

Performance Limits on Chemical Computation

by

George E. Homsy II

B.S.E. Electrical Engineering and Computer Science
University of California at Berkeley, 1987

S.M. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 1997

Submitted to the Department of Electrical Engineering and Computer Science in
partial fulfillment of the requirements for the degree of

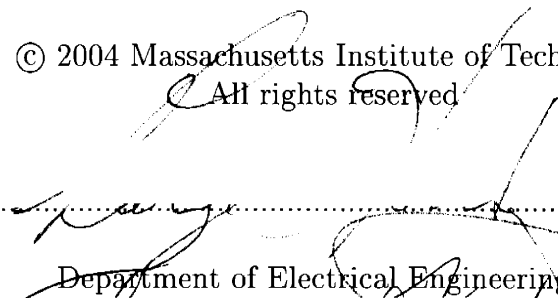
Doctor of Philosophy

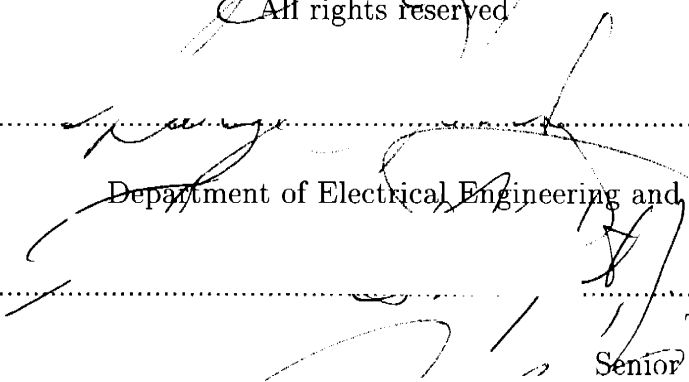
at the

Massachusetts Institute of Technology

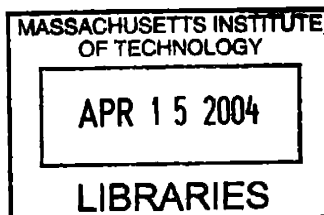
February, 2004

© 2004 Massachusetts Institute of Technology
All rights reserved

Author.....  George Homsy
Department of Electrical Engineering and Computer Science

Certified by.....  Thomas F. Knight
Senior Research Scientist
Thesis Supervisor

Accepted by.....  Prof. Arthur C. Smith
Chairman, Department Committee on Graduate Students



ARCHIVES

Performance Limits on Chemical Computation

by

George E. Homsy II

Submitted to the Department of Electrical Engineering and Computer Science
on January 9th, 2004 in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

ABSTRACT

A class of novel computers uses solute concentrations of distinct chemical species as logic signals and diffusion for signal transport. I establish a bound on the speed, density, and error rate of such computers from first principles.

I let the chemical computer engineer choose a “design tuple” of independent parameters: number of chemical species, total solute concentration, signal molecule size, and a parameter called the “cell size”. I establish a functional relation between the design tuple, and the “performance tuple”: (operating frequency, signal density, error rate). I give a lower bound on the probability of a logic error in one computation step, and an upper bound on the frequency of operation, both as functions of the design tuple.

I evaluate these bounds for ssDNA oligomers, and conclude that DNA computation has unacceptable error rates if the hybridization regions are less than eight nucleotides in length.

I then argue that, given a suitable scalar-valued performance metric as a function over performance tuples, there is a globally optimal design tuple maximizing performance. I present two conjectures seeking to explain (a) why neurons use small molecules to transport information, and (b) why cells have the size they do.

In part two I develop such a performance metric based on Toffoli’s *computation capacity* and *computation density*, extending and generalizing in these ways:

- as a local statistic on a uniform system, it can be evaluated for very large systems without exhaustive counting
- it explicitly takes the error rate of the underlying physical process into account

I then show a relation between this metric, and a quantity of dynamic systems called *specific ergodicity*. This is a novel result of theoretical importance, and is the central result of part two. It allows me to unambiguously compare the utility of computers varying vastly in speed, error rate, and signal density.

I compare the maximum possible performance of proposed DNA computers from the literature with current commodity electronic computers, and conclude that diffusion-driven, DNA hybridization based computers cannot exceed the performance of current electronic computers by more than a factor of 40000, and probably by much less.

Thesis supervisor: Thomas F. Knight
Senior Research Scientist

For my grandfather,
George E. Homsy

Special thanks to

- From the AI lab:
 - Tom Knight, Norm Margolus, Gerald Sussman, as well as Raissa D'Souza, for getting me started on this path, and for helping to refine my view of "The Way Things Must Be"
 - Gill Pratt, Hugh Herr, and the leglab crew; some of the most amazing engineers on the planet
 - Hal Abelson, Gerald Sussman, and the "amorphous crew": Becky Bisbee, Ron Weiss, Radhika Naghpal *et al.*
 - Ron Wiken, the man with a plan, and the keeper of The Dangerous Tools
 - The Tom Knight crew: Bunnie, Jeremy Brown, Nick Papadakis *et al.*; the Outside Thinkers
 - Dan and Jessica and the "San Francisco Axis"
 - Lisa Tucker-Kellog and Vera Ketelboeter
- Bina Altera, Jody House, Emily Dotton, Ida Martinac, and Melissa Colin, for love, compassion, and faith
- Laird Nolan, Ariane Martins, Katharina Trede, Jeremy Zucker, and Ann Sweeney, for emotional support in hard times
- Etienne de la Croix, for helping me stay sane
- Monica, Marilyn, Claire, and Peggy in the graduate office, for support and hard work, and always leaving me with a smile on my face
- Larry Frishkopf, for his understanding and patience
- Leroy, for sleeping next to me and purring me to sleep each night; and Ninja, for cleaning my ears and purring me awake each morning
- My family: Bud, Bryn, and Rob – every bit a part of me
- my advisor, Tom Knight – for patience, inspiration, enthusiasm, wit, and for always being true to his own Way of Happiness

Contents

I	Chemical Computation	13
1	Introduction to Chemical Computation	15
1.1	Roadmap	16
1.2	Overview of the model	17
1.3	Previous Work	21
1.3.1	Computation in living systems	21
1.3.2	Synthetic chemical computers	21
1.3.3	Theoretical considerations	21
1.3.4	Randomness, simulations, <i>etc.</i>	23
2	Performance Envelope of Chemical Computers	25
2.1	Model of computation	26
2.1.1	Definitions	28
2.1.2	Relations governing performance	29
2.2	The Molecules:	
	Relations between D_1 , D_L , v_1 , v_L , and L	31
2.2.1	Diffusion constant scaling	31
2.2.2	Volume scaling	32
2.3	The Transport Mechanism:	
	Relation between D_L , ΔV , and f	34
2.4	Geometry of the Reactor Vessel:	
	Relation between α , v_L , ΔV , N , and r	36
2.5	Limitations on the Code:	
	Relation between N , L , and d_{exp}	37
2.5.1	The approach	37
2.5.2	Definitions	37
2.5.3	A restatement of the problem	39
2.5.4	An upper bound on d_{exp}	39

2.6	The Gating Process: Relation between Δg_1 , d_{exp} , and $\langle \varepsilon \rangle$	42
2.7	Discussion	45
3	Biopolymers and DNA Computation	51
3.1	Applications: DNA Computers	52
3.1.1	Chemical and mechanical properties of DNA	52
3.1.2	General characteristics of proposed <i>in vitro</i> DNA computers	54
3.1.3	Linear bound on mismatch energy	55
3.1.4	Some reasonable operating parameters	56
3.1.5	Performance envelope of proposed <i>in vitro</i> DNA computers	57
3.1.6	Discussion	59
3.1.7	Summary of findings	66
3.2	Applications: Protein Computation in cells	68
3.2.1	Chemical and mechanical properties of proteins	68
3.2.2	Some reasonable operating parameters	70
3.2.3	Performance envelope of protein-based cellular information processing	71
4	Conclusions on Chemical Computation	73
4.1	Review	74
4.1.1	Implicit restrictions on random chemistries	74
4.1.2	Errors due to randomness in chemical reaction systems . .	75
4.1.3	Transport phenomena and computation speed	75
4.1.4	Discussion	77
4.2	Speculation: Neurons <i>vs.</i> other cell types	78
4.3	Speculation: Why are cells the size they are?	79
II	Measuring Computation	81
5	Introduction to Computation Density	83
5.1	Computational media	85
5.2	Traits of computational media	86
5.3	Proposed taxonomy of computational systems and media	88
5.4	Techniques	89
5.5	Roadmap: toward computation density of infinite, dust-like, stochas- tic media	91
5.5.1	Real-ization: computation density of error prone systems .	93

5.5.2	Measurement: relation of computation capacity to specific ergodicity for deterministic systems	94
5.5.3	How good is my gunk?: computation density of infinite, dust-like, error-prone media	95
5.5.4	Discussion	96
5.6	Applications	96
6	Computation Capacity and Computation Density	99
6.1	Toffoli's computation capacity and computation density	100
6.2	A statistical approach to computation capacity	101
6.2.1	Estimation techniques	102
6.3	Eliding the distinction between program and input	104
6.4	Computation density in spatially distributed systems	109
6.5	Computation density in time-evolving systems	110
6.6	Infinite systems: Cellular Automata	115
7	Computation Density of Error-prone Systems	121
7.1	Error-prone systems	122
7.2	Functional effectiveness	123
7.2.1	Useful specializations	124
7.3	Functional effectiveness as a measure of computation	125
7.3.1	A proposed measure	125
7.3.2	Discussion	126
7.3.3	Applied to feedback automata	129
7.4	Estimating computation density of error-prone media	131
8	Computation Density, Information Density, and Specific Ergodicity	133
8.1	Introduction	134
8.2	Review of specific ergodicity	135
8.3	Finite feedback automata	136
8.4	Infinite feedback automata (CA's)	138
8.5	Error-prone infinite feedback automata (stochastic CA's)	139
8.5.1	Discussion	139
8.5.2	Examples	140
9	Conclusions	143
9.1	Review of part II	144
9.2	Computation density of a chemical computer	145

9.2.1	Discussion	147
9.2.2	DNA computers revisited	148
9.2.3	Comparison with electronic computers	148
9.3	Incompleteness of work	151
9.3.1	Theoretical treatment of error correction	152
9.3.2	Enhancement of error model	152
9.3.3	Weakness in specificity	153
9.4	Future Work	153
9.4.1	Gene Expression Logic	153
9.4.2	Abstract models of chemistry	154
9.4.3	Simulations	154
9.4.4	Performance bounds for electronic computers	155
9.4.5	More thorough investigation of multicellularity	156
9.4.6	Sensitivity analysis of GEL systems	156
9.4.7	Generalization to arbitrary stochastic processes	156
III Appendices		157
A Sub-Additivity of Computation Capacity		159
B Relation between d_{\min} and d_{opt}		163

Part I

Chemical Computation

Chapter 1

Introduction to Chemical Computation

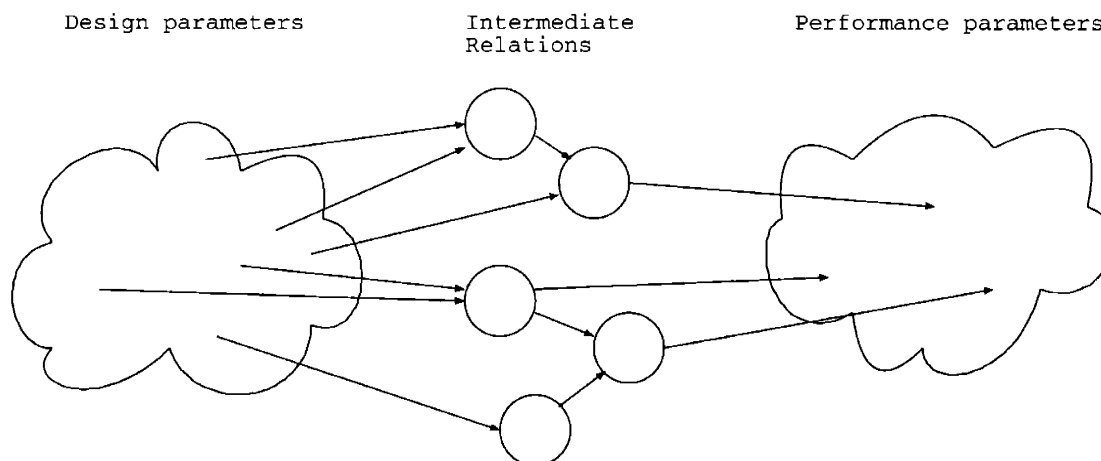


Figure 1.1: The basic plan of part one: establish a set of design parameters for chemical computers, and a set of performance parameters. Then develop a set of intermediate relations in order to show a direct functional dependence of the performance parameters on the design parameters.

1.1 Roadmap

Part one of this thesis is concerned with chemical computers. Here, I advance a specific abstract model for chemical computation which applies well to currently proposed chemical computers in the literature. I develop a series of theoretical analyses which, taken together, establish upper bounds on the “performance parameters” (speed, signal density, error rate, *etc.*) of chemical computers, in terms of a set of independently chosen design parameters, as shown schematically in figure 1.1. I then apply the results of my analyses to some specific chemical computation models: DNA hybridization based computing, and protein-interaction based computation in living cells.

In part two, I turn to the problem of direct comparison of chemical computers with electronic computers. This is difficult because the performance parameters of chemical computers can differ so radically from those of electronic computers: The “sweet spot” in electronic computer design is at high speed and low error rate, whereas chemical computers might be better toward much higher density, but lower speed and higher error rate. What I’m looking for in part two is to find a reasonable and general “figure of merit” for any computer – phrased in terms of the performance parameters – which will allow me to compare chemical computers with electronic computers in an unequivocal way. See figure 1.2 for a schematic

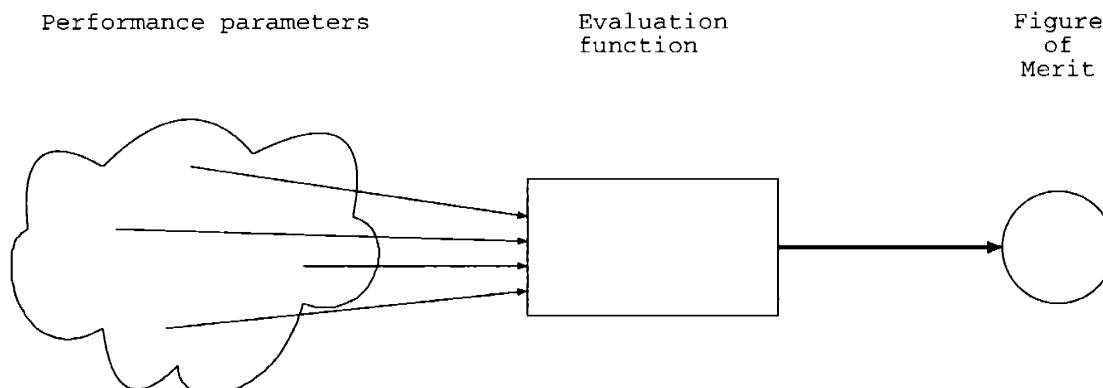


Figure 1.2: The basic plan of part two: Arrive at a reasonable and general “figure of merit”, a function of the performance parameters, which allows us to evaluate the overall usefulness of a computer *as a single number*.

representation.

