultures in their stead. This change necessitated another alteration in the algorithm.

3.3.1 Computing ($x=1$ or $y=0$) and ($x=0$ or $y=1$)

Figure B.5 shows the new strategy. A key feature here is the pooling operation (see central portion of figure). This pooling effect could be achieved using solid agar plates but it's trivial when liquid is used: survivors of the first two selections are literally poured together in the same test-tube and aliquots from this mixture are then used to inoculate the next set of flasks.

Some key details are missing from the figure. The flasks used were 125 milliliter erlenmeyers containing 50 milliliters of fluid. Liquid media were made fresh, except for the YEPD. The inoculated cultures were incubated at 30 degrees celsius on a rotating (30 rpm) platform.

To initiate the algorithm, a mixture of the four strains was required. To generate this, the following was done: overnight cultures of the four strains grown in YEPD were prepared, the concentrations of yeast contained therein were determined using turbidity measurements [BDS00], and then a mixture in sterile water consisting of $6 \times 10^4/ml$ individuals from each strain was formulated. Depending on how well the strains grew, this represented a 60- to 120-fold dilution of each of the strains. 50 microliters of the

above mixture were used to inoculate the first set of flasks. This provided each flask with ten thousand individuals of each strain. Note that 50 microliters into 50 milliliters represents a dilution effect of 1 to 1,000. While seemingly inconsequential, these details are important because initial conditions presumably dictate whether or not effects like carry-over or phenotypic leak come into play.

After one day of incubation, an obvious degree of turbidity developed in the inoculated flasks. Upon measurement, it was found that there were $1 \times 10^7/ml$ individuals in the condition B flask whereas there were $7 \times 10^6/ml$ in the condition C flask. These numbers were nearly 10-fold less than those achieved in YEPD medium. But this was to be expected since YEPD is a richer medium. Anyhow, a mixture was prepared in sterile water that was $6 \times 10^4/ml$ with respect to each set of survivors. 50 microliters of this mixture were then used to inoculate the second set of flasks.

Two days were required for the condition D flask to develop a high degree of turbidity. Upon measurement, it was found to contain $1 \times 10^7/ml$. An additional day was required by the condition A flask. It was found to contain only $2.6 \times 10^6/ml$ at the time of harvest. At this point (four days after initiating the algorithm), the computation was complete.

At each step during this computation, survivors were plated on YEPD plates, colonies were picked and their genotypes determined by subsequently streaking them out on the four conditions. This was done to see just how accurately the computation was proceeding. Since the problem consisted of four separate steps or selections, this necessitated four different analyses.

Sixteen survivors of the condition B selection were analyzed. Their raw behavior is shown in Figure B.6.¹ Their growth spectra are interpreted in Table B.1. A summary is assembled in Table B.2. As shown, one survivor (colony #13) simply failed to grow during its genotyping and was discounted. Another survivor (colony #2) grew on four of the five conditions. This is a physical impossibility for a single genotype and it was concluded that colony #2 was, in fact, a mixture of two different genotypes, namely 10 and 11. Due to this ambiguity, this colony was discounted. The remaining colonies did not yield ambiguous results: they all proved to be either 11 or 10.

Sixteen survivors of the condition C selection were analyzed. Their raw behavior is shown in Figure B.7. Their growth characteristics are detailed in Table B.3. A summary is assembled in Table B.4. These colonies proved to be either 10 or 00.

Ten survivors of the condition A selection were analyzed. Their raw behavior is shown in Figure B.8. Their growth characteristics are detailed in Table B.5. A summary is assembled in Table B.6. These colonies proved to be all 00.

Ten survivors of the condition D selection were analyzed. Their raw behavior is shown in Figure B.9. Their growth characteristics are detailed in Table B.7. A summary is assembled in Table B.8. These colonies all proved to be 11.

In summary, the calculation depicted in Figure B.5 yielded two genotypes, namely 00 and 11. By inspection, these are the only solutions to our original problem. Consequently, it can be concluded that the yeast computer actually works in practice, at

¹In this and subsequent pictures like it, the plates are all oriented similarly so that the same survivor appears at the same location on all the plates.

least in this problem instance.

3.3.2 Computing ($x=1$ and $y=0$) or ($x=0$ and $y=1$)

Just because the yeast computer can solve ($x = 1$ or $y = 0$) and ($x = 0$ or $y = 1$) doesn't mean it can solve any such problem. For instance, in the latter problem, the order of selections were: B or C, then A or D. Would accuracy suffer if we were given a problem to solve that necessitated a different ordering of selections? To see, the following was computed: ($x = 1$ and $y = 0$) or ($x = 0$ and $y = 1$).

The algorithm used to solve this problem is shown in Figure B.10. This entailed a fundamentally different flow of growth selections compared to the last, but the strains and the media were all the same. As in the previous computation, genotypic analyses was done to monitor the quality of each of the selections.

Sixteen survivors of the condition B selection were analyzed. Their raw behavior is shown in Figure B.11. Their growth characteristics are detailed in Table B.9. A summary is assembled in Table B.10. These colonies all proved to be either 11 or 10.

Sixteen survivors of the condition A were analyzed. Their raw behavior is shown in Figure B.12. Their growth characteristics are detailed in Table B.11. A summary is assembled in Table B.12. These colonies all proved to be either 01 or 00.

Ten survivors of the condition C selection were analyzed. Their raw behavior is shown in Figure B.13. Their growth characteristics are detailed in Table B.13. A summary is assembled in Table B.14. These colonies all proved to be 10.

Ten survivors of the condition D selection were analyzed. Their raw behavior is

shown in Figure B.14. Their growth characteristics are detailed in Table B.15. A summary is assembled in Table B.16. These colonies all tested 01.

In summary, the computation depicted in Figure B.10 yielded survivors of only two genotypes, namely 10 and 01. By inspection, these are the only solutions to our problem and the dual to our last solution set. Consequently, we can conclude that the yeast computer is operationally versatile and can handle any ordering of selections.

3.4 Error-Rate

How error-prone is the yeast computer? If we define errors as being colonies in the final survivor set which represent *incorrect* answers, then it can be concluded from the data just outlined above that the yeast computer is no more than 10% error-ridden. This is quite good considering that, at best, DNA-based computation exhibits 90% fidelity [FLL00].

The yeast computer's error rate is probably far lower than 10%. This was suggested by the following experiment. 200 survivors of condition D from the first computation were plated out onto condition C, F, G and H plates which select for {00, 10}, {11}, {10}, and {01}, respectively. Colonies developed on the F plates but on none of the others. This indicated an error-rate of no more than 0.5%. The same was done with 200 survivors of condition A from the first computation. This time, colonies developed on the C plates but on none of the others, indicating a similar rate of error. Further analysis will probably lower this estimation even more.

3.5 Scalability

Can the yeast computer solve any SAT problem regardless of its size and form? The short answer is yes. Consider the following. Any boolean expression, no matter how bizarre, can be converted to a conjunctive normal form (CNF) or to a disjunctive normal form (DNF) [HMU01]. The first problem solved in this thesis was a two-variable boolean expression of the CNF variety. The second was a two-variable boolean expression of the DNF variety. If the algorithms presented in Figures B.5 and B.10 can be generalized to encompass any number of variables and clauses, then this would suggest that the yeast computer can handle any sort of SAT problem, provided the problem is preprocessed into a normal form. This generalization is achievable as evidenced by the pseudocode shown below:

```
preprocess SAT problem into either conjunctive normal form (CNF) or
disjunctive normal (DNF) form;
if (CNF){
    $pool = (all possible answers);
    $clause_result = NULL;
    $var_result = NULL;
    for each clause{
        for each variable{
            if ("variable = 1"){
                $var_result = select for wild-type allele from $pool;}
```

```

else{

    $var_result = select for mutant allele from $pool;}

    $clause_result = $clause_result + $var_result;

    $var_result = NULL;}

$pool = $clause_result;

$clause_result = NULL;}

print "answer is $pool";}

if (DNF){

    $initial = (all possible answers);

    $pool = NULL;

    $var_result = NULL;

    for each clause{

        for each variable{

            if ("variable = 1"){

                $var_result = select for wild-type allele from $initial;}

            else{

                $var_result = select for mutant allele from $initial;}

            $initial = $var_result;}

        $pool = $pool + $initial;

        $initial = (all possible answers);}

    print "answer is $pool";}

```

Chapter 4

Summary and Conclusions

In its present incarnation, the yeast computer has several good features:

- It's very cheap. Computing with strands of DNA requires a DNA synthesizer, expensive enzymes and a lot of expertise. Computing with yeast requires none of these. It is very accessible.
- Its encoding is easily understood. The yeast computer ultimately operates on the level of DNA. However, it does so through the interface of a living organism. The human operator doesn't need to know anything about the complexities of DNA chemistry. All he needs to know is under what conditions yeast strains live or die.
- It requires little physical intervention from the human operator, especially as compared to DNA-based computing.
- It doesn't require a man-made power source. The yeast computer is powered by

yeast cell metabolism.

- It isn't error-prone. This was explained earlier.
- It has the potential for massive parallelism. Consider the following: a 500 milliliter erlenmeyer culture of yeast can grow up to contain 1×10^{12} cells. All of these cells can be operated on at the same time by simply changing the medium. This is tantamount to executing 1×10^{12} instructions in parallel. Of course, a yeast cell is bigger than a piece of DNA with the same information content. Nonetheless, the yeast computer's potential for parallel processing is impressive.

The yeast computer, however, has several obstacles to overcome:

- It's slow. In terms of actual time, the yeast computer creeps along at a snail's pace: it takes 4 to 5 days to perform the example computations described in this thesis. Much of this time was due to the fact we waited for the survivors to replicate themselves enough to turn the media turbid. This probably wasn't necessary. Using smaller volumes and smaller inocula, days could have been shaven off the required time. Further time reductions could be achieved (maybe) by imprinting very small numbers of yeast cells onto a *biochip* and electronically reading their responses to changes in the growth media (see for example [SSP+00]). Even this would require seconds per step, however. This is not to say that computing with living organisms is useless. Even discounting the potential for parallelism, this is far from true. Biological computers exemplified by the yeast computer can

process inputs that a normal electronic computer simply can't deal with. The latter requires binary bits of 0's and 1's, while the former can respond to chemicals, temperature, pressure, whatever and can do so in such a way as to yield information or computation, depending on the encoding. Viewed in this context, consider the present thesis. Another way to interpret the work herein is not to say a yeast computer was built that can do boolean algebra, but rather to say a very specialized computer was engineered that can discern and report on the presence or absence of certain chemicals in liquid media, namely lysine, uracil, 5-FOA and amino adipate. Before dismissing this as feeble, consider the possibilities if the yeast computer were encoded to discern important environmental pollutants instead—chemicals like PCB's or naphthalene. Such a computer could yield important information at superfund clean-up sites. Consider the possibilities if a biological computer were encoded to discern glucose-levels in the human body. One could imagine tuning such a computer to pump out just the right dose of insulin in response. The possibilities go on and on.

- The first step of any yeast-based algorithm requires a mixture of yeast strains representing all possible bit patterns. In this thesis, this was achieved by simply obtaining the strains and then mixing them together. This is not very satisfying and quickly becomes impractical as problems get bigger. One would prefer to start with a single strain and then have the computation evolve from this single source. How could this be achieved? One way would be through the use of a strain that has

an error-prone DNA polymerase (see for example [GT00]). Such strains exhibit a *mutator* phenotype. This means that every time they replicate mutations are randomly peppered through their genomes at a rate 100 to 1,000 times greater than under wild-type conditions. This would have the effect of generating an initial set of all possible bit patterns. Of course, getting this strategy to work would not be trivial. How many growth generations would be required to generate an initial set? This would depend on the size of the problem and the rate of mutation. Would the mutator phenotype make the genotypes of the final survivor set unreliable? If so, could some sort of statistical analysis be done to compensate? Only experience will tell.

- As mentioned earlier, there aren't many genes for which there are both positive and negative growth selections. This puts a severe limit on the number of variables the yeast computer can represent at the present time. As knowledge of yeast cell metabolism grows, one can only hope these conditions will be found for an increasing number of genes.

Bibliography

Bibliography

- [Adl94] L.M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.
- [Adl98] L.M. Adleman. Computing with DNA. *Scientific American*, 279:54–61, 1998.
- [BCLM98] G. Brassard, I. Chuang, S. Lloyd, and C. Monore. Quantum computing. *Proceedings of the National Academic of Science*, 95:11032–11033, 1998.
- [BDS00] D. Burke, D. Dawson, and T. Stearns. *Methods in Yeast Genetics*. Cold Spring Harbor Press, Cold Spring Harbor, NY, 2000.
- [BLF84] J.D. Boeke, F. LaCroute, and G.R. Fink. A positive selection for mutants lacking orotidine-5-phosphate decarboxylase activity in yeast: 5-fluoroorotic acid resistance. *Mol Gen Genet*, 197:345–6, 1984.
- [CS79] B.B. Chattoo and F. Sherman. Selection of *lys2* mutants of the yeast *Saccharomyces Cerevisiae* by the utilization of alpha-aminoadipate. *Genetics*, 93:51–65, 1979.

- [FLL00] D. Faulhammer, R.J. Lipton, and L.F. Landweber. Fidelity of enzymatic ligation for DNA computing. *Journal of Computational Biology*, 7:839–848, 2000.
- [GT00] M.F. Goodman and B. Tippin. Sloppier copier DNA polymerases. *Curr Opin Genet Dev*, 10:162–168, 2000.
- [HMU01] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, Boston, MA, 2001.
- [MD91] J.H. McCusker and R.W. Davis. The use of proline as a nitrogen source causes hypersensitivity to, and allows more economical use of 5-FOA in *Saccharomyces cerevisiae*. *Yeast*, 7:607–8, 1991.
- [RT00] M.A. Reed and J.M. Tour. Computing with molecules. *Scientific American*, 282:86–93, 2000.
- [SSP⁺00] M.L. Simpson, G.S. Sayler, G. Patterson, D.E. Nivens, E.K. Bolton, J.M. Rochelle, J.C. Arnott, B.M. Applegate, S. Ripp, and M.A. Guillhorn. An integrated CMOS microluminometer for low-level luminescence sensing in the bioluminescent bioreporter integrated circuit. *Sensors and Actuators*, 72:134–140, 2000.

Appendices

Appendix A

Figures and Tables: Chapter 2

Table A.1: Yeast strains

name	genotype	logical equivalence
FY4 (or S288c)	<i>LYS2 URA3</i>	11
BY4700	<i>LYS2 ura3</i>	10
BY4715	<i>lys2 URA3</i>	01
BY4724	<i>lys2 ura3</i>	00

Table A.2: Growth matrix

name	condition A	condition B	condition C	condition D	condition E
FY4	-	+	-	+	+
BY4700	-	+	+	-	+
BY4715	+	-	-	+	+
BY4724	+	-	+	-	+