Finally, at the end of part two, I combine these results to establish a direct comparison between chemical and electronic computers, as shown in figure 1.3. I give chemical computers the benefit of the doubt, assuming all design parameters are optimized to give the best figure of merit – while for electronic computers, I take the performance parameters available from any given commodity-level desktop workstation. In this case, highly skewed in favor of chemical computers, I conclude that chemical computers can never exceed electronic computers in performance by more than a factor of about 40000, and probably by much less¹.

1.2 Overview of the model

Current models of chemical computation differ from the conventional electronic model: signals are not carried on wires, but are co-distributed in solution and diffuse freely. Signal molecule diffusion is an important – indeed necessary – part of chemical computer action.

In electronic computers, the sequestering of signals to wires allows distinc-

¹The actual difference in performance is probably much less due to several factors: Unknown parameters in the chemical case, I've set at their optimal values; The bound on chemical computer performance parameters is very loose (actual performance is probably much lower); And electronic computers will continue to improve for some time yet.

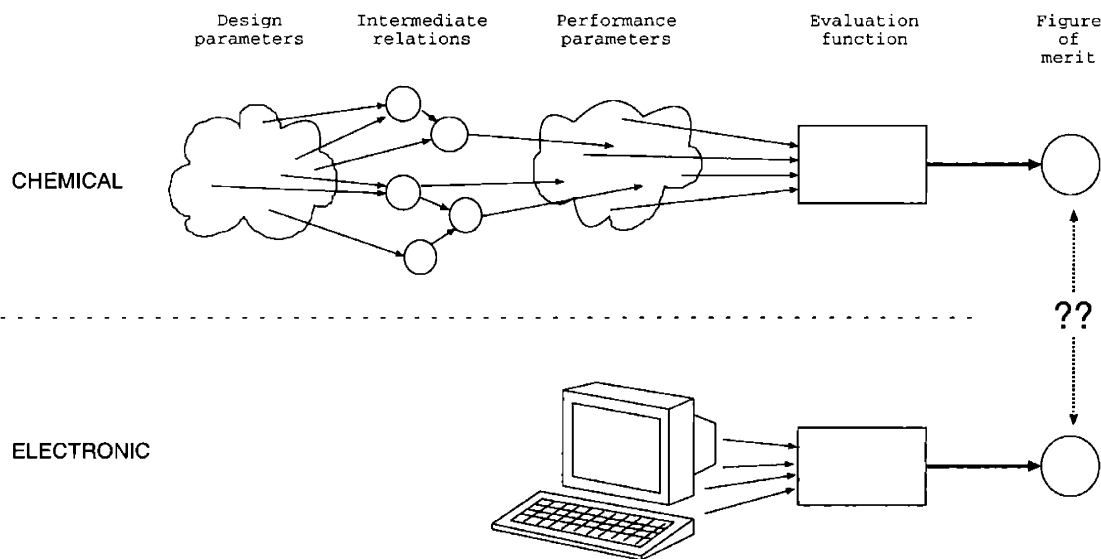


Figure 1.3: Using the combined results of parts one and two, to make a direct comparison between chemical and electronic computers.

tion of one signal from another. But in chemical computation, since signals are spatially comingled, they must be distinguished by some other mechanism. In all schemes proposed thus far, a different chemical species is chosen for each signal, and the differences in chemical binding specificity distinguish the signals.

I consider three distinct “performance parameters” as being adequate to describe the performance of a computer: Spatial density of logic signals, speed of operation, and error rate. I propose a particular set of physical restrictions under which a chemical computer is assumed to operate, and I develop several relations which determine an ultimate “performance envelope” within which the performance parameters of *any* such chemical computer must lie.

I then investigate the size and shape of this envelope by theoretical analyses: The chemically based signal distinction mechanism, combined with some reasonable assumptions about the classes of molecules being used as signals, some information theoretic tools, some elementary statistical mechanics, and simple diffusion kinetics, can give us fundamental upper bounds on the extent of the chemical computer performance envelope.

I assume the design variables the computer engineer may choose for a chemical

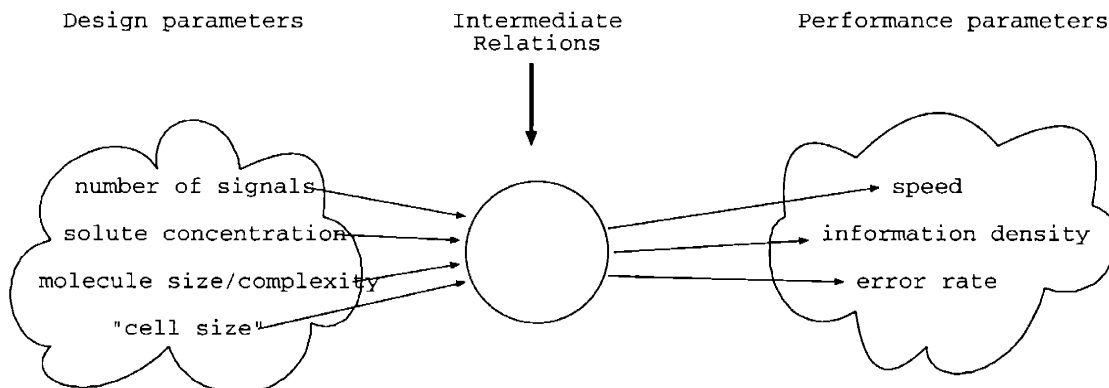


Figure 1.4: The strategy for part one of this thesis, showing the set of design parameters I assume for a chemical computer and the set of performance parameters. Part one of this thesis concerns developing the “intermediate relations”, to obtain bounds on the performance parameters in terms of the design parameters.

computer are: The operating density, the “cell size”², the length³ of the signaling molecules, and the number of signals. From these I derive bounds on the operating frequency and error rate of such a computer, in terms of these independent design variables and some constants related to the chemistry being used. This set of analyses is diagrammed schematically in figure

The maximum operating frequency is derived from the cell size and the signal molecule diffusion constant, which is in turn derived by applying a scaling relation to the diffusion constant of a monomer and the signal molecule length.

A bound on the error rate is determined as follows: First, we consider the signal molecules as being represented by codewords in a space of strings of length L . Now, under some reasonable assumptions, the number of signals combined with the codeword length and the base of the code determines an upper bound on the minimum separation distance⁴ between codewords. I apply an interaction energy scaling relation to this minimum separation distance to obtain an upper bound on the difference in Gibbs free energy between binding of the correct signal molecule and an incorrect signal molecule. I then apply the Maxwell-Boltzmann

²To clarify: we are not actually considering physically separated cells, but to simplify the analysis, it is convenient to quantize the spatial extent of the computer into some arbitrarily chosen cell size.

³Or more generally, the length of string needed to describe them.

⁴for some reasonable definition of “distance”, *e.g.* Hamming distance

distribution to this energy separation to find a lower bound on the fractional occupancy of receptor sites by incorrect signal molecules. I take this fractional occupancy as a error probability, hence arriving at a lower bound on the error rate of the computer, in terms of the design parameters.

1.3 Previous Work

Before developing my model in detail, I will begin by surveying relevant results from previous work on chemical computation.

1.3.1 Computation in living systems

Monod and Jacob suggested [55] that organisms regulate gene expression using logic-like functions. Many have since noted [69, 70, 56, 10, 16] that living systems implement logic functions using biochemical reaction networks. Smith and Schweitzer [68] argue that in fact, many biological functions performed by cells are Turing complete.

1.3.2 Synthetic chemical computers

Not all work in this field has been analytic. Engineers and computer scientists have long dreamed of a synthetic approach to designing chemical computers. In the early 1990's Hjelmfelt *et al.* [38, 39, 37] put the field on a firm theoretical footing, by giving explicit design procedures for chemical gates, amplifiers, perceptrons, *etc.* In 1994 Adleman [3] computed the solution to an instance of the Hamiltonian Path problem *in vitro* using DNA hybridization. In 1995, Bray [16], Lipton [45], Smith and Schweitzer [68], Roweiss *et al.* [66], and others expanded theoretically on the work of Adleman, by proposing various Turing complete chemical computer designs, at least on paper.

In 1997 Knight and Sussman [41] gave an example of how logic gates might be implemented using synthetic genetic regulatory networks. Weiss *et al.* have expanded upon this work, and provided more details on the approach, in [80] and [79].

1.3.3 Theoretical considerations

Magnasco [47] has offered a proof that chemical kinetics is Turing universal. Magnasco treats chemistry as abstract: He chooses an abstract set of reactions and rate constants and then assumes a set of chemical species can be found to meet those specifications. It is one contribution of this work to demonstrate that at least for biopolymers, this assumption is not borne out in practice: that crosstalk is an unavoidable consequence of chemical computation; that crosstalk is bounded

below by the density of signals used; and hence that any single-vessel chemical reaction system based on biopolymers has bounded computation density.

This work is certainly not the first nor the only treatment of error in chemical computation – various authors have discussed the speed and reliability of chemical computation in considerable depth. For instance, Rose *et al.* [62] give a statistical mechanical treatment of the mismatch probability in DNA computing, which they term the “Computational Incoherence”, ξ . Other theoretical treatments of error for particular models of chemical computation have been given by Roweis *et al.* [65], Winfree *et al.* [81], Rose *et al.* [63, 64], and others.

Roweis *et al.* [65], and Karp *et al.* [42] have discussed error correction techniques in the context of DNA hybridization based computing. These approaches share the premise that a certain error rate is inherent in the process of combinatorial generation of possible solutions, and extraction of valid solutions, from a mixture of hybridized DNAs. They discuss the error rate *vs.* time tradeoff attainable by using simple consensus-based or retry-based error detection and correction schemes.

Another line of inquiry pertains to error minimization in the underlying chemical process, rather than *post facto* error correction, as discussed by the above papers. The majority of approaches to this problem center on informed choice of ssDNA oligonucleotide signal sequences. Adleman’s original experiment used randomly chosen sequences, and hinted that enlightened sequence design was likely to give better results. Deaton *et al.* [20, 19, 21], Smith [67], Frutos *et al.* [28], Zhang and Shin [83], Hartemink and Gifford [35], Brenneman and Condon [17], Ackermann and Gast [2], and Penchovsky and Ackermann [59] have all given methods for enlightened oligonucleotide sequence design.

A completely different approach to error rate analysis comes from Bialek [15], who discusses the relation between switching speed and spontaneous switching time in bistable chemical switches. In the terms of this study, the spontaneous switching time is an error rate, and the switching speed gives a maximum effective operating frequency. Bialek’s approach is distinguished from the previously mentioned work, in at least two respects:

- it deals with a completely abstract model of chemistry; no assumptions are made on the structure of the underlying signal molecules
- the computational model is a time-dependent switching system, rather than a combinatorial search system as described by Adleman and subsequent authors

We will try in our analysis to be informed by this approach – but with more emphasis on both physical chemistry of error, and on systems design. Experimental treatment of this system with our measure of computation density would provide a first order applied test of the theory developed in this thesis.

1.3.4 Randomness, simulations, *etc.*

Real chemical systems behave stochastically, and this effect is especially apparent at extremely low concentrations [30, 31]. Of course this low concentration condition occurs quite frequently in biochemistry⁵. McAdams *et al.* [54][52][53][11] make the case very pointedly for use of stochastic methods in chemical simulation and analysis, rather than the more conventional differential equation approach, for treatment of gene expression systems. They move on to explicit simulations, developing and using a simulator based on earlier theoretical work by Gillespie [30, 31] on simulating the chemical Master equation.

Peccoud and Ycart [58] also note that gene expression must be treated stochastically to obtain meaningful results, and made some analytical inroads toward understanding of species population density in a single-gene system. This paper did not treat simulation techniques explicitly, as did the Arkin and McAdams papers. A later paper by Goss and Peccoud[32] introduced a paradigm for reasoning about stochastic chemical systems in terms of stochastic Petri nets. This, too, did not treat simulation explicitly – but simulation techniques for stochastic Petri nets are well known, and a simulator based on this formalism would likely give results similar to Arkin and McAdams’s⁶.

Random chemistries have been used by Bagley and Farmer [12], who used numerical simulation to demonstrate spontaneous emergence of autocatalytic networks and “metabolisms” in systems driven away from equilibrium by a chemostatic “power supply”. They mention a forthcoming paper with results for synthetic but non-random (polymer) chemistries, in which interaction energies and rate constants are determined using string-matching algorithms. This paper seems never to have appeared; I mention it here because of my independent arrival at the idea of using string matching to produce rate constants for synthetic polymer chemistries.

⁵for instance, most structural genes have a copy number of one per cell

⁶and results produced by derivative simulators, such as the more efficient but functionally equivalent simulators of Gibson and Bruck [29] and of Lyons *et al.* [46]

Chapter 2

Performance Envelope of Chemical Computers

2.1 Model of computation

We assume our computer is made up of a reactor vessel containing a solution (or a gas) of N chemical species in varying concentrations. These will be our signal molecules, or “signals” for short. We assume the signals are chosen from some combinatorial set \mathcal{S} . By combinatorial, we mean all elements of \mathcal{S} can be named by strings of some length, say L , over an alphabet \mathcal{Q} , of size q . Assume further that the fraction of the reactor vessel occupied by signal molecules is bounded above by some limit α , chosen by us as a design parameter of the computer.

Since we are representing signal levels by chemical concentrations, it is important to distinguish between *types* of molecules, and *instances* of molecules (that is, individual molecules). In what follows, *signal* will refer to a type of molecule, and *signal molecule* will refer to an individual instance of such a molecule.

Each signal has one or more interaction sites, which are designed to dock with another interaction site on another signal. This docking serves the function of logical interaction, or gating. For purposes of this argument, we will assume each signal has exactly one interaction site, hence interaction sites can be named by signal names. Similarly to the above, we must distinguish between types – or chemical species – of interaction sites, and individual instances. We will refer to a single instance of an interaction site as a *site*, and to a species of interaction site as a *site class*.

The computation model is as follows: We consider the reactor vessel as a cellular automaton, with a cell size of ΔV and a time step of τ . Each cell is treated as a CSTR (continuously stirred tank reactor). The logical interactions between signals occur within the cells, and information transport is provided by diffusion of the signal molecules to neighboring cells¹. There is a relation between the cell size and the operating frequency, since for larger cells, it will take signal molecules longer to diffuse to adjacent cells. We will treat this relation in more detail later.

To simplify the analysis we assume synchronous operation in time steps of length τ (see note²). At the beginning of each time step, in each cell, each signal is

¹The reason for choosing diffusion-limited transport is as follows: If our computer does not have any fine physical structure to facilitate routing and distinction of signals, then diffusion or active mixing (*e.g.*: stirring) are the only reasonable options available. And, while active mixing is appealing from the point of view of speed, it is incompatible with the Cellular Automaton treatment – or at least trivializes it, since the entire computer is now reduced to a single cell. The analysis in this chapter, it turns out, also covers this trivial “single cell” case – so there is no need to do a separate analysis for the stirred computer.

²This is obviously unrealistic, since chemical reactions are asynchronous, but the difference

assumed to be in one of two states: “present”, corresponding to logic 1, or “absent”, corresponding to logic 0. For a signal s to be “present” in a cell, means that there are on the order of r signal molecules of type s in the cell. Conversely, for a signal s to be “absent” in a cell, means the expected number of signal molecules of type s in the cell is less than $r \cdot \frac{1}{\sigma}$. We’ll call r the “redundancy”; and for lack of a better term, we’ll call σ the signal to noise ratio³. They are both key factors in determining the error rate of the computer, as we shall see later.

During each time step, all signal molecules in a “present” signal diffuse within the cell and sample the other molecules in the cell for matching interaction sites. τ is chosen long enough so that the characteristic diffusion length of the signal molecules in time τ is at least equal to the cell size⁴. This guarantees each signal molecule will sample each other type of signal molecule with high probability on each time step.

Denote a particular site class by s . Instances of other site classes can bind to it with varying affinities. Near the end of each time step, the fractional occupancy of s by other site classes is assumed to be distributed according to the Maxwell-Boltzmann distribution, where the state energies are taken to be the free energies of association of s with each other site class⁵.

At the end of each time step, a point sample in time is taken of all sites in s according to the distribution above. If s is “present”, and if the site is docked with a site of class s' , all is well. But if s is present, and docked with a site of a different site class, a *molecule error* is said to have occurred. If *all* sites of a site class in a cell have molecule errors, a *signal error* is said to have occurred.

We assume a model for the free energies of association as follows: Assume a metric $d(\cdot, \cdot)$ is given on \mathcal{S} , and let s, s' be an intended interaction, as defined above. Now, let $\Delta g_{s,s'}$ be the free energy of interaction between the intended pair of site classes s and s' . We assume that the *difference* in Δg 's between the intended match for s and unintended matches for s is bounded above by a linear

should not affect the validity of the results. See, for example, [51] or [72], for a discussion of functionally simulating synchronous cellular automata using asynchronous cellular automata.

³it is, in fact, closely related to the SNR, as we will see

⁴In fact, a bit larger than the cell size, to allow for signals to diffuse to neighboring cells as well. But this will be discussed later.

⁵Notice this is a completely static treatment; nothing is said regarding reaction rates or equilibration times. This is a strong position to take if we wish to bound the performance of chemical computers *above*: In the bound developed in this chapter, reaction equilibration times are assumed fast compared with diffusion times to neighboring cells. If this is not the case, this can only slow down the effective clock speed of a chemical computer, so the bound still holds, although more loosely.

function of the distance between s' and the unintended match:

$$\forall x \in \mathcal{S}, x \neq s' : \Delta g_{s,x} - \Delta g_{s,s'} \leq \Delta g_1 \cdot d(x, s'),$$

or

$$\forall x \in \mathcal{S}, x \neq s' : \Delta g_{s,x} \leq \Delta g_{s,s'} + \Delta g_1 \cdot d(x, s'), \quad (2.1)$$

where Δg_1 gives the slope of the linear function. Presumably, since (s, s') is the intended match, $\Delta g_{s,s'} < \Delta g_{s,x}$, so Δg_1 is positive.

2.1.1 Definitions

The following chart gives brief descriptions of the important variables needed to analyze chemical computer performance in this model. This instantiation is geared specifically toward biopolymers, since it assumes signals are actually physically represented by strings, and the diffusion constant of signals follows some power-law scaling relation in the length of the strings. Further, in section 2.1.2 below, we assume the metric on signals is the Hamming distance. Of course, in a more general setting, the assumptions cannot be as simple as these.

D_1	Diffusion constant of monomer
L	Length of signal molecules
D_L	Diffusion constant of signal molecules
ΔV	Cell volume
α	Fractional occupancy of signal molecules in reactor
f	Operating frequency
N	Number of signals
q	Base of signal code (size of alphabet)
d_{exp}	Lower bound on expected separation distance of codewords
Δg_1	Interaction energy scaling constant
ϵ	molecule error probability
ε	signal error probability
v_1	effective volume of a monomer
v_L	effective volume of a signal molecule
r	redundancy

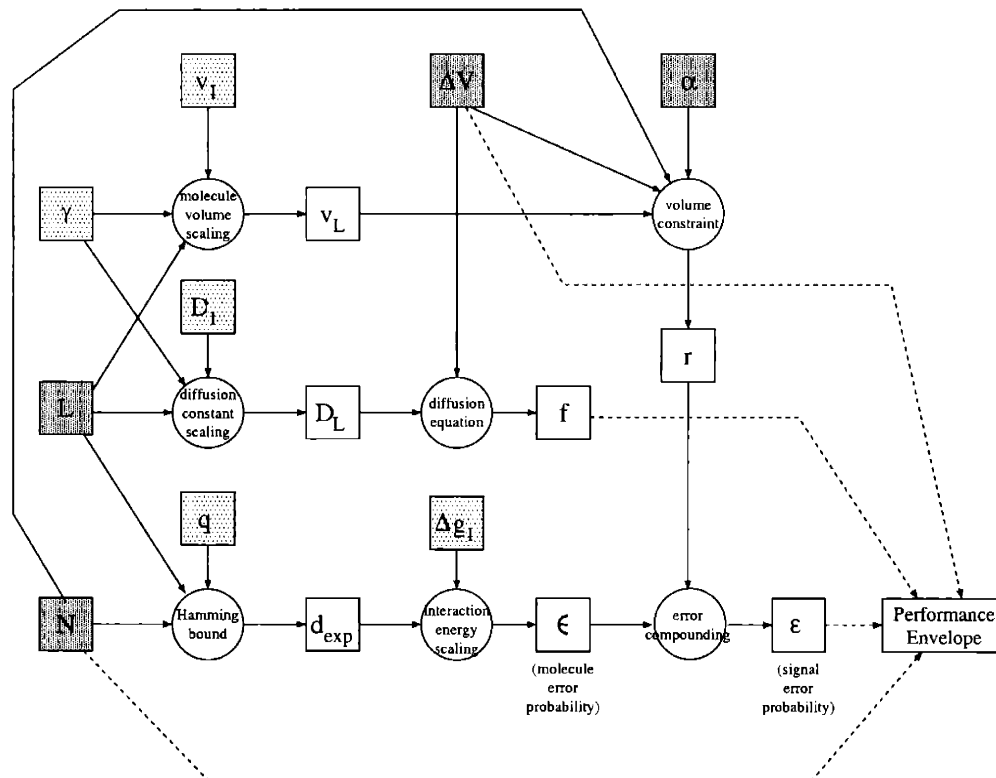


Figure 2.1: Relations and quantities governing performance of a chemical computer. Shaded quantities (α , L , ΔV , and N) are design parameters; lightly shaded quantities (D_1 , q , v_1 , γ , and Δg_1) are constants derived from the particular chemistry used; and unshaded quantities are resultant attributes.

2.1.2 Relations governing performance

Figure 2.1 shows the interrelations between the design variables of a chemical computer, constants derived from the chemistry, and the resultant performance attributes. Quantities are shown in squares; relations are shown as circles. Shaded quantities (α , L , ΔV , and N) are design parameters; lightly shaded quantities (D_1 , q , v_1 , γ , and Δg_1) are constants derived from the particular chemistry used; and unshaded quantities are resultant attributes. Of these attributes, f , ϵ , ΔV , and N itself, will be used to measure the quality of the computer, by defining the limits of its performance envelope. I will discuss each relation in substantially more detail below.

One important introductory note is in order: Very often in the following analy-

ses, I will resort to mean-field methods. That is, often instead of strictly bounding some quantity, I will bound its expectation value. For instance, in the above definitions, I use d_{exp} as a lower bound on the *expected* minimum separation distance of a codeword, instead of strictly bounding the separation distance. While this weakens the result, it allows a tractable analysis of the problem. Future work may replace some of these mean-field analyses with strict bounds.

2.2 The Molecules:

Relations between D_1 , D_L , v_1 , v_L , and L .

We turn first to consideration of basic geometric constraints on our signal molecules: How large are they, and how fast can they move?

2.2.1 Diffusion constant scaling

Let us first consider how the diffusion constant D_L varies with the description length L . Clearly, for any chemistry, the size of \mathcal{S} increases with L . But how fast?

For the biopolymer case, we can explicitly evaluate the scaling law. First, the diffusion constant of any particle can be expressed in terms of its radius. We have, from DeGennes[18]:

$$D = \frac{k_B T}{6\pi\eta_s R_D}, \quad (2.2)$$

where T is temperature, η_s is the solvent viscosity, and R_D is a “characteristic radius” of the molecule. We will assume constant temperature and constant viscosity, so all we need to find a scaling law for D is to find a scaling law for R_D . In the biopolymer case, we can treat two limits explicitly: a compact molecule (corresponding roughly to a globular protein), and a free floating linked chain (corresponding roughly to a strand of DNA).

Case 1: Compact molecule: This case will correspond to the lower limit of R_D , so let us seek only a lower bound on R_D . The smallest value of R_D we could achieve would be if all of the monomers of the molecule were the smallest possible monomer, and the monomers were perfectly packed. In this case, the minimum volume is Lv_{min} , where v_{min} is the volume of the smallest monomer. So the minimum R_D would occur if the molecule were a perfect sphere with this volume. Letting the radius of such a sphere be denoted R_{min} , we have:

$$R_D \geq R_{min} = \left(\frac{3Lv_{min}}{4\pi} \right)^{1/3}.$$

Or, in the parlance of computational complexity theory,

$$R_D = \Omega(L^{1/3}). \quad (2.3)$$

Case 2: Free floating linked chain: We can model a free-floating linked heteropolymer as a non-self-intersecting random walk. Flory [24] determines a scaling law for the characteristic radius of such walks. He defines the characteristic radius as the expected distance between the position of a randomly chosen link in the chain, and the center of mass:

$$R_D = E[|\vec{r} - E[\vec{r}]|],$$

where \vec{r} is a random variable giving the position of a link. Flory concludes under these assumptions that the characteristic radius scales as:

$$R_D = \Omega(L^{3/5}). \quad (2.4)$$

Discussion: At one end of the spectrum we have compact molecules, whose radius scales as $L^{1/3}$. The radius of free floating linked chain molecules scales as $L^{3/5}$. That is, it increases faster with increasing L . We could imagine molecule classes with even faster increase of the radius, such as rigid rods for instance. Presumably their radius would scale essentially as L^1 .

Clearly all other classes of signals in which the mass of the molecule scales as L will have radius scaling between these extremes. We have not treated classes in which the mass does not scale with the description length. For instance, presumably a longer string is needed to describe a branched polymer of size n , than a linear polymer of the same size.

But let us stick to the linear description length molecules for now. For these let us say that each class of molecule has some scaling exponent γ describing the scaling of R_D with respect to L :

$$R_D = \Omega(L^\gamma), \quad (2.5)$$

where γ is between $1/3$ and 1 , depending on the geometry of our signal molecules. We will see later that although γ affects the value of the performance bound, it does not affect the boundedness property itself.

We combine equations 2.2 and 2.5 to deduce a scaling law for D_L :

$$D_L = D_1 L^{-\gamma}. \quad (2.6)$$

2.2.2 Volume scaling

In subsequent sections, we will also need to have an idea of the *volume* of the reaction mixture occupied by each molecule. We already have most of the answer

2.2. THE MOLECULES:
RELATIONS BETWEEN D_1 , D_L , V_1 , V_L , AND L .

above in equation 2.5: The occupied volume of the molecule scales as the cube of the radius. So, if the radius scales as L^γ , then the volume must scale as $L^{3\gamma}$:

$$v_L = v_1 L^{3\gamma}. \quad (2.7)$$

We will use this occupied volume later to calculate the redundancy of the system from the cell size, the number of signals, and the fractional occupancy of the reactor vessel.

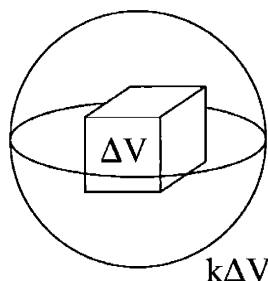


Figure 2.2: The critical diffusion volume, V_{crit}

2.3 The Transport Mechanism: Relation between D_L , ΔV , and f .

We now explore the relation between D_L , ΔV , and f . Recall D_L is the diffusion constant of the signals, and f is the maximum frequency of operation (that is, the inverse of the cycle time τ).

Let V_{crit} be the critical volume associated with each signal molecule, through which it must diffuse in order to have sampled its neighboring signal molecules with high probability. Put another way: consider the total probability density of the presence function of the signal molecules of a single signal s , call it $f_s(\vec{x}; t)$. This probability density may be initially concentrated around some set of discrete initial positions, but will always evolve to fill the volume of the reactor cell uniformly. If the molecules are initially almost uniformly distributed throughout the cell and each particle undergoes an independent diffusion process, then when the characteristic width of each diffusion process is at least the characteristic radius associated with the critical volume, then $f(\vec{x}; t)$ will be approximately uniform over the critical volume. The time at which this occurs is our definition of τ , and the characteristic volume through which each individual molecule must diffuse, we will call V_{crit} . This in turn gives us a characteristic diffusion length, s_{crit} , which varies as the cube root of V_{crit} . In our case, since we are considering the reactor as a cellular automaton, we will take as V_{crit} the volume of a cell plus its “nearest neighbors”. And further, let us say this volume is a small multiple of ΔV , say $k\Delta V$ (see figure 2.2).

We use the diffusion equation to reason about the relation between D_L , ΔV , and f . Using

$$D\tau \cong \Delta X^2$$

2.3. THE TRANSPORT MECHANISM:
RELATION BETWEEN D_L , ΔV , AND F .

and putting $D = D_L$, $\tau = 1/f$, and $\Delta X = s_{crit} = V_{crit}^{1/3} = (k\Delta V)^{1/3}$, we obtain

$$f = D_L V_{crit}^{-2/3} \tag{2.8}$$

$$= D_L (k\Delta V)^{-2/3} \tag{2.9}$$

This relation gives us the maximum speed at which we can expect our computer to operate, in terms of the diffusion constant, cell size, and neighborhood size.