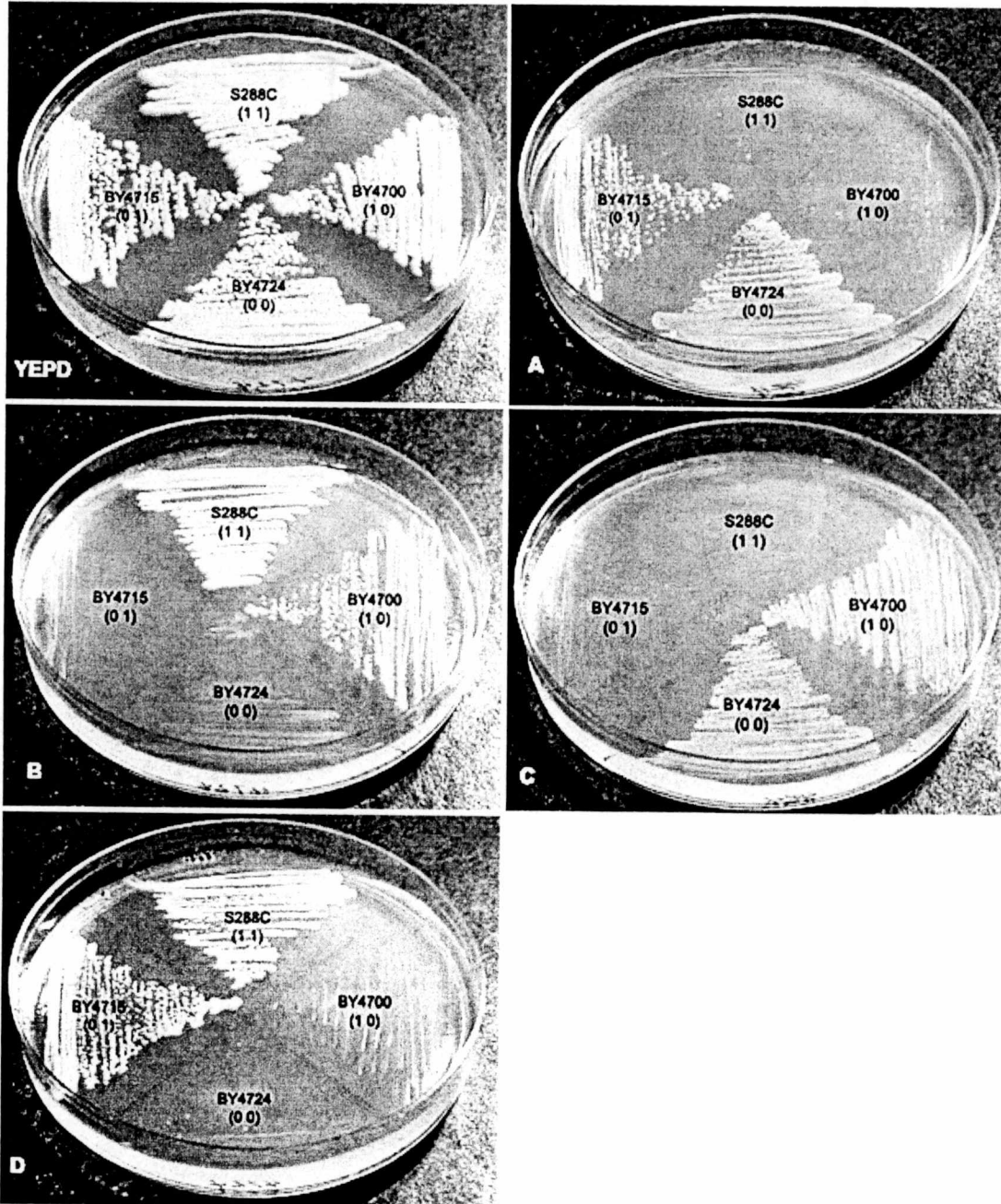


Figure A.1: Implementation of the growth matrix

Table A.3: Growth matrix: an extension

name	condition F	condition G	condition H	condition I
FY4 (S288c)	+	-	-	-
BY4700	-	+	-	-
BY4715	-	-	+	-
BY4724	-	-	-	+

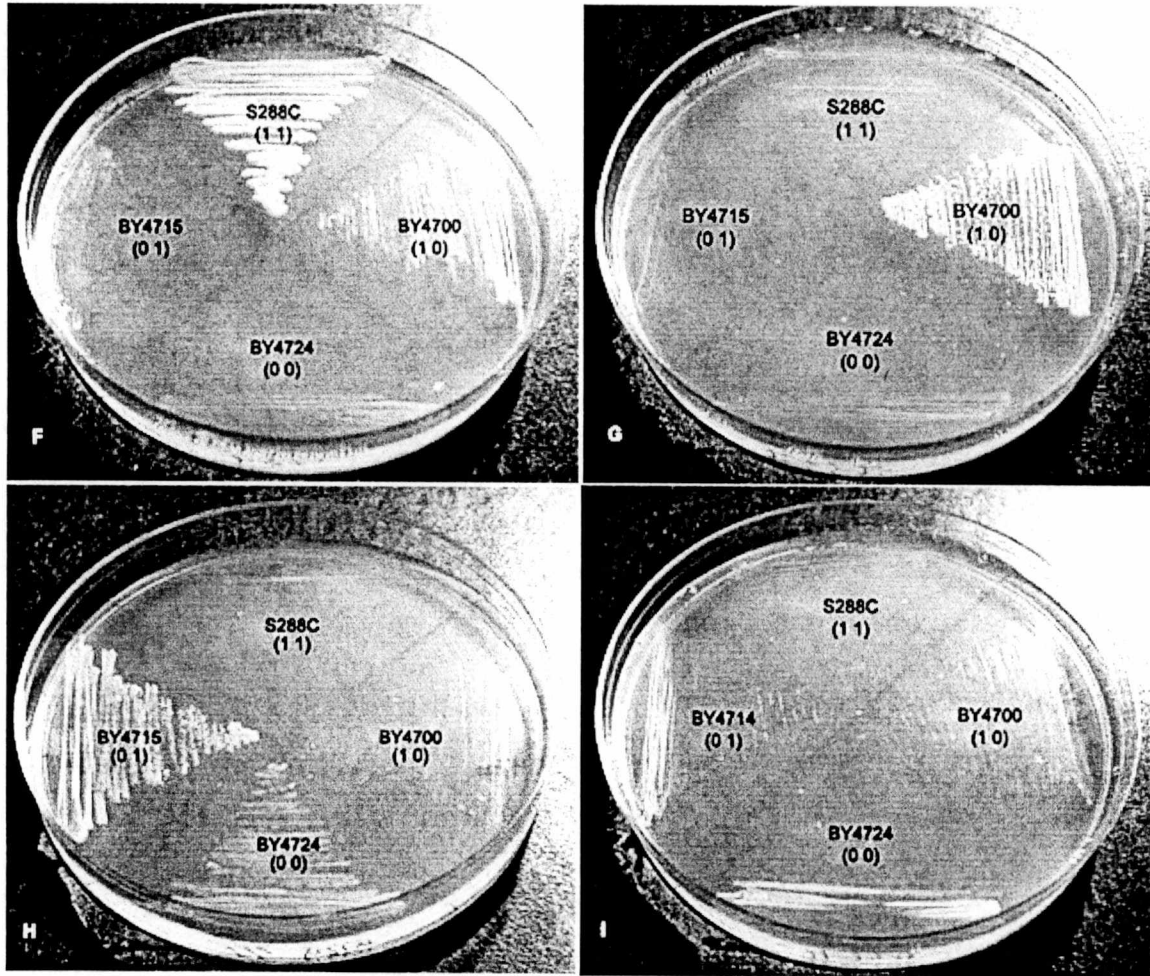


Figure A.2: Implementation of an extension of the growth matrix designed to select for singletons