2.4 Geometry of the Reactor Vessel: Relation between α , v_L , ΔV , N , and r

The volume of each cell is ΔV , and the allowable fractional occupancy by signal molecules is α . So the maximum allowable volume of signal molecules in each cell is at most $\alpha\Delta V$: $v_{max} \leq \alpha\Delta V$. On the other hand, the maximum allowable volume of signal molecules is given by the number of species, times the redundancy, times the volume of a single signal molecule: $v_{max} = Nrv_L$. Combining these two, we have

$$Nrv_L \leq \alpha\Delta V.$$

And, rearranging, we have

$$r \leq \frac{\alpha\Delta V}{Nv_L}. \quad (2.10)$$

A note is in order: For our diffusion relation to hold, we must be working at low densities. In other words, $\alpha \ll 1$. Later, when we come to detailed analysis of proposed biochemical computation schemes, we will choose a suggested α from typical DNA or protein concentrations given in the literature.

2.5 Limitations on the Code: Relation between N , L , and d_{exp} .

2.5.1 The approach

We will treat the relation between N , L , and d_{exp} using an information theoretic approach. We will show a bound on d_{exp} – the minimum expected separation distance (in the Hamming metric) of any code in \mathcal{Q}^L – in terms of N , the number of codewords, and L , the string length. This bound is based on the Hamming bound, but has been relaxed in favor of better analytic tractability.

Let \mathcal{S} denote the space of strings of length L over an alphabet \mathcal{Q} : $\mathcal{S} \equiv \mathcal{Q}^L$. We wish to choose a *code* in \mathcal{S} : $C \subset \mathcal{S}$, with “nice” properties. In particular, we would like to pick the N strings in C such that they are as far apart as possible. We will first show that given N , we may choose an optimal code in which the codewords are at least some distance d_{opt} apart (as in figure 2.3(a)). Then, we will show that for *any* code of size N , d_{exp} – the *expected* Hamming distance between a codeword and its nearest neighbor – is linearly related to d_{opt} . This means that even though we might be able to choose a code in which *some* codewords are very far from their neighbors, we can’t find one in which the *expected* distance of a codeword from its neighbors is larger than a value linearly related to d_{opt} .

2.5.2 Definitions

definition: Let w be a codeword in a code C , and let $d(\cdot, \cdot)$ be a metric on codewords. The *separation distance* $\underline{d}(w)$ of codeword w is defined as the smallest distance from w to any other codeword:

$$\underline{d}(w) \equiv \inf_{x \in C, x \neq w} d(x, w).$$

definition: The *minimum separation distance* of a code C is the smallest separation distance of any codeword in C :

$$d_{\min}(C) \equiv \min_{w \in C} \underline{d}(w).$$

definition: The *expected separation distance* of a code C is the expected separation distance of a uniformly chosen codeword in C :

$$d_{\text{exp}}(C) \equiv E_{w \in C} [\underline{d}(w)]. \tag{2.11}$$

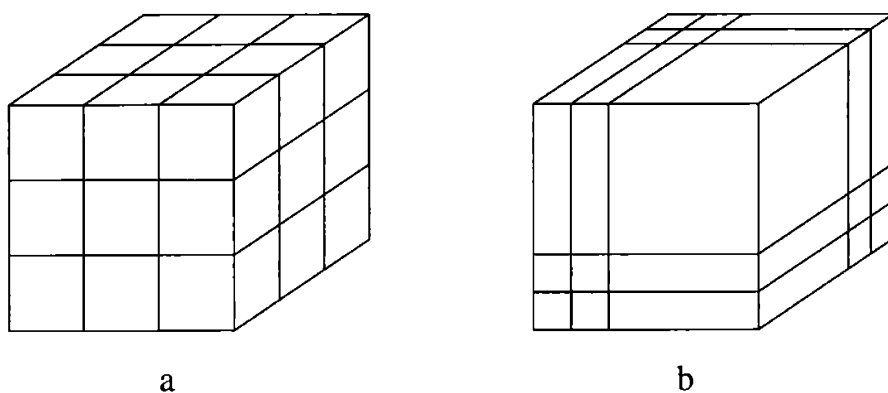


Figure 2.3: A pair of codes. *Note:* the metric used in the figure is \mathcal{R}^3 , *not* the Hamming distance; the figure is *schematic only*.

- (a) an “optimal” code, in which $N = 27$. The “cubelets” are of side 1. $d_{\text{opt}} = d_{\text{exp}} = 1$.
- (b) another code with $N = 27$. In this case, \underline{d} of the “big” block is $\frac{5}{4}$, but $d_{\text{exp}} \approx 0.53$, since most codewords have $\underline{d} = \frac{1}{2}$.

2.5.3 A restatement of the problem

Let's restate our problem as follows:

conditions: Given a distance δ and a pairing relation $\{(w, w')\}$, we seek a maximal set of strings $C \equiv \{w_1, w_2, \dots, w_N\} \subset \mathcal{S}$ with the following properties:

1. $w \in C$ iff $\bar{w} \in C$
2. there exists $\delta > 0$ such that, for all $w, \bar{w} \in C$, $\min_{x \neq w} [d(x, w)] = \underline{d}(w) \geq \delta$
and $\min_{x \neq \bar{w}} [d(x, \bar{w})] = \underline{d}(\bar{w}) \geq \delta$

The first condition says that for each site class, its pair is also a site class. The second says there exists a nonzero lower bound on the Hamming distance between the "correct" match for a signal and any "incorrect" match for the same signal; that is, the distance between a correct match for a signal and any incorrect matches will be at least δ .

Referring to the definition of d_{\min} , above, we see that condition (2) is equivalent to:

$$d_{\min} \geq \delta.$$

question: How large can $N = \|C\|$ possibly be, given a specified value of δ ? Or conversely, given N , how large can δ be? In other words, given N , what is d_{\min} ?

2.5.4 An upper bound on d_{exp}

The Hamming bound is:

$$NV(q, L, \frac{d_{\min} - 1}{2}) \leq q^L, \tag{2.12}$$

where $V(q, n, r)$ denotes the volume of a "sphere" of radius r in an n dimensional Hamming space over q symbols. The classic expression for V is

$$V(q, n, r) \equiv \sum_{i=0}^r \binom{n}{i} (q-1)^i.$$

But this is difficult to work with because there is no general closed form expression for it. Instead, we will work with a modified, looser version of the Hamming bound by using a lower bound on V :

Define $\underline{V}(q, n, r) \equiv q^r$. Then we have the following

lemma: $\forall q, n \geq r : \underline{V}(q, n, r) \leq V(q, n, r)$

proof:

$$\begin{aligned} \underline{V}(q, n, r) &= q^r \\ &= [(q-1) + 1]^r \\ &= \sum_{i=0}^r \binom{r}{i} (q-1)^i \\ &\leq \sum_{i=0}^r \binom{n}{i} (q-1)^i \\ &= V(q, n, r) \end{aligned}$$

So we can write the following looser bound for our code:

$$N \underline{V}(q, L, \frac{d_{\min} - 1}{2}) = N q^{(d_{\min} - 1)/2} \leq q^L \quad (2.13)$$

$$\begin{aligned} N &\leq q^{L - (d_{\min} - 1)/2} \\ \log_q N &\leq L - \frac{d_{\min} - 1}{2} \\ d_{\min} &\leq 2(L - \log_q N) + 1 \end{aligned} \quad (2.14)$$

So, if we must have at least N signals of length L , this gives an upper bound on the minimum distance between correct and incorrect matches. Conversely, if we require all mismatched signals to have distance at least δ from any matched signal, we cannot choose more than $q^{L - (\delta - 1)/2}$ signals.

Note that this says nothing about the separation of *all* codewords in an arbitrary code of size N . In fact, in general it is possible to choose a code of size N with minimum separation distance d_{\min} as given above, but with *some* codewords having a very large separation distance⁶. The best we can hope to do in general is to bound the expectation value of the separation distance, given a randomly chosen codeword. This is a bit tricky, but can be done as follows:

⁶A proof of this by example is easily constructed. For instance, in Z_l^2 using the R^∞ (Manhattan) metric, we may choose a pathological combination of N and l such that for this N , the minimum distance is *almost* $d + 1$, but not quite. So the minimum distance is d , but we can pack all the codewords tightly in one portion of the space with distance d separating them, and leave room for a codeword that is far from all other codewords. See also figure 2.3 for a more intuitive, schematic example.

2.5. LIMITATIONS ON THE CODE:
RELATION BETWEEN N , L , AND D_{EXP} .

definition: The *optimal spacing* d_{opt} of a space \mathcal{S} and a code-size N is the maximum possible minimum separation distance over all codes in \mathcal{S} of size N :

$$d_{\text{opt}}(\mathcal{S}, N) \equiv \max_{C \subset \mathcal{S}, |C|=N} d_{\text{min}}(C).$$

Since $d_{\text{opt}}(\mathcal{S}, N) = d_{\text{min}}(C)$ for some code C , we have (from 2.14):

$$d_{\text{opt}}(\mathcal{S}, N) \leq 2(L - \log_q N) + 1. \quad (2.15)$$

Now consider the following

lemma: If the optimal spacing of N and \mathcal{S} is d_{opt} , then for *any* code C in \mathcal{S} of size N ,

$$E[\underline{d}(w)] \equiv d_{\text{exp}}(C) \leq 2(d_{\text{opt}}(\mathcal{S}, N) + 1).$$

proof: For the proof, see appendix B.

Applying the lemma to inequality 2.15, we conclude:

$$d_{\text{exp}}(C) \leq 2[2(L - \log_q N) + 1] + 1 \quad (2.16)$$

$$= 4(L - \log_q N) + 3. \quad (2.17)$$

This relation has the desired form: it puts an upper bound on the expected distance between mismatched codewords, in terms of the length of the codewords, and how many codewords we have. As one might expect, in our model of chemical computation, this in turn allows us to put a lower bound on the resultant error probability. We discuss this further in the next section.

2.6 The Gating Process: Relation between Δg_1 , d_{exp} , and $\langle \epsilon \rangle$

Let's now turn to the heart of the matter: the process of chemical interactions taking place within the reactor vessel. To a first approximation, each interaction takes place with a probability determined by the concentrations of the reactants; and the Gibbs free energy of the intermediate reaction complex, ΔG . Presumably our computer is designed so that signal molecules which are *intended* to interact, do so with high probability, and signal molecules *not* intended to interact, do so with low probability. This means we should design our intended interactions so that they have a low ΔG , and try and keep things such that unintended interactions have a high ΔG .

Intuitively, the more signals we pack into a small chemical conformation space, the smaller the ΔG of unintended interactions will be, hence the more likely errors will be. We can formalize this with the help of the bound on d_{exp} developed in the preceding section, combined with the concept of the linear interaction energy bound, introduced at the beginning of this chapter in equation 2.1, repeated here for reference:

$$\forall x \in \mathcal{S}, x \neq s' : \Delta g_{s,x} \leq \Delta g_{s,s'} + \Delta g_1 \cdot d(x, s'). \quad (2.18)$$

Recall that here, (s, s') is an intended interaction; (s, x) is an unintended interaction; and Δg_1 is a positive number, giving the dependency of $\Delta g_{s,x}$ on $d(x, s')$. Assuming this relation holds, we can use the Maxwell-Boltzmann distribution and some simple statistical reasoning to lower bound ϵ , the probability of a molecule error.

Let us reuse the notation of equation 2.18: $\Delta g_{s_1, s_2}$ will denote the free energy of association of site s_1 with site s_2 . Now consider a site s , its intended docking mate s' , and another site x which is at Hamming distance $d(x, s')$ from the intended docking mate s' . We wish to evaluate the probability of crosstalk; that is, if s' is absent and x is present, what is the probability that a site instance of s will be occupied by a site instance of x ? First, consider the case where both s' and x are present. By the Maxwell-Boltzmann distribution, we have the probability of occupancy of s by s' and x , respectively, as:

$$\begin{aligned} \Pr[ss'] &= \frac{1}{Z} \exp(-\Delta g_{s,s'}/KT) \\ \Pr[sx] &= \frac{1}{Z} \exp(-\Delta g_{s,x}/KT) \end{aligned}$$

2.6. THE GATING PROCESS:
RELATION BETWEEN ΔG_1 , D_{EXP} , AND $\langle \varepsilon \rangle$

We can explicitly compute the conditional probability that s is occupied by x , given that it is occupied by either s' or x :

$$\begin{aligned} \Pr[sx|sx \cup ss'] &= \frac{\Pr[sx]}{\Pr[sx] + \Pr[ss']} \\ &= \frac{\exp(-\Delta g_{s,x}/KT)}{\exp(-\Delta g_{s,s'}/KT) + \exp(-\Delta g_{s,x}/KT)} \\ &> \frac{\exp(-\Delta g_{s,x}/KT)}{2 \exp(-\Delta g_{s,s'}/KT)} \\ &= \frac{1}{2} \exp[(\Delta g_{s,s'} - \Delta g_{s,x})/KT] \end{aligned}$$

But $\Delta g_{s,x} - \Delta g_{s,s'} \leq \Delta g_1 d(x, s')$, from inequality 2.18. So $\Delta g_{s,s'} - \Delta g_{s,x} \geq -\Delta g_1 d(x, s')$. Since $\exp(\cdot)$ is monotonic increasing, we have

$$\Pr[sx|sx \cup ss'] > \frac{1}{2} \exp(-\Delta g_1 d(x, s')/KT).$$

Now, this whole discussion has been for the state in which both x and s' are present. For the state in which only x is present, the probability that an instance of x is bound to an instance of s is even higher. So, when signal s' is absent, we have a lower bound on the probability that it will *appear* that s' is *present* because of crosstalk from a signal x :

$$\epsilon_{s,x} > \frac{1}{2} \exp(-\Delta g_1 d(x, s')/KT).$$

We wish to bound the probability of crosstalk from *any* signal, not just one specific signal. We can make a very loose bound, by simply observing that the probability of crosstalk overall is at least as great as the probability from any specific signal. To make this bound as tight as possible, let us pick a signal as close to s' as possible:

$$\begin{aligned} \epsilon_s &> \frac{1}{2} \exp(-\Delta g_1 \underline{d}(s')/KT) \\ &= \exp(-\Delta g_1 \underline{d}(s')/KT - \ln 2) \end{aligned}$$

where recall $\underline{d}(w)$ is the minimum separation distance of the codeword w .