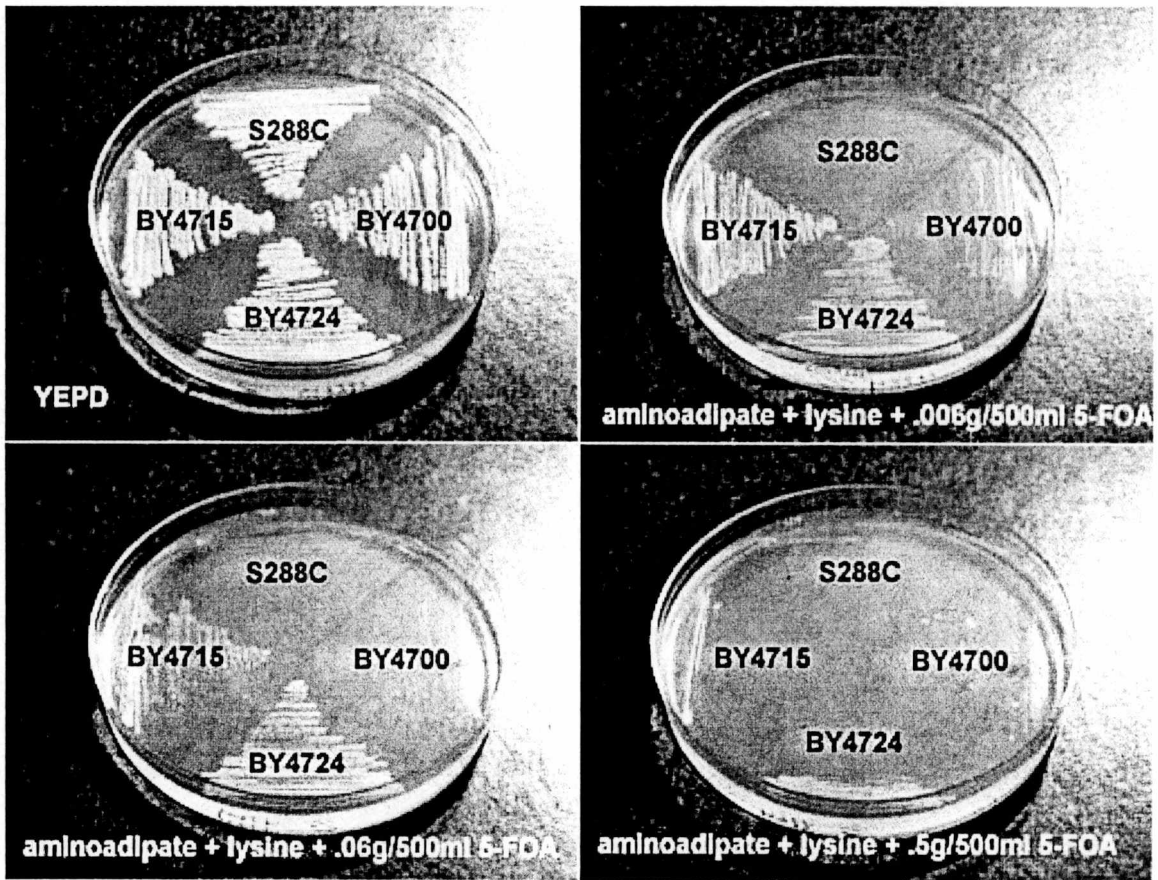


Figure A.3: Additional attempts to achieve condition I

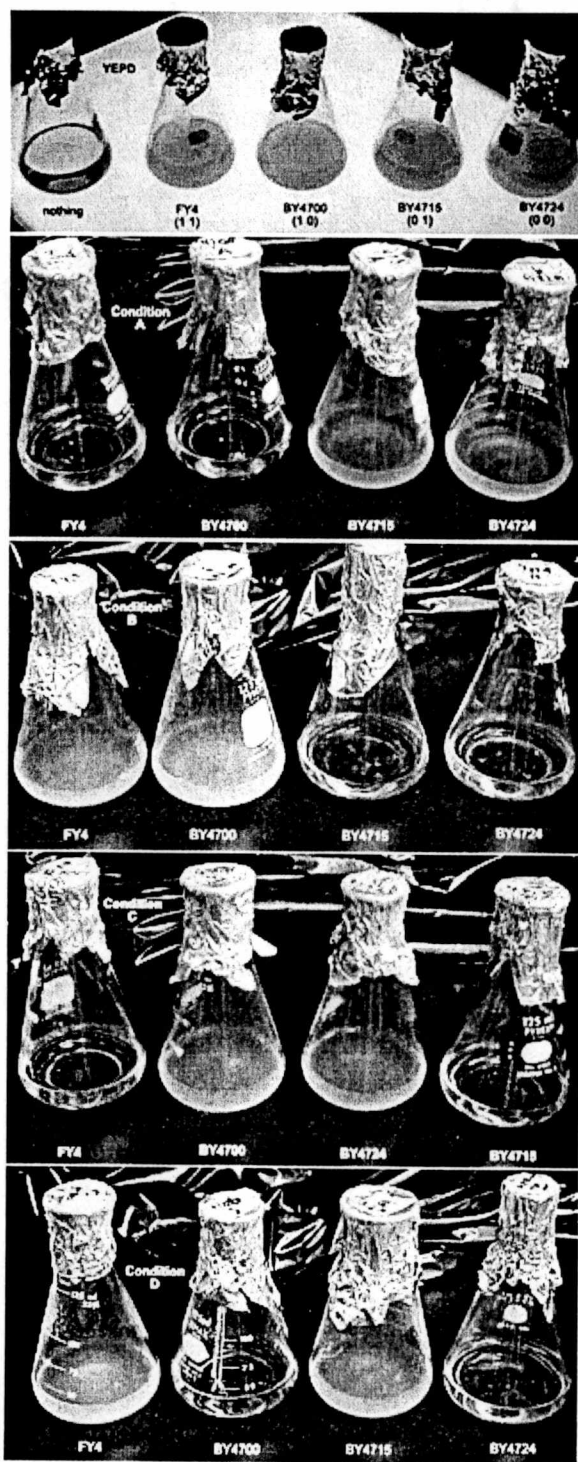


Figure A.4: Implementation of growth matrix in liquid medium

Appendix B

Figures and Tables: Chapter 3

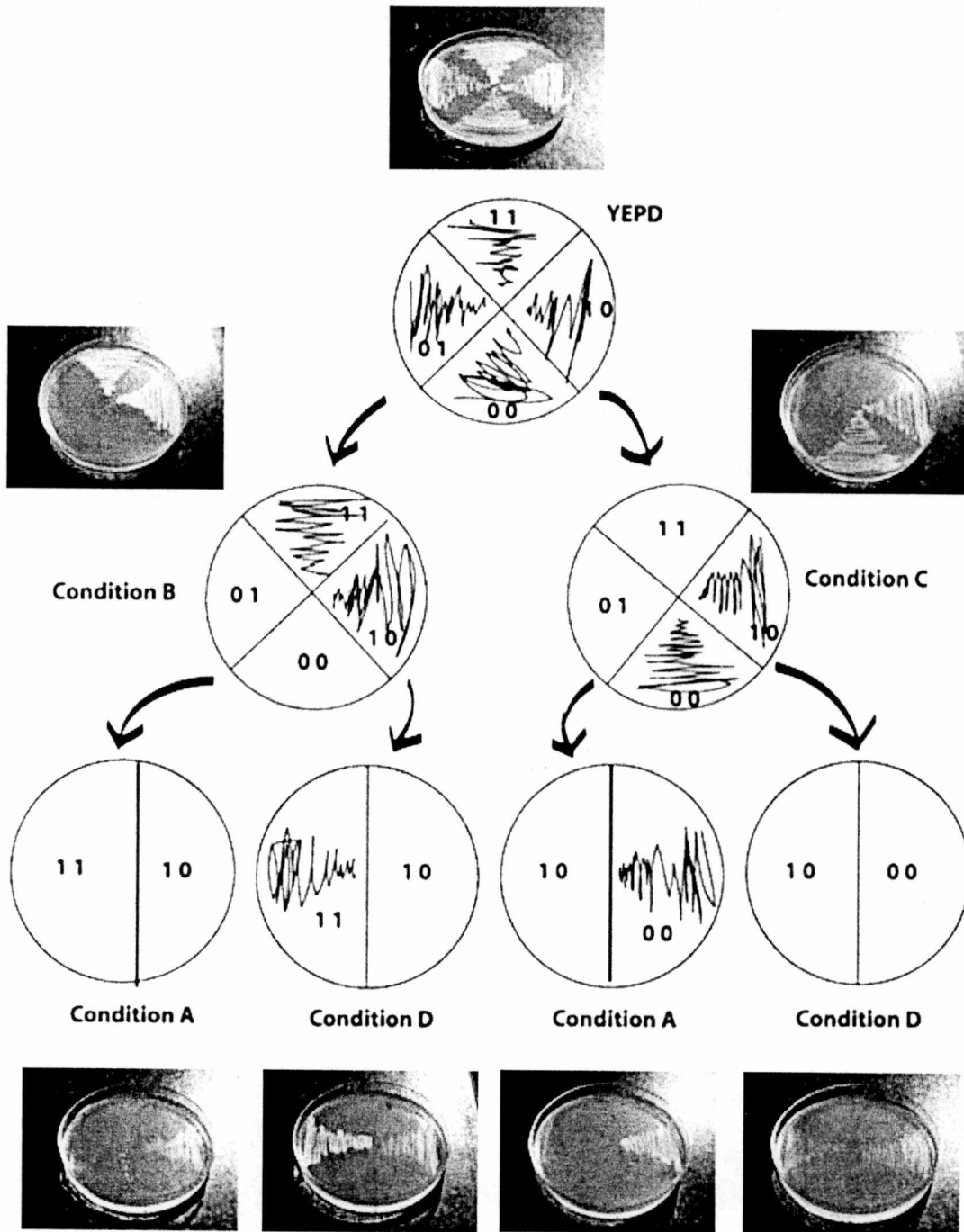


Figure B.1: Computing using agar plates

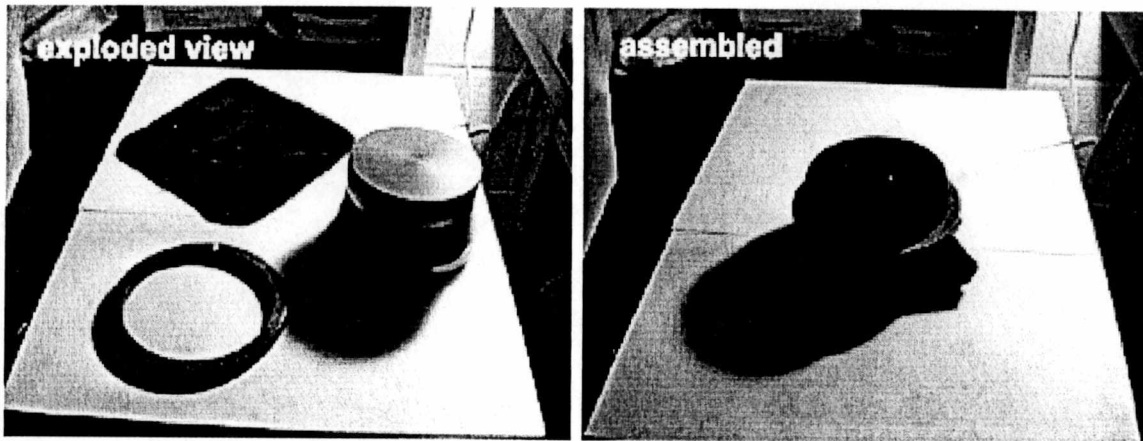


Figure B.2: Device used to make replica plates

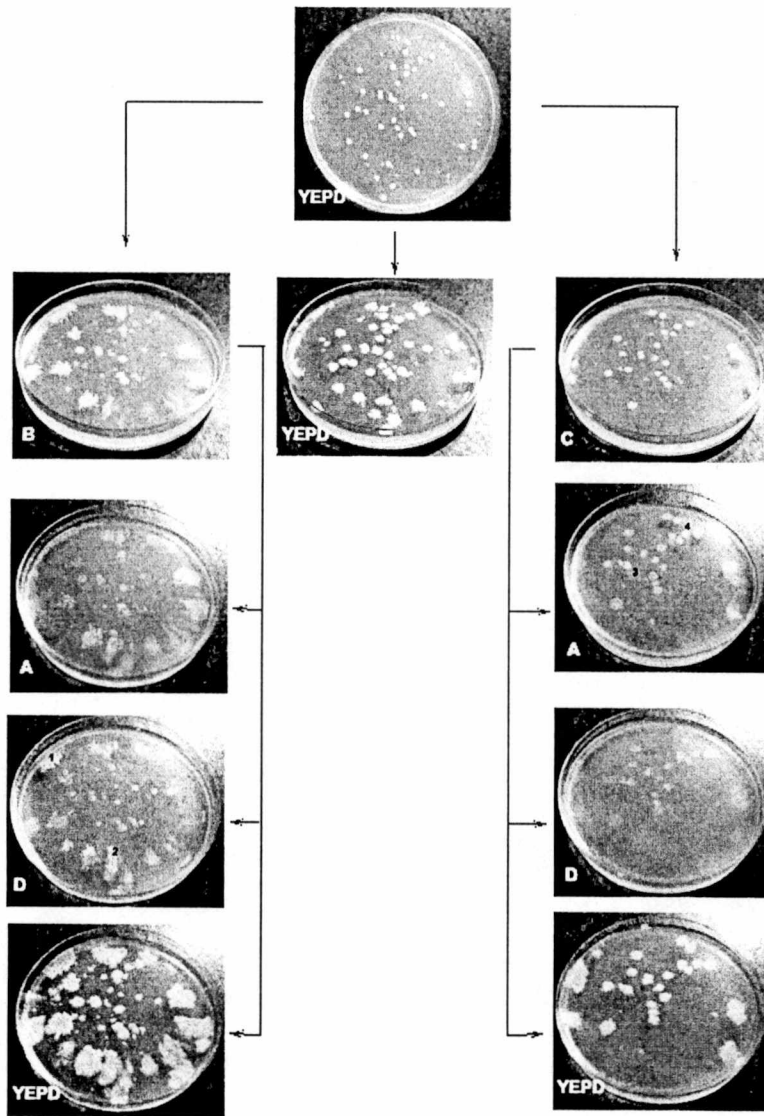


Figure B.3: Computing using replica-plating

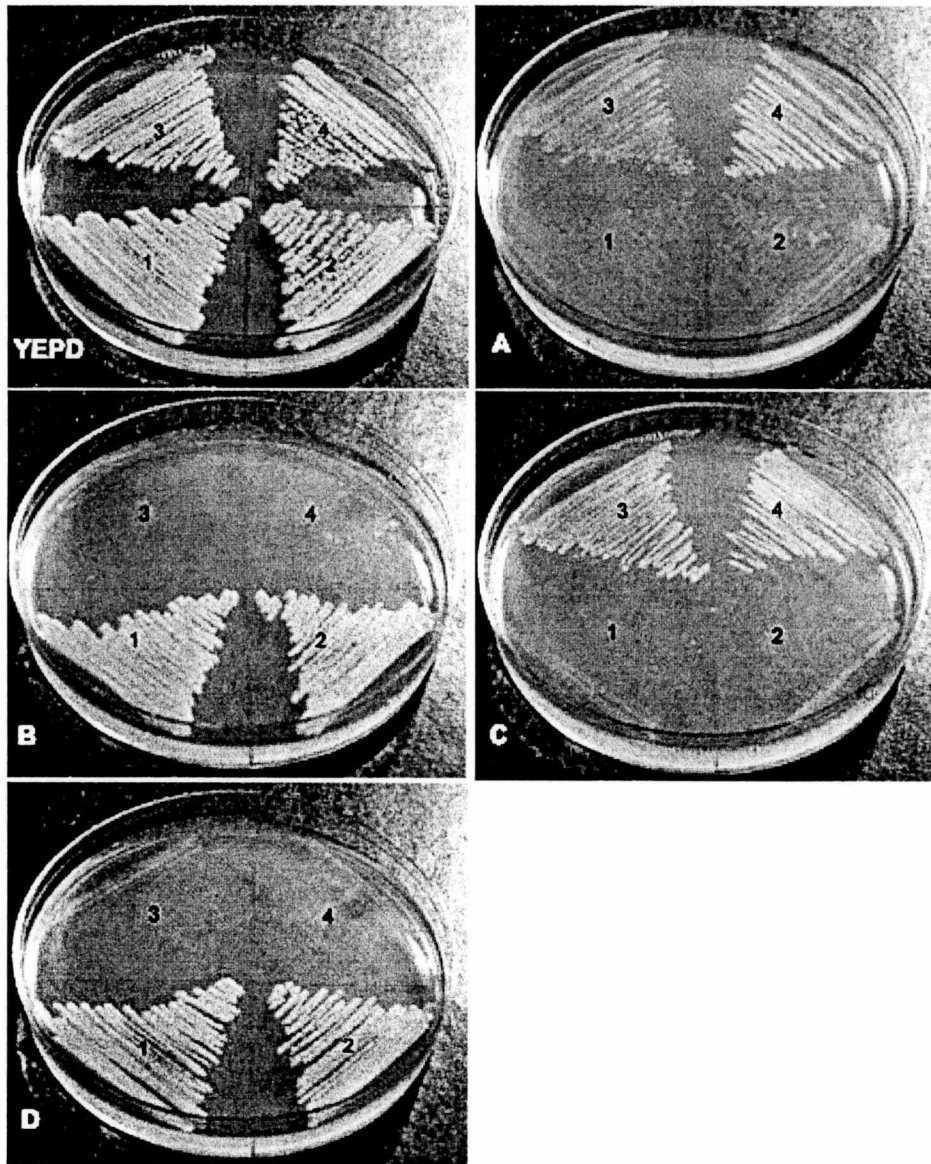


Figure B.4: Genotyping of replica-plating survivors

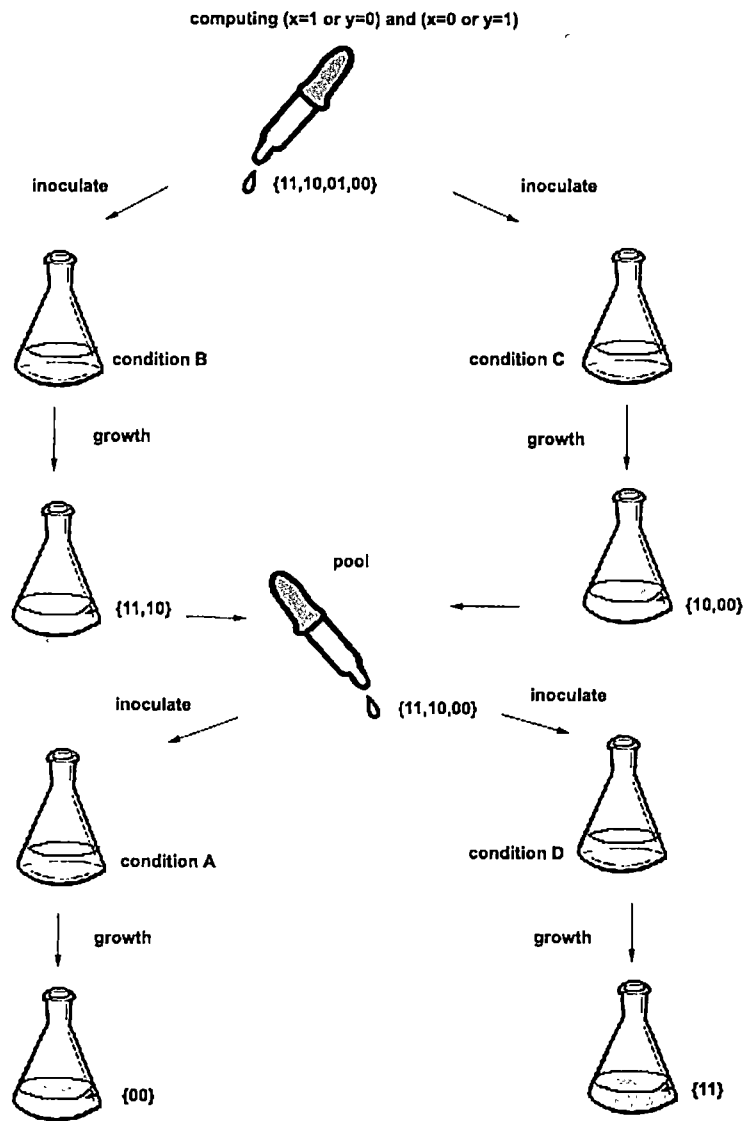


Figure B.5: A liquid-phase algorithm.