This is the probability of a *molecule error*. Now let us bound the probability of a *signal error* – that is, the probability that *all* instances of a site class are erroneously bound. There are r instances of site s (recall r is the redundancy). If we take the occurrence of a molecule error at each site instance to be independent, we conclude the probability of a signal error is the r th power of the probability of a molecule error:

$$\begin{aligned}\varepsilon_s &= \epsilon_s^r \\ &> \exp[-r[\Delta g_1 \underline{d}(s')/KT + \ln 2]].\end{aligned}$$

Now, to get anywhere from here, we must resort to a mean-field approximation. Namely, we will replace ε_s by its expectation value over all possible values of s :

$$\begin{aligned}\langle \varepsilon \rangle &\equiv E_s[\varepsilon_s] \\ &> E_s[\exp[-r[\Delta g_1 \underline{d}(s')/KT + \ln 2]]] \\ &> \exp(E_s[-r[\Delta g_1 \underline{d}(s')/KT + \ln 2]]),\end{aligned}$$

where the last step follows from the convexity of $\exp(\cdot)$.

But since r , Δg_1 , K , and T are constants,

$$\begin{aligned}E[-r[\Delta g_1 \underline{d}(s')/KT + \ln 2]] &= -r[(\Delta g_1/KT)E[\underline{d}(s')] + \ln 2] \\ &= -r[(\Delta g_1/KT)d_{\text{exp}} + \ln 2].\end{aligned}$$

So,

$$\langle \varepsilon \rangle > \exp[-r[(\Delta g_1/KT)d_{\text{exp}} + \ln 2]] \tag{2.19}$$

$$\tag{2.20}$$

This relation gives the predicted dependence of $\langle \varepsilon \rangle$ on d_{exp} : The expected probability of error goes down as d_{exp} increases, and as the redundancy r increases.

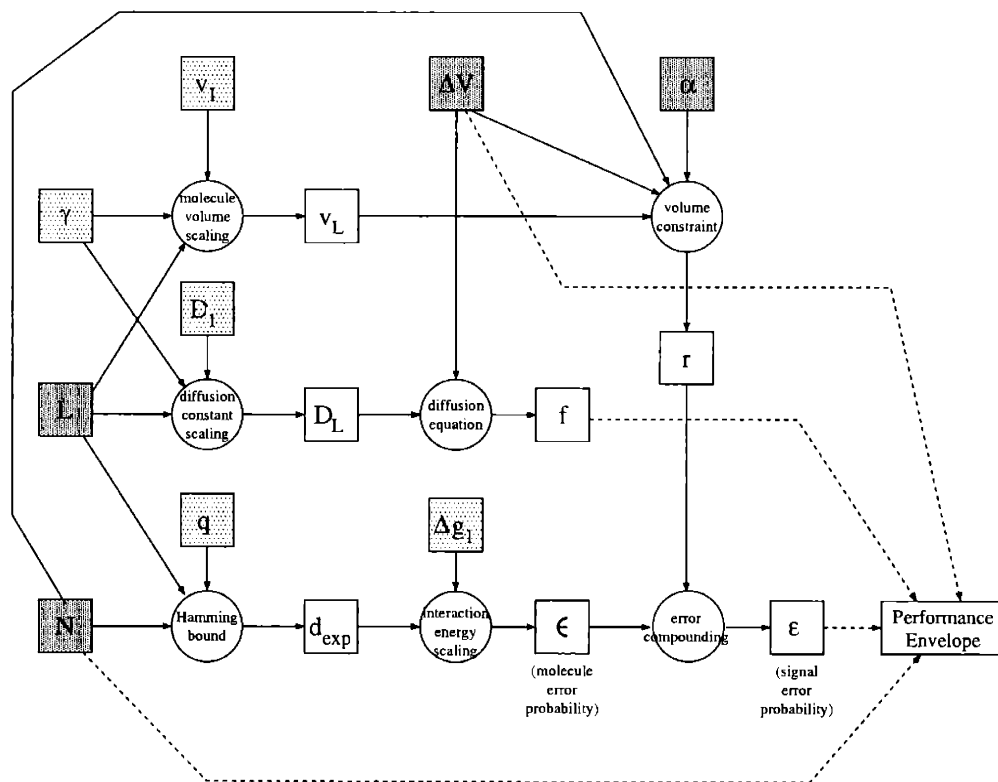


Figure 2.4: Relations and quantities governing performance of a chemical computer.

2.7 Discussion

Recall figure 2.1, given at the beginning of this chapter, which shows the parameters of the chemistry, the other design parameters of the computer, and the relations between them which give rise to the resultant performance attributes. This figure is repeated here for reference, as figure 2.4.

Let us review the previous sections of this chapter, and collect the relevant relations here in one place, for easy reference:

$$D_L = D_1 L^{-\gamma} \quad (2.21)$$

$$v_L = v_1 L^{3\gamma} \quad (2.22)$$

$$f \leq D_L (k\Delta V)^{-2/3} \quad (2.23)$$

$$r \leq \frac{\alpha \Delta V}{N v_L} \quad (2.24)$$

$$d_{\text{exp}} \leq 4(L - \log_q N) + 3 \quad (2.25)$$

$$\langle \varepsilon \rangle > \exp[-r[(\Delta g_1/KT)d_{\text{exp}} + \ln 2]] \quad (2.26)$$

Now let's see what we can deduce about the performance parameters f and $\langle \varepsilon \rangle$, by combining these relations.

First f , since it is simpler. Substituting 2.21 into 2.23, we have

$$f \leq D_1 L^{-\gamma} (k\Delta V)^{-2/3}. \quad (2.27)$$

Let us call this relation the “final f ” relation, since it gives a bound on the frequency of the computer, in terms of the design parameters.

Now let's look at $\langle \varepsilon \rangle$. We start by substituting 2.24 into 2.26, to obtain

$$\langle \varepsilon \rangle > \exp\left[-\frac{\alpha\Delta V}{Nv_L} \cdot \left[\frac{\Delta g_1}{KT} \cdot d_{\text{exp}} + \ln 2\right]\right]. \quad (2.28)$$

Then we can further substitute both 2.22 and 2.25 into this, to obtain

$$\langle \varepsilon \rangle > \exp\left[-\frac{\alpha\Delta V}{Nv_1 L^{3\gamma}} \cdot \left[\frac{\Delta g_1}{KT}(4(L - \log_q N) + 3) + \ln 2\right]\right]. \quad (2.29)$$

Let us call this the “final $\langle \varepsilon \rangle$ ” relation, since it gives a bound on the error of the computer, in terms of the design parameters.

Let's do a bunch of sanity checks, to see if this all sounds reasonable:

Sanity checks on f :

- f goes as the native diffusion constant D_1 .
- f goes inversely as the length L .
 - for compact molecules, f goes as $L^{-1/3}$
 - for stringy molecules, f goes as $L^{-3/5}$
 - for a given L , f (the speed) is lower for stringy than for compact molecules
- f goes inversely as ΔV : $\Delta V^{-2/3}$ – for larger cells, the computer is slower

Sanity checks on $\langle \varepsilon \rangle$:

- $\langle \varepsilon \rangle$ goes inversely as α : $e^{-\alpha}$
- $\langle \varepsilon \rangle$ goes inversely as ΔV : $e^{-\Delta V}$
- $\langle \varepsilon \rangle$ goes as N : $e^{-\frac{1}{N} + \log_q N} = N e^{\frac{1}{\log q}} e^{-\frac{1}{N}}$
- $\langle \varepsilon \rangle$ goes as L : $e^{-L^{-3\gamma} - L^{1-3\gamma}}$
- for compact signal molecules, $\langle \varepsilon \rangle$ goes up *less* rapidly with L than for stringy signal molecules

Shape of the performance envelope

- varying N :
It is not surprising that $\langle \varepsilon \rangle$ increases with N , all other factors being equal. This is for two reasons: The polynomial term is due to the decrease of d_{exp} with increasing number of signals; and the exponential term is due to the decrease in chemical redundancy with increasing number of signals. f is not affected by N in this analysis.
- varying L :
One of the most interesting things about this analysis is the asymptotic dependence of $\langle \varepsilon \rangle$ on L . One might think that increasing L would decrease $\langle \varepsilon \rangle$. And indeed this may be true, up to some critical value of L . But notice that if L gets very large, $\langle \varepsilon \rangle$ will eventually be bound by the lower bound given by equation 2.29, hence $\langle \varepsilon \rangle$ will eventually start increasing. Why is this? Closer inspection of the form and derivation of equation 2.29 leads us to conclude that the interaction energy portion of the error term can continue getting better without bound, but the *effective volume* of the molecules as they get larger, eventually critically limits the chemical redundancy of the system.
 f goes down with L , since larger molecules move more slowly.
In summary, *both* performance parameters (f and $\langle \varepsilon \rangle$) get asymptotically worse as L gets large. So, all other things being equal, there is some intermediate value of L at which optimal performance is obtained.
- varying ΔV :
 $\langle \varepsilon \rangle$ goes inversely as ΔV , since for larger ΔV , the chemical redundancy is

higher. All other things being equal, this would make us want to choose a larger ΔV to get better performance.

But, f goes inversely as ΔV , since if signals have to travel farther, data movement is slower, and the operating speed suffers.

- varying α :
 - variations in α are not asymptotically important, since $\langle \varepsilon \rangle$ goes as $e^{-\alpha}$, and α has the limited range $(0, 1)$. In fact, our bounds on the performance parameters $\langle \varepsilon \rangle$ and f are not even particularly tight for α near the ends of its range, for at least the following reasons:
 - If α approaches zero, the redundancy approaches zero also. But our analysis of signal error probability assumed $r \geq 1$. In fact, if $r \ll 1$, this would mean that signal molecules are very sparse, and that we should be treating the system as a single continuous stirred tank reactor with low concentrations. In this case the chemical concentrations are stochastic [30], evolving according to an infinite-state Markov process described by the Kolmogorov equations for the set of possible reactions [58].
 - If α approaches 1, then our approximation that D_1 – the diffusion constant of monomers – is a constant independent of α becomes a very bad approximation indeed. If α approaches 1, then there is little space left for solvent, the solution becomes supersaturated, and our signal molecules begin to precipitate. In the extreme case, the system is solid, and D_1 is zero.
- varying q :
 - q is not actually an “independent” parameter, according to figure 2.4, but it is interesting to speculate whether high- or low- radix codes would be “better” for chemical computation. Referring back to equation 2.29, we see our bound on $\langle \varepsilon \rangle$ goes as $\exp[-\Delta g_1(1 - \frac{1}{\log q})]$. This would say that larger q and larger Δg_1 are both better for attaining lower error rates; and this is consistent with intuition. The problem is that they are both parameters of the chemistry being used; they cannot be chosen independently. And with a little thought, it is certainly reasonable to believe that chemistries with larger values of q tend to have smaller values for Δg_1 .⁷

⁷Such an analysis is outside the scope of this discussion. But an argument would proceed

along lines similar to the information-theoretic argument regarding the bound on d_{exp} of appendix B, followed by the developments in sections 2.5 and 2.6: You cannot fit an unboundedly large amount of information into a fixed-size configuration space.

CHAPTER 2. PERFORMANCE ENVELOPE OF CHEMICAL COMPUTERS

Chapter 3

Biopolymers and DNA Computation

3.1 Applications: DNA Computers

Typical proposals for DNA computation found in the literature use diffusion for signal transport, and hybridization of matching regions of single stranded DNA (ssDNA) for signal recognition and logical interaction [3, 45, 68, 16, 61, 66, 81]. In this section we will investigate the performance capabilities of such computers by putting some reasonable numbers to our performance bound.

Bear in mind during this discussion that this is a *very* coarse numerical approximation – The performance bound itself is very coarse, so doing better numerically would be wasted effort. Additionally, trying to pinpoint performance limits to high precision would be outside the scope of this work.

3.1.1 Chemical and mechanical properties of DNA

We begin by *briefly* summarizing the chemical and mechanical properties of DNA relevant to our analysis.

Single stranded DNA is a heteropolymer, composed of a glycoposphate backbone with a single nitrogenous base covalently bonded to each sugar of the backbone. The bases found in “garden variety” DNA are Adenine, Thymine, Cytosine, and Guanine, denoted A , T , C , G , respectively. To a first approximation, the bases may be attached in any number, in any sequence. Hence ssDNA molecules of length L can be uniquely denoted by strings in $\{A, T, C, G\}^L$.

Two regions of such ssDNA may *hybridize* by “zipping up” into a countercsense double helix, in which bases from one ssDNA are hydrogen bonded to their corresponding bases from the other ssDNA. Due to the geometry of the specific bases, this interaction is favored if A and a T are found at corresponding positions, or if G and C are found at corresponding positions. The pairs (A, T) and (C, G) are called Watson-Crick pairs. The eight other pairs (*i.e.*: AA , TT , CC , GG , AC , AG , TC , TG) are called *mismatches*.

To a coarse approximation, good enough for our purposes, the interaction energy of two matched countercsense ssDNAs of length L may be modeled as

$$\Delta g = \Delta g_0 + L\Delta g_{\text{match}}, \quad (3.1)$$

where L denotes the number of Watson-Crick pairs, and the Δg 's are constants. Δg_0 is known as an “initiation” energy; it can be thought of as representing the free energy of a (hypothetical) hybridized string of length 0. Approximate values

of the Δg 's derived from SantaLucia [40] are shown in the table^{1,2,3}:

Δg_0	$2.03 \frac{\text{kcal}}{\text{mol}}$
Δg_{match}	$-1.67 \frac{\text{kcal}}{\text{mol}}$

A more sophisticated energy model could be constructed by explicitly taking into account the difference in Δg introduced by *mismatches* between the two strands, rather than just counting a mismatch as contributing zero to Δg . See, for example, the very thorough papers by Allawi [5, 6, 7] and SantaLucia [40], detailing the Δg contribution to DNA hybridization energy for various types of mismatch.

The average mass of a monomer of ssDNA is approximately $305 \frac{\text{g}}{\text{mol}}$, assuming an approximately even mixture of *A*, *T*, *C*, and *G* (see note ⁴).

We will treat the average excluded solute volume per monomer very approximately, by considering a hybridized pair of ssDNAs of length L to assume a cylindrical shape, of diameter 2nm , and with a length of $(0.34 \times L)\text{nm}$. We divide the total volume of this cylinder by $2L$, to obtain:

$$v_1 \approx 0.53\text{nm}^3.$$

As far as a value for γ goes, we will assume very simplistically that ssDNAs assume the form of a self-avoiding random walk, hence giving a γ of $3/5$, as discussed in section 2.2.1, and in [24].

We may calculate D_1 in at least two different ways:

1. *de novo*, by the formula given in DeGennes[18] and repeated in section 2.2:

$$D_1 = \frac{k_B T}{6\pi\eta_s R_D}. \quad (3.2)$$

2. By obtaining D_L for some value of L , and scaling it down using relation 2.6:

$$D_L = D_1 L^{-\gamma}.$$

¹ Δg values are given at pH 7, 1.0 M[Na^+], 25° C

² Δg_0 was estimated by computing ΔG_{37}° for initiation with terminus (*G,C*), and ΔG_{37}° for initiation with terminus (*A,T*), taking their mean, and doubling it (since oligomers have two ends)

³ Δg_{match} was estimated by computing ΔG_{37}° for each possible nearest neighbor (NN) propagation sequence, and taking the mean (weighted by their frequency in a uniform IID nucleotide sequence)

⁴the spread in masses between monomer types is small, hence the average mass is relatively independent of the mixture of types, at least for coarse calculations

de novo calculation of D_1 : To get an approximate value, we first assume a monomer looks like a spherical “blob”, for diffusion purposes. So

$$v_1 = \frac{4}{3}\pi R_D^3,$$

hence

$$R_D = \left[\frac{3v_1}{4\pi} \right]^{1/3}$$

and so

$$D_1 = \frac{k_B T}{6\pi\eta_s} \left[\frac{4\pi}{3v_1} \right]^{1/3}.$$

For our situation, we will use room temperature and water as a solvent ($\eta_s \approx 0.0009 \frac{\text{N}\cdot\text{s}}{\text{m}^2}$). This gives

$$\begin{aligned} D_1 &\approx \frac{(1.4 \cdot 10^{-23} \frac{\text{J}}{\text{K}})(300\text{K})}{6\pi(9 \cdot 10^{-4} \frac{\text{N}\cdot\text{s}}{\text{m}^2})} \cdot \left[\frac{4\pi}{3 \cdot (0.53\text{nm}^3)} \right]^{1/3} \\ &\approx (2.5 \cdot 10^{-19} \frac{\text{m}^3}{\text{s}}) \left(\frac{1}{0.5\text{nm}} \right) \\ &\approx 5 \cdot 10^{-10} \frac{\text{m}^2}{\text{s}} \\ &= 500 \frac{\mu\text{m}^2}{\text{s}} \end{aligned}$$

scaling calculation of D_1 : I have not calculated D_1 using this technique.

3.1.2 General characteristics of proposed *in vitro* DNA computers

Typical proposals for *in vitro* DNA computation found in the literature use diffusion for signal transport, and hybridization of matching regions of single stranded DNA (ssDNA) for signal recognition and logical interaction [3, 45, 68, 16, 61, 66, 81].

In all these schemes, we may denote interaction sites of ssDNA as strings over the DNA nucleotide alphabet, $\mathcal{Q} = \{A, T, G, C\}$. If the hybridization regions are of length L , then our signals may be named by strings in $\mathcal{S} = \mathcal{Q}^L$. For any $w_1, w_2 \in \mathcal{S}$, let $d(w_1, w_2)$ denote the Hamming distance between the strings w_1 and

w_2 . $d(\cdot, \cdot)$ is a metric on \mathcal{S} . For any string w , let \bar{w} denote the reverse complement of w .

The problem of choosing sequences for the signaling regions has been well studied [3, 20, 67, 83, 35, 28]. We summarize (in highly simplified form) as follows:

Consider a set of strings $C \equiv \{w_1, w_2, \dots, w_N\} \subset \mathcal{S}$ with the following properties:

1. $w \in C$ iff $\bar{w} \in C$
2. for any $w, \bar{w} \in C$, $E_w[d(w)] \geq \delta$ and $E_{\bar{w}}[d(\bar{w})] \geq \delta$

The first condition says that for each signal, its hybridization match is also a signal. The second says the expected distance between the “correct” match for a signal and any “incorrect” match for the same signal – that is, the expected number of mismatched base pairs for incorrect signal matches – is at least δ .

The problem at hand is, we wish to choose such a set C , with as large a value of δ as possible.

If we recall the definition of d_{exp} given in equation 2.11, we may restate condition (2) more simply and clearly as $\delta \leq d_{\text{exp}}$. In fact, since d_{exp} is the largest value attainable for δ , we may as well choose our signal set such that $\delta = d_{\text{exp}}$. This will give us the largest possible expected mismatch distance for a signal set of size N .

3.1.3 Linear bound on mismatch energy

Now, let’s look at the difference in interaction energies between a correct signal match and an incorrect signal match, and argue that it obeys the linear mismatch energy bound, as shown in equation 2.1 (later repeated as equation 2.18). Let w be a signal, and let \bar{w} be its intended match. In our DNA model, \bar{w} is the reverse complement of w . So, by equation 3.1, the interaction energy is

$$\Delta g_{w, \bar{w}} = \Delta g_0 + L \Delta g_{\text{match}}, \quad (3.3)$$

since w and \bar{w} match at all indices. Now, consider any mismatch for w , call it x , which is at distance δ from \bar{w} : $d(\bar{w}, x) = \delta$.

We have:

$$\Delta g_{x, w} = \Delta g_0 + n_{\text{match}}(x, w) \Delta g_{\text{match}}, \quad (3.4)$$

where $n_{\text{match}}(x, w)$ is the number of indices at which x and w match.

But since (w, x) is at least δ away from a perfect match, we know

$$n_{\text{match}}(x, w) \leq L - \delta \tag{3.5}$$

$$n_{\text{match}}(x, w) - L \leq -\delta \tag{3.6}$$

$$L - n_{\text{match}}(x, w) \geq \delta \tag{3.7}$$

Subtracting 3.4 from 3.3, we obtain

$$\Delta g_{w,\bar{w}} - \Delta g_{x,w} = (L - n_{\text{match}}(x, w))\Delta g_{\text{match}}.$$

Now substituting 3.7, we have

$$\Delta g_{w,\bar{w}} - \Delta g_{x,w} \geq \delta \Delta g_{\text{match}},$$

or, rearranging,

$$\Delta g_{x,w} \leq \Delta g_{w,\bar{w}} - \delta \cdot \Delta g_{\text{match}}.$$

This is precisely the linear mismatch-energy bound we are looking for. It says simply that the difference in interaction energy for a mismatched pair *versus* a matched pair is bounded above by a linear function of the magnitude of the mismatch.

3.1.4 Some reasonable operating parameters

Typical L :

In most proposed schemes, L is proposed to be about 40. This also gives

$$\begin{aligned} v_L &= v_1 L^{3\gamma} \\ &\approx (0.53\text{nm}^3)(40)^{9/5} \\ &\approx 400\text{nm}^3 \end{aligned}$$

Typical N :

In order for our computer to be remotely useful, we should have a value for N of at least 100 or so.

Typical α :

Recall α is the fraction of solution volume occupied by signals. For a typical monomer concentration of approximately $100\mu\text{M}$, and using our signal molecule “volume” v_L of approximately 400nm^3 , this gives

$$\begin{aligned}\alpha &\approx \left(\frac{100\ \mu\text{mol}}{L}\right)\left(\frac{6 \cdot 10^{23}}{\text{mol}}\right)(400\text{nm}^3)\left(\frac{1\text{l}}{10^{24}\text{nm}^3}\right) \\ &\approx \frac{0.024}{L} \\ &\approx 0.0006\end{aligned}$$

Typical ΔV :

If α is 0.0006, L is 40, and N is 100 – and if we wish a redundancy of unity – this gives a minimum value of ΔV of

$$\begin{aligned}\Delta V_{\min} &= \frac{Nv_L}{\alpha} \\ &\approx \frac{100 \cdot 400\text{nm}^3}{0.0006} \\ &\approx 70 \cdot 10^6\text{nm}^3 \\ &\approx 0.07 \cdot \mu\text{m}^3.\end{aligned}$$

For a more reasonable value of redundancy, say 10, we should adopt a ΔV closer to $0.7\mu\text{m}^3$.

3.1.5 Performance envelope of proposed *in vitro* DNA computers

We now have all the information we need, to start evaluating the size and shape of the performance envelope. We begin with the final- f relation, and evaluate the bound it gives us on the operating frequency. The final- f relation is:

$$f \leq D_1 L^{-\gamma} (k\Delta V)^{-2/3}$$

In three dimensions, the number of neighboring cells is somewhere between 6 and 27, depending on which neighborhood one chooses. For the sake of argument, let us pick a reasonable compromise value for k of 20.

As discussed above, we have D_1 is $500 \frac{\mu\text{m}^2}{\text{s}}$ as above, L is 40, γ is $3/5$, and ΔV is $0.7 \mu\text{m}^3$. So,

$$\begin{aligned} f &\leq \left(500 \frac{\mu\text{m}^2}{\text{s}}\right) (40)^{-3/5} (20 \cdot 0.7 \mu\text{m}^3)^{-2/3} \\ &\approx \left(500 \frac{\mu\text{m}^2}{\text{s}}\right) (0.11) \frac{1}{(14 \mu\text{m}^3)^{2/3}} \\ &\approx \left(500 \frac{\mu\text{m}^2}{\text{s}}\right) (0.11) \frac{1}{5.8 \mu\text{m}^2} \\ &\approx 9.5 \text{ Hz} \end{aligned}$$

Let us now proceed to the final- ε relation:

$$\langle \varepsilon \rangle > \exp\left[-\frac{\alpha \Delta V}{N v_1 L^{3\gamma}} \cdot \left[\frac{\Delta g_1}{KT} (4(L - \log_q N) + 3) + \ln 2\right]\right]. \quad (3.8)$$

We evaluate this bound given the parameters we have developed above.

First off, we know that $\frac{\alpha \Delta V}{N v_1 L^{3\gamma}}$ is just the redundancy, r , which we chose to be 10.

We know that for DNA at room temperature, $\frac{\Delta g_1}{KT}$ is about

$$\begin{aligned} \frac{\Delta g_1}{KT} &\approx \frac{1.67 \frac{\text{kcal}}{\text{mol}}}{0.59 \frac{\text{kcal}}{\text{mol}}} \\ &\approx 2.8 \end{aligned}$$

And we evaluate $4(L - \log_q N) + 3$ as

$$\begin{aligned} 4(L - \log_q N) + 3 &\approx 4\left(40 - \frac{\ln 100}{\ln 4}\right) + 3 \\ &\approx 4(40 - 3.3) + 3 \\ &\approx 150 \end{aligned}$$

But, we know that in no case should d_{exp} be greater than L , so in this case (few signals, large description length), we may replace the $4(L - \log_q N) + 3$ term with L . So,

$$\begin{aligned} \frac{\Delta g_1}{KT} d_{\text{exp}} + \ln 2 &\approx 2.8 \cdot 40 + 0.7 \\ &\approx 113. \end{aligned}$$

And finally, we compute:

$$\begin{aligned}\langle \varepsilon \rangle &> \exp[-10 \cdot 113] \\ &\approx 10^{-490},\end{aligned}$$

which is effectively zero. Some techniques for tightening this bound are discussed in the following section.

3.1.6 Discussion

Let's look at some reasons the lower bound on $\langle \varepsilon \rangle$ is so ridiculously small, and discuss some ways we might tighten that bound in the future.

crosstalk, false negatives, and false positives

In section 2.6, we only treated errors due to a signal s binding instances of an incorrect other signal. Let's call this "crosstalk". We didn't consider two other important sources of error, though:

- If s and s' are both present, but a molecule of class s remains *unbound*, a molecule error has also occurred. Let's call this a "false negative".
- If s is present and s' absent, but s is bound by *anything else*, a molecule error has also occurred. Let's call this a "false positive".

To see how false negatives and positives affect the error probability, consider the free energy diagram shown in figure 3.1. When s' goes from being present to being absent, this increases the Δg of complex ss' by an amount $RT \ln \sigma$ (recall σ is the ratio of the concentration of a signal when it is present to when it is absent). We can now see that we have two conflicting design requirements for $\Delta g_{\text{crosstalk}}$:

- in the case where s' is absent, $\Delta g_{\text{crosstalk}}$ should be large to avoid excessive crosstalk
- in the case where s' is present, $\Delta g_{\text{crosstalk}}$ should be small enough so that false negatives don't become a problematic source of error

This dilemma is a classic example of the sensitivity *vs.* selectivity problem.

A good guess would be to choose the signal molecule set such that $\Delta g_{\text{crosstalk}}$ is about one half the magnitude of $d_{\text{exp}} \Delta g_1$. Obviously, further analysis is warranted to see if this is the best choice in general. But the point remains: Not

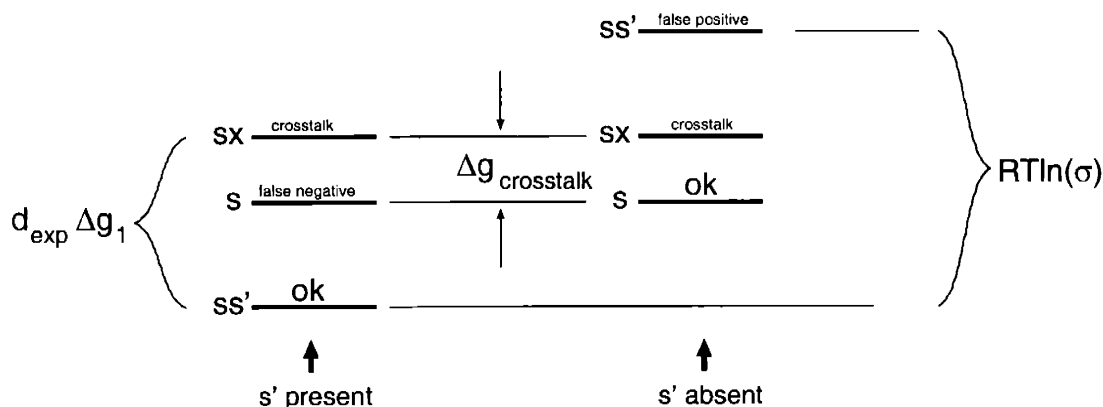


Figure 3.1: A free-energy diagram of states s (unbound), ss' (correctly bound if s' is present), and sx (incorrectly bound), for two situations – On the left, s' is present. On the right, s' is absent.

As can be seen, $\Delta g_{\text{crosstalk}}$ must be small enough to avoid false negatives in the s' -present case, yet large enough to avoid excessive crosstalk in the s' -absent case (see text).

all of the d_{exp} margin is available for avoiding crosstalk; only about half of it is. This cuts the exponent in the final- ε relation (equation 3.8) by a factor of about one half.

sloppy combinatorics when computing $\langle \varepsilon \rangle$ from ε

When we computed the signal error probability from the molecule error probability, also in section 2.6, we took the very simplistic assumption that a signal error occurs for s if and only if molecule errors occur for *all* molecules of class s in a cell. In fact, this represents a ridiculously stringent assumption, but it is easy to work with because the signal error probability can be expressed in closed form.

Since instances of s are indistinguishable, the only manner in which a signal error is distinguishable from a non-error, will be by counting erroneously bound instances of s . A more general treatment of the signal error probability would explicitly take this into account.