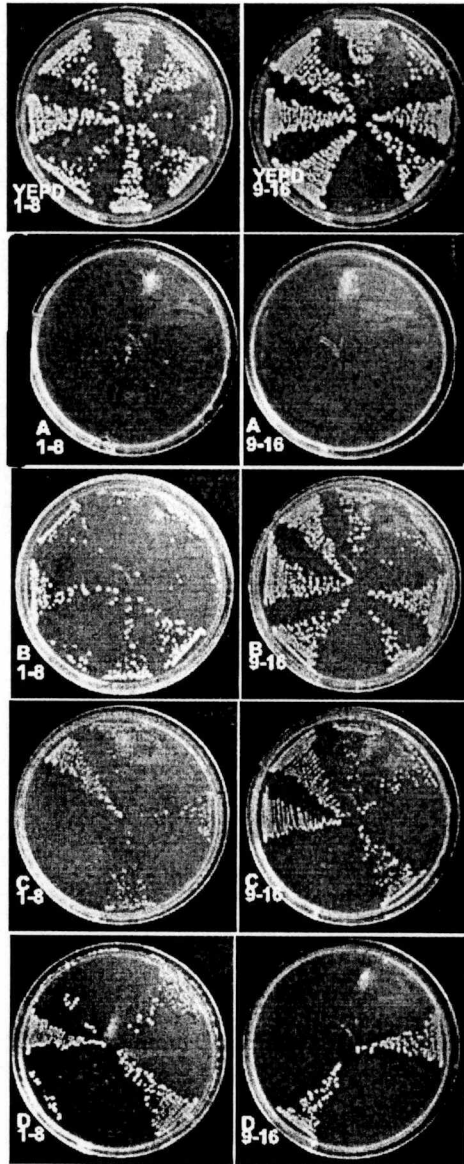


Figure B.6: Problem 1: genotyping of condition B survivors

Table B.1: Problem 1: growth spectra of condition B survivors

colony	YEPD	condition A	condition B	condition C	condition D
1	+	-	+	+	-
2	+	-	+	+	+
3	+	-	+	-	+
4	+	-	+	-	+
5	+	-	+	+	-
6	+	-	+	-	+
7	+	-	+	+	-
8	+	-	+	-	+
9	+	-	+	+	-
10	+	-	+	+	-
11	+	-	+	+	-
12	+	-	+	-	+
13	-	-	-	-	-
14	+	-	+	+	-
15	+	-	+	-	+
16	+	-	+	+	-

Table B.2: Problem 1: summary of condition B survivors

logical equivalence	corresponding colonies/total
11	8/14
10	6/14
01	0/14
00	0/14

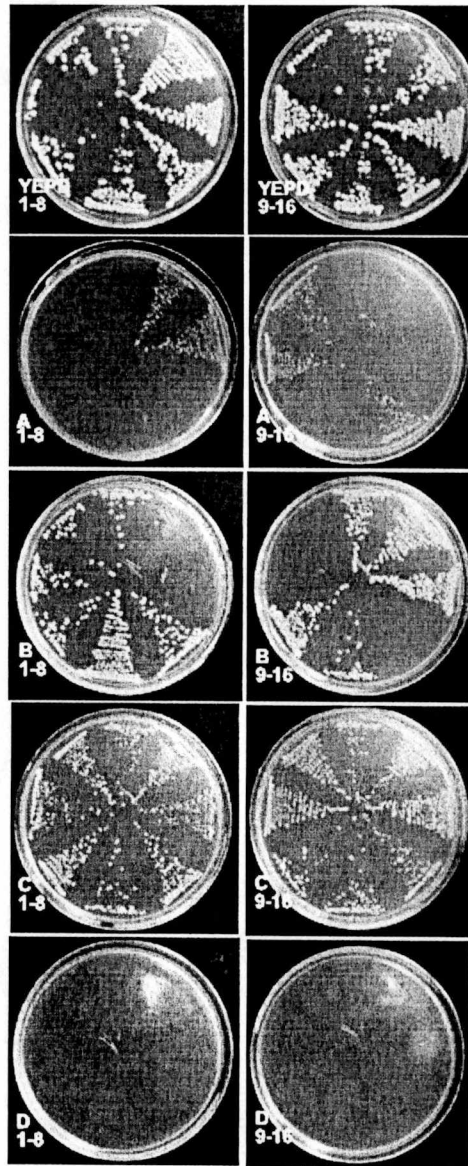


Figure B.7: Problem 1: genotyping of condition C survivors

Table B.3: Problem 1: growth spectra of condition C survivors

colony	YEPD	condition A	condition B	condition C	condition D
1	+	-	+	+	-
2	+	-	+	+	-
3	+	-	+	+	-
4	+	-	+	+	-
5	+	-	+	+	-
6	+	-	+	+	-
7	+	+	-	+	-
8	+	+	-	+	-
9	+	-	+	+	-
10	+	+	-	+	-
11	+	+	-	+	-
12	+	-	+	+	-
13	+	-	+	+	-
14	+	+	-	+	-
15	+	-	+	+	-
16	+	-	+	+	-

Table B.4: Problem 1: summary of condition C survivors

logical equivalence	corresponding colonies/total
11	0/16
10	11/16
01	0/16
00	5/16

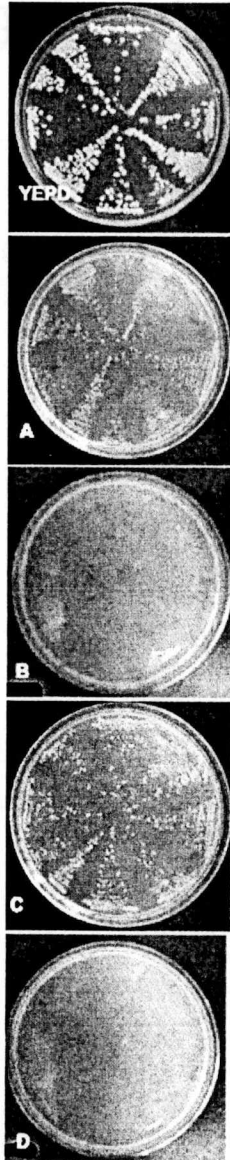


Figure B.8: Problem 1: genotyping of condition A survivors

Table B.5: Problem 1: growth spectra of condition A survivors

colony	YEPD	condition A	condition B	condition C	condition D
1	+	+	-	+	-
2	+	+	-	+	-
3	+	+	-	+	-
4	+	+	-	+	-
5	+	+	-	+	-
6	+	+	-	+	-
7	+	+	-	+	-
8	+	+	-	+	-
9	+	+	-	+	-
10	+	+	-	+	-

Table B.6: Problem 1: summary of condition A survivors

logical equivalence	corresponding colonies/total
11	0/10
10	0/10
01	0/10
00	10/10

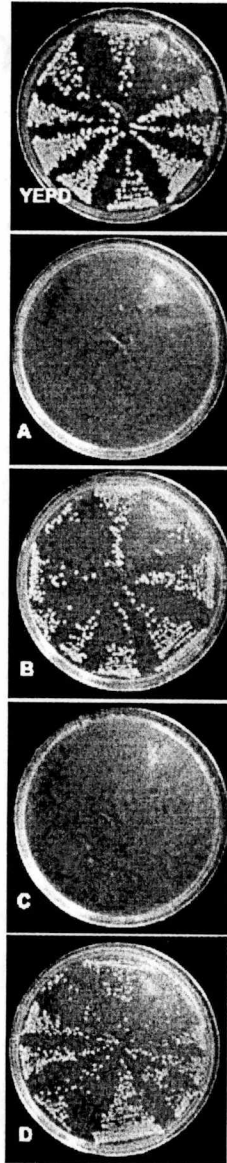


Figure B.9: Problem 1: genotyping of condition D survivors

Table B.7: Problem 1: growth spectra of condition D survivors

colony	YEPD	condition A	condition B	condition C	condition D
1	+	-	+	-	+
2	+	-	+	-	+
3	+	-	+	-	+
4	+	-	+	-	+
5	+	-	+	-	+
6	+	-	+	-	+
7	+	-	+	-	+
8	+	-	+	-	+
9	+	-	+	-	+
10	+	-	+	-	+

Table B.8: Problem 1: summary of condition D survivors

logical equivalence	corresponding colonies/total
11	10/10
10	0/10
01	0/10
00	0/10

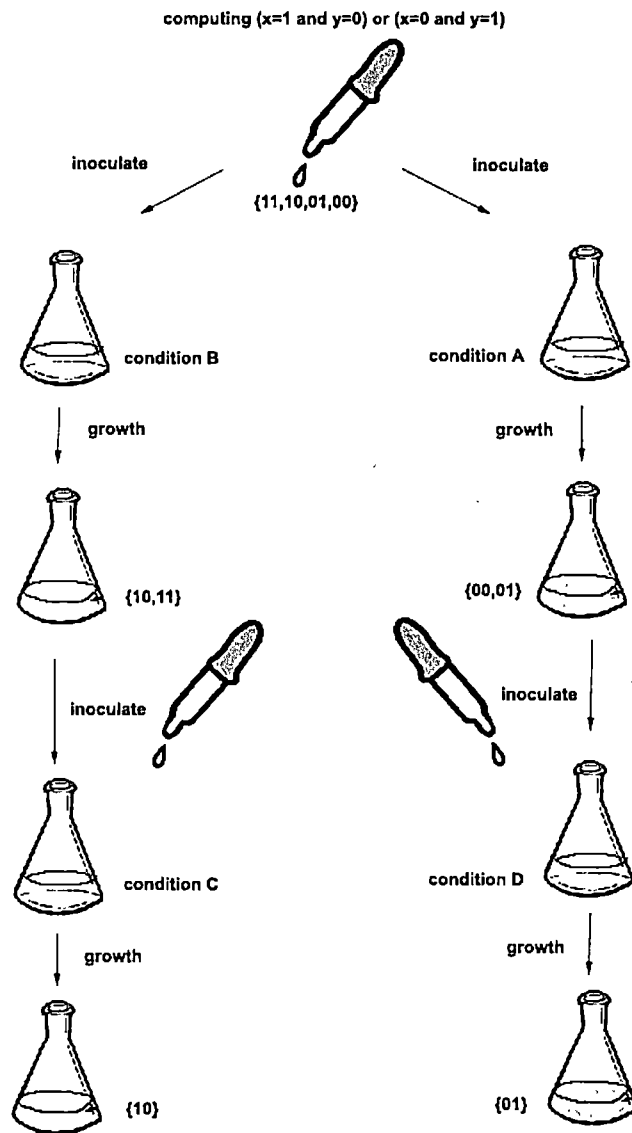


Figure B.10: Problem 2

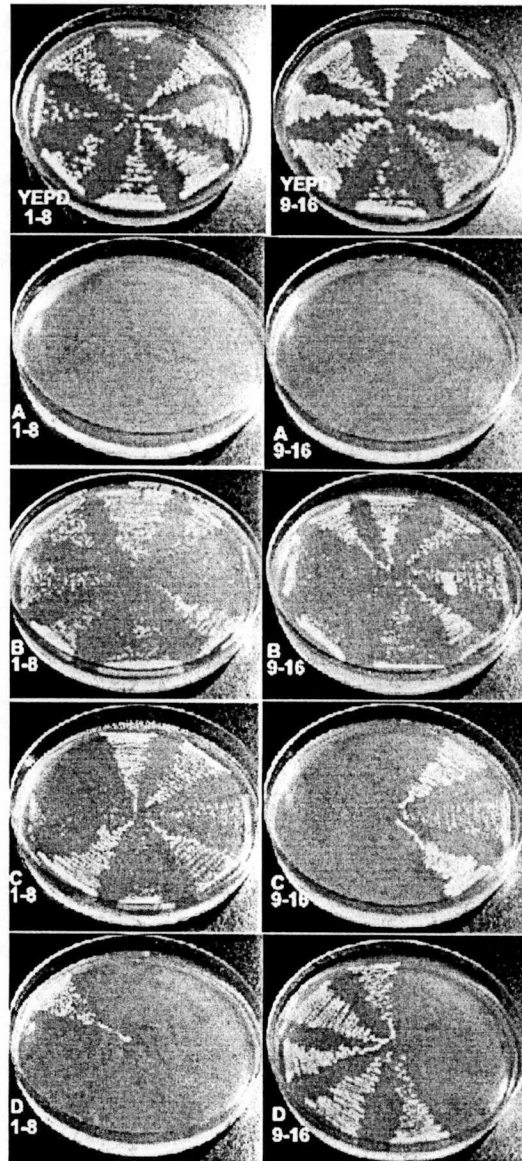


Figure B.11: Problem 2: genotyping of the condition B survivors

Table B.9: Problem 2: growth spectra of condition B survivors

colony	YEPD	condition A	condition B	condition C	condition D
1	+	-	+	+	-
2	+	-	+	-	+
3	+	-	+	+	-
4	+	-	+	+	-
5	+	-	+	+	-
6	+	-	+	+	-
7	+	-	+	+	-
8	+	-	+	+	-
9	+	-	+	+	-
10	+	-	+	-	+
11	+	-	+	-	+
12	+	-	+	-	+
13	+	-	+	-	+
14	+	-	+	+	-
15	+	-	+	+	-
16	+	-	+	+	-

Table B.10: Problem 2: summary of condition B survivors

logical equivalence	corresponding colonies/total
11	5/16
10	11/16
01	0/16
00	0/16

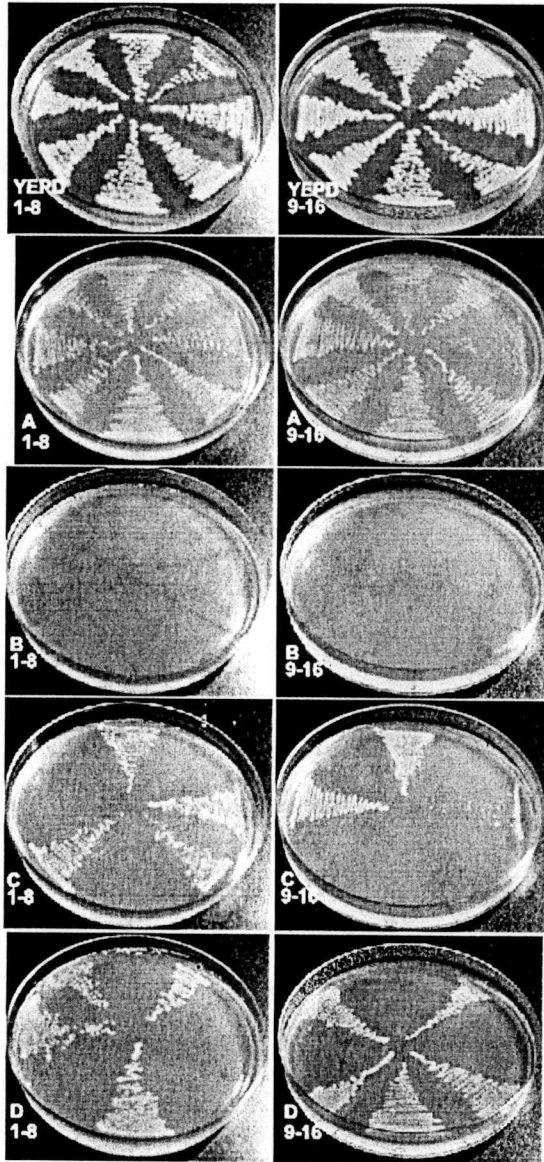


Figure B.12: Problem 2: genotyping of the condition A survivors

Table B.11: Problem 2: growth spectra of condition A survivors

colony	YEPD	condition A	condition B	condition C	condition D
1	+	+	-	+	-
2	+	+	-	-	+
3	+	+	-	-	+
4	+	+	-	+	-
5	+	+	-	-	+
6	+	+	-	+	-
7	+	+	-	+	-
8	+	+	-	-	+
9	+	+	-	+	-
10	+	+	-	-	+
11	+	+	-	+	-
12	+	+	-	-	+
13	+	+	-	-	+
14	+	+	-	-	+
15	+	+	-	+	-
16	+	+	-	-	+

Table B.12: Problem 2: summary of condition A survivors

logical equivalence	corresponding colonies/total
11	0/16
10	0/16
01	9/16
00	7/16

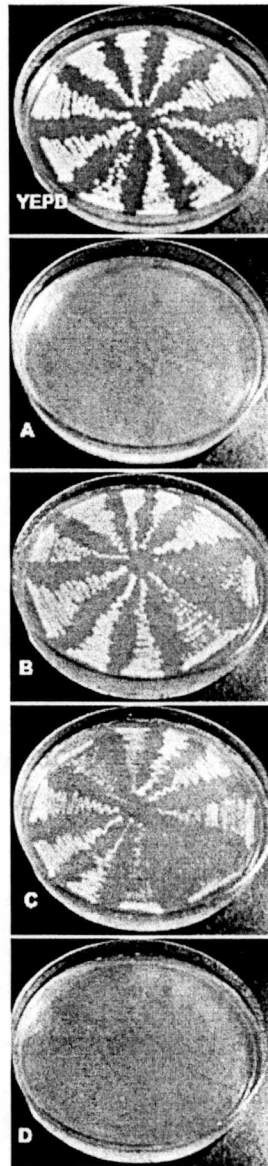


Figure B.13: Problem 2: genotyping of the condition C survivors

Table B.13: Problem 2: growth spectra of condition C survivors

colony	YEPD	condition A	condition B	condition C	condition D
1	+	-	+	+	-
2	+	-	+	+	-
3	+	-	+	+	-
4	+	-	+	+	-
5	+	-	+	+	-
6	+	-	+	+	-
7	+	-	+	+	-
8	+	-	+	+	-
9	+	-	+	+	-
10	+	-	+	+	-