Consider the probability distribution of the number of molecules of class s which are erroneously bound, call it $\tilde{n}_e(s)$. To a good approximation⁵, $\tilde{n}_e(s)$ is a

⁵This approximation ignores coupling between the number of other molecules available to erroneously bind an instance of s . But, since there are plenty of other signal molecules around,

sum of r independent Bernoulli trials with expectation ϵ , hence $\tilde{n}_e(s)$ is binomial:

$$\Pr[\tilde{n}_e(s) = n_0] = \binom{r}{n_0} \epsilon^{n_0} (1 - \epsilon)^{r - n_0}. \quad (3.9)$$

If we choose a threshold value m (of erroneously bound instances of s) for which a signal error is said to occur, we have

$$\Pr[\text{error}] = \epsilon \quad (3.10)$$

$$= \Pr[\tilde{n}_e(s) \geq m] \quad (3.11)$$

$$= \sum_{i=m}^r \binom{r}{i} \epsilon^i (1 - \epsilon)^{r-i}. \quad (3.12)$$

In section 2.6, we effectively set $m = r$, thereby taking only the single smallest term of 3.9 as the error term. In general, we could pick a smaller value of m , tightening the bound on $\langle \epsilon \rangle$ due to crosstalk errors.

But obviously we can't choose m too small, because doing so would increase the sensitivity to false negatives. How small an m we could take is open to investigation, which should probably be approached via numerical simulation.

We can, however, get an idea how significant this effect is, by trying $m \approx \frac{r}{2}$, and seeing where it leads. First of all, we will have trouble evaluating the sum 3.12 analytically. We can of course get a lower bound by picking the largest term (the term for which $i = m$). We obtain

$$\Pr[\text{error}] \geq \binom{r}{r/2} [\epsilon \cdot (1 - \epsilon)]^{r/2}.$$

For any ϵ smaller than $\frac{1}{2}$, we can show $\binom{r}{r/2} (1 - \epsilon)^{r/2} > 1$, so

$$\Pr[\text{error}] > \epsilon^{r/2}.$$

This corresponds roughly to an additional halving of the exponent in the final- ϵ relation (equation 3.8).

and many of them are close to s' , we make the approximation that erroneous binding of all instances of s is independent.

nonzero conformational entropy

When we approximated the Gibbs factor of an invalid (crosstalk) state sx , we underestimated it substantially, since we assumed s and x would be aligned exactly end to end. In fact, this is ridiculously improbable; there are many more conformations in which x could bind to s . A more accurate treatment would explicitly account for the conformational entropy of all possible binding conformations between s and x . Rose *et al.* make some progress along these lines in [62], in which they numerically compute error probabilities by summing Δg over a large probability subset (the set of “staggered zipper” conformations) of the possible binding conformations.

This effect also has a potentially very significant effect on the error bound, and is a good area for additional work.

a conjecture

A reasonable conjecture we might make from the above arguments, is that we’ve underestimated the signal error probability by approximately a power of four. Obviously, this bears considerably more discussion. But in the next subsection, I will proceed with computing tradeoffs between L , N , and ΔV , as if this conjecture were valid. In other words, I will somewhat arbitrarily drop the factor of 4 in the exponent of 3.8. I will also drop the $\ln 2$ term for brevity, since it does not contribute significantly to the result. Our new conjectured “approximate final- ε ” relation is thus

$$\langle \varepsilon \rangle > \exp\left[-\frac{\alpha \Delta V}{N v_1 L^{3\gamma}} \cdot \frac{\Delta g_1}{KT} (L - \log_q N)\right].$$

With this in mind we will recalculate $\langle \varepsilon \rangle$ in the above case, and try to see if we can get a more reasonable tradeoff between L , N , and ΔV .

better balancing of L , N , and ΔV

First, let us recalculate for the $(40, 100, 0.7 \mu\text{m}^3)$ case. f is unchanged at 9.5Hz, but our estimate for the $\langle \varepsilon \rangle$ bound is now:

$$\begin{aligned} \langle \varepsilon \rangle &> \exp[-10 \cdot 2.8 \cdot (40 - \log_4(100))] \\ &\approx \exp[-10 \cdot 102] \\ &\approx 10^{-440}. \end{aligned}$$

Clearly, this is still a ridiculous tradeoff toward low error, especially since the speed is so low. Let’s try some different values.

Imagine we use 10-mers instead of 40-mers, and we still have 100 signals. Also, let us put $r = 4$ instead of 10. α is 0.0006, as before. v_L is now $(0.53\text{nm}^3)(10)^{9/5} \approx 33\text{nm}^3$. A reasonable value of ΔV is

$$\begin{aligned}\Delta V &= \frac{rNv_L}{\alpha} \\ &\approx \frac{4 \cdot 100 \cdot 33\text{nm}^3}{0.0006} \\ &\approx 22 \cdot 10^6\text{nm}^3 \\ &= 0.022\mu\text{m}^3,\end{aligned}$$

and so f is

$$\begin{aligned}f &\leq \left(500 \frac{\mu\text{m}^2}{\text{s}}\right)(10)^{-3/5}(20 \cdot 0.022\mu\text{m}^3)^{-2/3} \\ &\approx \left(500 \frac{\mu\text{m}^2}{\text{s}}\right)(0.25) \frac{1}{(0.44\mu\text{m}^3)^{2/3}} \\ &\approx \left(500 \frac{\mu\text{m}^2}{\text{s}}\right)(0.25) \frac{1}{0.58\mu\text{m}^2} \\ &\approx 220 \text{ Hz}.\end{aligned}$$

Now, the conjectured $\langle \varepsilon \rangle$ bound is

$$\begin{aligned}\langle \varepsilon \rangle &> \exp[-4 \cdot 2.8 \cdot (10 - \log_4 100)] \\ &\approx \exp[-74] \\ &\approx 10^{-32}.\end{aligned}$$

Clearly, for any problem solvable in less-than-astronomical time, an error rate of 10^{-32} is likely to be acceptable. And the benefit in frequency and cell size is great. So we would consider it a good tradeoff to pick $L = 10$, $r = 4$, rather than $L = 40$, $r = 10$.

Let's go even further; let's make the $\log_q N$ term in $L - \log_q N$ equal to about half the total. If $N = 100$, then $\log_q N \approx 3.3$. So let's pick $L = 7$. We now have $v_L \approx (0.53\text{nm}^3)(7)^{9/5} \approx 18\text{nm}^3$, and $\Delta V = \frac{4 \cdot 100 \cdot 18\text{nm}^3}{0.0006} \approx 0.012\mu\text{m}^3$. So f is approximately

$$\begin{aligned}f &\leq \left(500 \frac{\mu\text{m}^2}{\text{s}}\right)(7)^{-3/5}(20 \cdot 0.012\mu\text{m}^3)^{-2/3} \\ &\approx \left(500 \frac{\mu\text{m}^2}{\text{s}}\right)(0.31) \frac{1}{(0.24\mu\text{m}^3)^{2/3}}\end{aligned}$$

$$\begin{aligned} &\approx \left(500 \frac{\mu\text{m}^2}{\text{s}}\right)(0.31) \frac{1}{0.38\mu\text{m}^2} \\ &\approx 400 \text{ Hz.} \end{aligned}$$

And the conjectured $\langle \varepsilon \rangle$ bound is

$$\begin{aligned} \langle \varepsilon \rangle &> \exp[-4 \cdot 2.8 \cdot (7 - 3.3)] \\ &\approx \exp[-41] \\ &\approx 10^{-18}. \end{aligned}$$

pushing the limits: The large- N case

One claim often made in the DNA computation literature is that DNA computation may be in some ways better than electronic computation for solving large combinatorial problems. The model here is to consider the reaction as taking place in a single large vessel, with a large number of signal molecules (exponential in the problem size) representing candidate solutions for the problem at hand.

Let's try to push N up to near-astronomical values, to see how performance is affected. First, let's pick a challenging problem not readily solvable with modern electronic supercomputers. Say for instance the SAT problem with 70 variables, as Adleman treats in [4]. This problem has $2^{70} \approx 10^{21}$ possible solutions. To encode signals in a reasonable manner, Adleman chooses $L = 720$.

Let's also assume $r = 1$ is good enough for our purposes. This will give an *upper* bound on performance.

The ΔV (the reactor volume) for this situation comes out to $1.4 \cdot 10^{20} \mu\text{m}^3$, or 1400m^3 , which is ridiculously large but not unimaginable. But even worse, the estimated operating frequency, assuming diffusive transport, is about $5 \cdot 10^{-20} \text{Hz}$. This is on the astronomical time scale, meaning that diffusive transport is not feasible for problems of this size.

One could certainly consider stirring the reactor vessel, but at some cost:

- In a stirred reactor, we lose all spatial coherence; hence there is no backing down from the "single cell" reactor model. We can no longer trade signal density for redundancy.
- The problem of mixing a reactor vessel containing such a huge number of distinct chemical species, such that each species has a chance to react with each other (or with a fixed probe) is a nontrivial problem. This is because, with such small numbers of instances of each species (in this case, one),

the mixing must be completely ergodic in order for the reactions to take place predictably. Achieving ergodic mixing in large reactors is an entire problem field unto itself, and well outside the scope of this discussion; but clearly there are geometric and mechanical limits on the speed at which such mixing may be done.

In addition to these mentioned above, there is another problem lurking with extremely large N : The problem of reaction equilibration times growing arbitrarily long.

To see this, first consider that in this model of computation I have done only a simple static analysis of binding affinities; nothing is said about reaction *rates*. I have assumed the equilibration times for the chemical interactions between signals are at most comparable with the typical diffusion time to neighboring cells, as discussed earlier in section 2.1. If this is not the case – that is, if the equilibration times are much *longer* than the diffusion times – then I have overestimated the maximum operating speed of the chemical computer.

How much difference might we expect this to make? First, consider that the rate of forward progress for a binary association reaction $A + B \rightarrow AB$ goes as the product of the concentrations: $\frac{\partial[AB]}{\partial t} \sim [A] \cdot [B]$. Let's assume for simplicity that both concentrations are equal to the average concentration of a signal which is "present": $[A] = [B] \equiv [S]$. This nominal "present" concentration is simply $\frac{r}{\Delta V}$: the number of molecules of a "present" signal per unit volume. Using $\Delta V = \frac{rNv_L}{\alpha}$, this gives a scaling law on the nominal concentration of signals:

$$[S] \sim \frac{\alpha}{Nv_L}.$$

In turn, this means that for fixed α and v_L , the reaction rate for every binary reaction in the system scales *down* as $\frac{1}{N^2}$. So, since the target concentration goes as $\frac{1}{N}$ and the reaction rate goes as $\frac{1}{N^2}$, we would expect the relaxation time for any intended reaction to scale as $\frac{N^{-1}}{N^{-2}} = N$.

Clearly, this is bad news for the astronomically large N case: Even *with* a well-stirred tank reactor, the equilibration time goes up with N . In this case not because diffusion times go up, but because reaction rates go down with reduced partial concentration of signal molecules.

Another way to think about this, is that for a signal molecule of type s to interact with a signal molecule of type s' , not only must s and s' get to *nominally* the same place (that is, the probability densities of their locations must substantially overlap, either by diffusion or by stirring or by some other method) – but

r	L	N	ΔV (μm^3)	v_L (nm^3)	ΔV_{min} (μm^3)	f (Hz)	$\langle \varepsilon \rangle$ bound (strict)	$\langle \varepsilon \rangle$ bound (conjecture)
10	40	100	0.7	400	0.07	9.5	10^{-490}	10^{-440}
4	10	100	0.022	33	0.0055	220	10^{-128}	10^{-32}
4	7	100	0.012	18	0.003	400	10^{-72}	10^{-18}
1	720	2^{70}	$1.4 \cdot 10^{20}$	73000	$1.4 \cdot 10^{20}$	$5 \cdot 10^{-20}$	≈ 0	≈ 0
1	14	10000	1.0	61	1.0	13	10^{-34}	$3 \cdot 10^{-9}$
1	12	1000	0.075	46	0.075	85	10^{-34}	$3 \cdot 10^{-9}$
1	9	16	0.00075	28	0.00075	2200	10^{-34}	$3 \cdot 10^{-9}$

Table 3.1: **Summary of findings:** Conjectured and strict bounds on f and $\langle \varepsilon \rangle$, for various combinations of r , L , N , and ΔV discussed in the text. Recall that α is assumed to be 0.0006 in all cases.

they must also have the chance to find each other and interact on the *microscopic* scale. The chance of them finding each other is in turn determined by the product of their probability densities. Put another way – if signal molecule concentrations are low, their probability densities are small, and the chance of them *finding* each other to interact is smaller, so the reactions proceed more slowly. So active mixing is not a panacea for building chemical computers with extremely large N .

pushing the limits: The high speed limit

As a final probe on the size and shape of the performance envelope of DNA computers, let's see how fast we might hope them to operate.

Similarly to the large- N case, we will not waste reactor volume on excessively good error performance: We'll take $r = 1$, and pick L as small as possible to achieve a conjectured error bound of $3 \cdot 10^{-9}$ (see note⁶). I have evaluated three such cases, with $N = 10000$, $N = 1000$, and $N = 16$. I picked $N = 16$ as a stopping point, since trying to program a cellular automaton to accomplish any reasonably complex task with such a small N is already quite difficult. The speeds of these three cases come out at 13 Hz, 85 Hz, and 2200 Hz, respectively.

3.1.7 Summary of findings

The findings of this section on DNA computation are summarized in table 3.1.

⁶This error bound is so high as to be unacceptable for all but the shortest computations, but I chose it to give chemical computation the benefit of the doubt.

3.1. APPLICATIONS: DNA COMPUTERS

Before we leave this topic, realize that one factor I have glossed over in this analysis is the choice of α . Recall I chose $\alpha = 0.0006$ based on a $100 \frac{\mu\text{mol}}{\text{l}}$ concentration of monomers. Given a more generous choice for α , the operating frequency will increase, up to a point at which steric hindrance becomes a significant factor in molecular transport. The point at which this occurs is outside the scope of this work; but let's assume for the sake of argument that at $\alpha \approx 0.05$, operating frequency starts to drop precipitously due to steric hindrance.

With all other variables held constant, f goes as $\alpha^{2/3}$. So if we could increase α this far, all speeds in the table would increase by a factor of about $\left(\frac{0.05}{0.0006}\right)^{2/3}$. This yields a speedup factor of about 20.

3.2 Applications: Protein Computation in cells

Let's take a quick look at how good biological cells might be at computing with proteins. To do this, we will compute the hypothetical speed and error rate of protein-protein interaction in *E. Coli*, using values from the literature. First, some very brief background on proteins.

3.2.1 Chemical and mechanical properties of proteins

A protein is, for our simple purposes⁷, a polymer of amino acids, which folds up in the cytoplasm to a globular mass. There are 20 amino acids available. The sequence of amino acids determines the details of the structure into which the protein folds.