Table B.14: Problem 2: summary of condition C survivors

logical equivalence	corresponding colonies/total
11	0/10
10	10/10
01	0/10
00	0/10

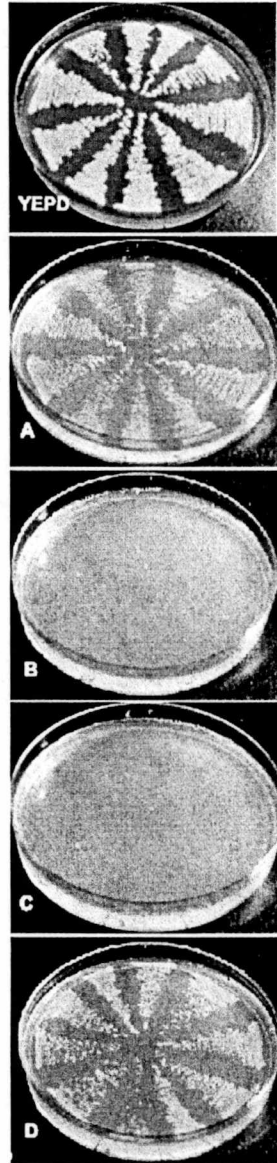


Figure B.14: Problem 2: genotyping of the condition D survivors

Table B.15: Problem 2: growth spectra of condition D survivors

colony	YEPD	condition A	condition B	condition C	condition D
1	+	+	-	-	+
2	+	+	-	-	+
3	+	+	-	-	+
4	+	+	-	-	+
5	+	+	-	-	+
6	+	+	-	-	+
7	+	+	-	-	+
8	+	+	-	-	+
9	+	+	-	-	+
10	+	+	-	-	+

Table B.16: Problem 2: summary of condition D survivors

logical equivalence	corresponding colonies/total
11	0/10
10	0/10
01	10/10
00	0/10

Appendix C

Glossary of Genetic Jargon and Abbreviations

- allele: organisms of the same species share a set of genes, but this doesn't mean that any one particular gene has the identical DNA sequence from one individual to the next. Oftentimes, there are differences, sometimes important ones. These differences are termed as *allelic*, and an individual's particular set of genes are called its *alleles*. Allelic differences are important to the yeast computer. A yeast's alleles determine the logical entity it represents.
- amino acid: the chemical subunits of protein.
- bactoagar: extract of seaweed, liquifies as high temperatures, solidifies at room temperature; used to make agar plates on which to grow yeast colonies.

- complement: In the parlance of genetics, the strand ATCG is the complement of TAGC. See the definition of DNA for more details.
- DNA: abbreviation for deoxyribonucleic acid. In nature, it's most often found as a double-stranded helix. The two helices are composed of linked, linear arrays of nucleotides or *bases*—designated A, T, G and C. In the context of a double-stranded helix, A is always found across from a T on the complementary strand. Likewise, G is always found across from a C.
- gene: any segment of DNA which encodes a molecule (usually a protein).
- genome: an organism's entire genetic content.
- genotype: loosely synonymous with genome. Often, an organism's genotype is stated in reference to just a few genes to distinguish it from others that have similar genomes but have allelic differences in those particular genes.
- genotyping: the act of determining an organism's genotype.
- gms: abbreviation for *grams*.
- mgs: abbreviation for *milligrams*.
- mls: abbreviation for *milliliters*.
- mutant: an organism or gene that harbors a mutation.
- mutation: any inheritable change in the DNA sequence of a gene. Some mutations have consequences, most don't.

- phenotype: the observable traits of an organism; the physical manifestation of an organism's genome.
- protein: a linear array of amino-acids specified by a gene.
- strain: a genetically homogeneous line of something living (e.g., a yeast strain).
- YEPD: abbreviation for *yeast-extract*, *peptone*, *dextrose*. This is a very nutrient-rich medium used to grow a variety of mutants that can't make certain nutrients themselves.
- YNB: abbreviation for *yeast nitrogen base*.
- wild-type: a genome with no mutations.

Appendix D

Media Formulations

D.1 Condition A

D.1.1 agar plates

Add 10 grams of glucose + 10 grams of bactoagar to 430 milliliters of distilled water and autoclave for 30 minutes. While this is autoclaving, filter-sterilize the following mixture: 50 milliliters of 10X YNB without amino acids or ammonium sulfate + 1.5 milliliters of 1% lysine + 1 gram of amino adipate dissolved in 20 milliliters of distilled water. Use a 20 micron filter for the filter-sterilization. After the former is finished autoclaving and has cooled for 10 minutes in a 50 degree celsius water bath, combine the two solutions and swirl vigorously to mix. Dole out into petri dishes to the desired thickness.

D.1.2 liquid

Filter-sterilize the following mixture:

12.5 mls of 10X YNB without amino acids or ammonium sulfate

0.4 mls of 1% lysine

1.25 mls of 2 mg/ml uracil

5 mls of 50 mg/ml amino adipate

0.5 gms of glucose

106 mls of distilled water

D.2 Condition B

D.2.1 agar plates

Add 10 grams of glucose + 10 grams of bactoagar to 450 mls of distilled water and autoclave for 30 minutes. While this is autoclaving, filter-sterilize the following mixture: 50 mls of 10X YNB + 5 mls of 2 mg/ml uracil. After the former has cooled, combine the two solutions and aliquot into sterile petri dishes.

D.2.2 liquid

Filter-sterilize the following mixture:

12.5 mls of 10X YNB

1.25 mls of 2 mg/ml uracil

0.5 gms of glucose

111 mls of distilled water

D.3 Condition C

D.3.1 agar plates

Add 10 grams of glucose + 10 grams of bactoagar to 375 mls of distilled water and autoclave for 30 minutes. Meanwhile, make the following mixture: 0.5 gms of 5-fluorouracil (5-FOA) + 50 mls of 10X YNB + 1.5 mls of 1% lysine + 5 mls of 2 mg/ml uracil + 69 mls of distilled water. Heat on a hot-plate with vigorous stirring until the flakes of 5-FOA dissolve. This takes a while but don't allow to boil. Filter-sterilize. Once the latter has cooled, combine and pour plates.

D.3.2 liquid

Filter-sterilize the following mixture:

0.5 gms glucose

0.125 gms 5-FOA

111 mls distilled water

0.4 mls of 1% lysine

1.25 mls of 2 mg/ml uracil

12.5 mls of 10X YNB

D.4 Condition D

D.4.1 agar plates

Add 10 grams of glucose + 10 grams of bactoagar to 450 mls of distilled water and autoclave for 30 minutes. Upon cooling, add 1.5 mls of 1% lysine + 50 mls of 10X YNB.

D.4.2 liquid

Filter-sterilize the following mixture:

0.5 gm glucose

12.5 mls of 10X YNB

0.4 mls 1% lysine

112 mls distilled water

D.5 Condition I, alias "YEPD"

D.5.1 agar plates

Dissolve 5 gms of yeast extract + 10 gms of peptone + 10 gms of glucose + 10 gms of bactoagar in 500 mls of distilled water and autoclave.

D.5.2 liquid

Same as above except omit the bactoagar.

D.6 Misc solutions

- 1% lysine: 1 gm of L-lysine in 100 mls of distilled water; filter-sterilize.
- 2 mg/ml uracil: 0.2 grams uracil in 100 mls of distilled water; filter-sterilize.
- aminoadipate: buy DL-alpha-aminoadipic acid from Sigma. Add 1 gm to a 50 ml conical tube + 20 mls of distilled water. Add glacial acetic acid until dissolved. This is tricky. Add just 100 microliters at a time until about .5 mls have been added. Then add just 50 microliters at a time. Vortex vigorously and monitor the pH of the solution using pH paper. Stop adding acid when the pH spikes; avoid adding excess. Don't worry if a small amount of the chemical doesn't seem to dissolve-the selection will still work.

Vita

Colton Smith was born in Memphis, obtained a BA from the University of Tennessee in Knoxville in 1984 and a PhD in virology from Harvard in 1988. He received the MS degree in computer science from the University of Tennessee in December 2001.