Each protein has one or more “active sites”, which mediates its interaction with other molecules in solution. For our purposes, these active sites will be our “signaling regions” – they are considered to interact with other in order to effect the logical progress of the information processing underway in the cell.⁸

The surface area of an active site is typically much smaller than the total surface area of the protein to which it belongs. This is because a good portion of the protein's mass is concerned with ensuring the protein folds reliably and quickly into its “proper” shape in solution, and with stabilizing the resulting structure.

γ will be taken to be 1/3, corresponding to a dense-packed spherical blob. This is quite nearly correct for globular (soluble) proteins.

We will take D_1 as the diffusion constant of a spherical blob-shaped hypothetical amino acid residue of average molecular weight, by the formula given in [18] and repeated in section 2.2:

$$D_1 = \frac{k_B T}{6\pi\eta_s R_D}. \quad (3.13)$$

estimating v_1 : To get R_D , we must have an estimate for v_1 . The 20 amino acids have somewhat different volumes, but we may compute an approximate average volume, from the following two items:

⁷We will not be making special consideration here for membrane-bound proteins, proteins with quaternary structure, or modified proteins (peptidoglycans, glycosylated proteins, *etc.*). We consider only single-chain, globular, soluble proteins.

⁸This also is a gross simplification. Many protein active sites interact with small substrate molecules (*e.g.* signalling molecules, nutritional molecules, *etc.*), many with binding sites on DNA, *etc.*

3.2. APPLICATIONS: PROTEIN COMPUTATION IN CELLS

1. The partial specific volume of a soluble protein varies only slightly from protein to protein, and assumes an average value of about $\frac{0.73\text{cm}^3}{\text{g}}$ (see Harpaz [33]⁹). This corresponds to a v_1 of about $1.2 \text{ \AA}^3/\text{Dalton}$.
2. The average amino acid residue in a protein has a molecular weight of 120 Daltons.

From these two, we conclude the approximate average volume per residue is

$$\begin{aligned} v_1 &\approx 1.2 \cdot 120 \text{ \AA}^3 \\ &= 0.24 \text{ nm}^3 \end{aligned}$$

calculation of D_1 : Recall we're assuming a monopeptide looks like a spherical "blob", for diffusion purposes. So

$$R_D = \left[\frac{3v_1}{4\pi} \right]^{1/3}$$

and so

$$D_1 = \frac{k_B T}{6\pi\eta_s} \left[\frac{4\pi}{3v_1} \right]^{1/3}.$$

Using room temperature and water as a solvent ($\eta_s \approx 0.0009 \frac{\text{N}\cdot\text{s}}{\text{m}^2}$), this gives

$$\begin{aligned} D_1 &\approx \frac{(1.4 \cdot 10^{-23} \frac{\text{J}}{\text{K}})(300\text{K})}{6\pi(9 \cdot 10^{-4} \frac{\text{N}\cdot\text{s}}{\text{m}^2})} \cdot \left[\frac{4\pi}{3 \cdot (0.24 \text{ nm}^3)} \right]^{1/3} \\ &\approx (2.5 \cdot 10^{-19} \frac{\text{m}^3}{\text{s}}) \left(\frac{1}{0.39 \text{ nm}} \right) \\ &\approx 6.5 \cdot 10^{-10} \frac{\text{m}^2}{\text{s}} \\ &= 650 \frac{\mu\text{m}^2}{\text{s}}. \end{aligned}$$

⁹the range of partial specific volumes of the 13 proteins represented in the Harpaz study ranged from 0.70 to $0.75 \frac{\text{cm}^3}{\text{g}}$

3.2.2 Some reasonable operating parameters

Typical L :

According to Neidhardt and Umbarger [57], in *E. Coli*, L is typically about 370 (see note¹⁰). This gives

$$\begin{aligned} v_L &= v_1 L^{3\gamma} \\ &\approx (0.24\text{nm}^3)(370) \\ &\approx 89\text{nm}^3. \end{aligned}$$

Typical N :

Again consulting Neidhardt and Umbarger, we see that 2D protein blot assays yield an estimate of the typical number of proteins in *E. Coli*:

$$N \approx 1850.$$

Typical ΔV :

Assuming the overall density of a cell is approximately that of water, and knowing the mass of a single cell [57], we have

$$\Delta V \approx 9.5 \cdot 10^{-15} \frac{\text{g}}{\text{cell}} \cdot \frac{1\mu\text{m}^3}{10^{-12}\text{g}} = 0.95 \frac{\mu\text{m}^3}{\text{cell}}.$$

Typical α :

Recall α is the fraction of solution volume occupied by signals. Again referring to Neidhardt and Umbarger, the dry-weight of a cell is about $2.8 \cdot 10^{-13}\text{g}$, of which about 55% is protein. Hence the protein mass is about $1.5 \cdot 10^{-13}\text{g}$. If again we take the specific gravity of proteins to be about 0.73 (see above), then the volume of protein in a cell is about

$$1.5 \cdot 10^{-13}\text{g} \cdot \frac{1\mu\text{m}^3}{0.73\mu\text{g}} \approx 0.2\mu\text{m}^3.$$

From this, we have

$$\alpha \approx \frac{0.2\mu\text{m}^3}{1\mu\text{m}^3} = 0.2.$$

¹⁰Neidhardt and Umbarger [57] give the molecular weight of a typical protein in *E. Coli* as 40kDa , and an average molecular weight per amino acid residue as 108Da . This yields an average length of $40\text{kDa}/108\text{Da} = 370$.

Typical r :

If α is 0.2, L is 370, N is 1850, and ΔV is $1\mu\text{m}^3$, this gives an approximate value for r :

$$\begin{aligned} r &= \frac{\alpha\Delta V}{Nv_L} \\ &\approx \frac{0.2 \cdot 1\mu\text{m}^3}{1850 \cdot 89\text{nm}^3} \\ &\approx 1200. \end{aligned}$$

3.2.3 Performance envelope of protein-based cellular information processing

estimating f We will now proceed to estimate f , using the final- f relation:

$$f \leq D_1 L^{-\gamma} (k\Delta V)^{-2/3}$$

Again, take k as about 20, D_1 is $650\frac{\mu\text{m}^2}{\text{s}}$ as above, L is 370, γ is $1/3$, and ΔV is $1\mu\text{m}^3$. Then

$$\begin{aligned} f &\leq \left(650\frac{\mu\text{m}^2}{\text{s}}\right)(370)^{-1/3}(20 \cdot 1\mu\text{m}^3)^{-2/3} \\ &\approx \left(650\frac{\mu\text{m}^2}{\text{s}}\right)(0.14)\frac{1}{(20\mu\text{m}^3)^{2/3}} \\ &\approx \left(650\frac{\mu\text{m}^2}{\text{s}}\right)(0.14)\frac{1}{7.4\mu\text{m}^2} \\ &\approx 12 \text{ Hz}. \end{aligned}$$

estimating $\langle\varepsilon\rangle$: There are a couple of problems with calculating $\langle\varepsilon\rangle$. First of all, we don't have a natural metric for measuring molecular "difference", as we did in the case of DNA. That is not to say none exists; just that it is not obvious how to find it. I have done some preliminary work on an abstract chemistry model which might be of use here, but not yet in sufficient detail to make any hypotheses of value¹¹.

¹¹Also, "Minimal basis set" models of physical chemistry might be of use in obtaining a more general result.

Another pitfall in the case of proteins, is that much of the amino acid sequence is of little or no relevance to the structure of the active site¹². This means that any estimate of distance based on simple Hamming distance between the *entire* sequence is likely to severely overestimate differences in interaction energy, for a randomly chosen perturbation. Obviously, in this case, our estimate of $\langle \epsilon \rangle$ is going to be astronomically low.

Finally, the difficulty of finding a corresponding Δg_1 so that the linear mismatch energy bound is obeyed, while at the same time providing a reasonably tight bound, is not easy. First, because of the above comments regarding irrelevance of the majority of amino acid substitutions *outside* the active site; but also because of the wide range of energetic effects of amino acid substitutions *inside* the active site.

One simple way around this set of problems might be the following approach: Let us imagine it can be shown that the number of amino acids typically relevant to the behavior of the active site is roughly a constant fraction of the total number of amino acids in a protein. Then we might have a chance at a reasonable distance metric – at least in the abstract – by considering that a protein has two different lengths: The “informationally effective length”, corresponding to the length of the substring specifying the active site, and the “mass length”, corresponding to the total size of the protein. The mass length would be used for calculating sizes and diffusion constants, while the informationally effective length would be used for estimating differences in interaction energies. But for now, we will leave this outside the scope of the present work.

¹²see, for instance, the large body of literature concerning substitution of nonstandard amino acid residues into proteins, in order to determine *which* residues contribute to the structure and function of the active site

Chapter 4

Conclusions on Chemical Computation

4.1 Review

In the first part of this thesis, I have improved understanding of the speed, density, and error rate of computation obtainable from chemical reaction systems. The approach used differs from that seen in the literature to date, since it considers several distinct aspects of chemical computer action, and tries to provide a more unified, synoptic view. The issues I have considered are summarized here, along with the most important messages and findings from each.

4.1.1 Implicit restrictions on random chemistries

This model accounts for the fact that freely chosen chemistries cannot be realistic in the general sense: Unlike a randomly chosen chemistry, real sets of reaction systems have many implicit dependencies among the rate- and equilibrium constants. Our simple way of modeling this effect in this work has been to develop a “string matching” approach to approximating equilibrium constants. In this model, the Δg of interaction between two molecules is related to distances between their description strings, in a string matching metric.

This has obvious direct relevance to computation schemes based on DNA hybridization, since string matching provides an easy approximation for DNA hybridization energy. The model developed in chapters 2 and 3 gives a relation between oligonucleotide length, number of signals, and error probability. Given two of these parameters, this model can inform us as to what we might expect from the third. This, in turn, can inform us as to how to choose design parameters for DNA computers to solve specific classes of problems.

The applicability of this approach may not be limited to DNA computation. It is certainly imaginable that we might extend this approach to other classes of chemistries – the only thing which is needed is to find a metric on the space of chemical species, such that the difference in interaction energies between two species and a common substrate, is linearly related to the distance between those two species. This may well be possible to do in general, by *constructing* such a metric from the matrix of interaction energies. But in the general case, the constant of proportionality Δg_1 is likely to be very large. Chemistries which are most useful for chemical computation, are those with small Δg_1 .

4.1.2 Errors due to randomness in chemical reaction systems

The second important aspect of the model given in chapters 2 and 3, is that it accounts for the stochastic nature of chemical reaction systems, and explicitly relates the rate of errors introduced into the computation to the underlying physics. In this particular treatment, I have done so by a three part argument:

- first: imposing a Maxwell-Boltzmann distribution on the fractional occupancies of bound states of signal molecules
- second: assuming synchronous operation, with a single sample of the state occupancy determining the forward progress of the computation
- finally, choosing an arbitrary cutoff as to the fractional occupancy of correctly bound states which will give rise to “correct” forward progress of the computation.

The first assumption is quite easy to justify, but not so the second and third. A more convincing exposition would relax the assumptions of synchronous sampling, and of simple cutoff logic.

In order to do this, I suspect one reasonable way would be to go directly back to the chemical Master equation, and pursue a mathematical treatment from the standpoint of time-coherence of stochastic processes. Such a treatment is for the moment, outside my existing skill set. To pursue this, the first step would be to understand enough terminology to be able to rigorously formulate the question, and then do a literature search on related topics, to see which tools might be applied.

Another problem with my treatment of this source of error, is that the cutoff assumption, combined with the coarseness of my counting of the fraction of states meeting the cutoff threshold, makes the resulting error bound ridiculously loose. A more careful counting could improve the error bound by a factor of two or more in the exponent.¹

4.1.3 Transport phenomena and computation speed

The third important aspect of the model presented in this thesis, is that it makes an effort to model the speed of evolution of the system in terms of physical trans-

¹review the conjecture in section 3.1.6, for more details as to why this is the case

port phenomena. Analyses of chemical computation which take these transport phenomena into account have been somewhat lacking in the literature to date.²

To put one in the correct frame of mind, consider that in the classical combinatorial DNA computing case, we are talking about equilibration of chemical systems with astronomically huge numbers of species, all present in vanishingly small concentrations. Equilibration of such systems is *necessarily* slow. The estimate I present in section 2.3 is crude, to be sure; but it at least gives some impression of the computation speeds we might expect from these systems.

Another important point of this treatment is that computation speed is explicitly related to several design parameters and physical-chemical constants. These factors include:

1. **fundamental particle size and solvent viscosity:** Since in this model signals must diffuse from one cell to its neighboring cells, the rate at which this diffusion takes place is an important determinant in overall speed. The diffusion constant of signal molecules is determined both by the solvent, and by the size of a single component of the signal molecule (the “monomer volume”, or v_1 , as defined in chapter 2).
2. **size of combinatorial space from which signal molecules are chosen:** The diffusion constant of signal molecules is also affected by how many times larger they are than the smallest possible signal molecule. This, in turn, affects the size of the combinatorial space from which signals can be chosen.
3. **computation “cell size”:** Although to my knowledge, no one has given detailed theoretical analysis nor experimental construction to the idea of a general chemical Cellular Automaton using diffusive transport, it seems fair to consider it as a reasonable first approach to large-scale chemical computation. At any rate, since the cell size is a parameter of my model, it certainly will not *reduce* the computation ability of the system below that of the “single cell” (*i.e.* Continuous Stirred Tank Reactor) case.

Taking cell size explicitly into account allows us to acknowledge and quantify the link between number of signals, redundancy, and the physical size of the signal molecules. This in turn has allowed us to make explicit the tradeoff between number of signals and speed.³

²for instance, Adleman talks of multiple cubic meter vats of ssDNA in solution, without taking into account the time required to process such vast amounts of fluid; much less the time it would take for the natural chemical processes to relax to equilibrium in such an astronomically huge, slowly mixing stochastic space

³and also the tradeoff between number of signals and error rate (see above)

4.1.4 Discussion

The results carry critical implication for the field of DNA computing, since it appears to be severely limited in computational speed. This confirms the widespread opinion that DNA computation should perhaps be suitable for combinatorial computing, but also brings into question the applicability of DNA computation to any kind of serialized problem. Building molecular computers to attack deeply serial problems may necessitate seeking more spatially organized approaches. One of these might be trying to approach Drexler's "machine phase" by spatially organizing the computational elements on some sort of scaffold. Perhaps in the near term, a more spatially organized approach to molecular transport in solution might be attempted, for instance by using MEMS to micromanipulate miniscule amounts of fluid.

4.2 Speculation: Neurons *vs.* other cell types

Consideration of the tradeoff between number of signals and speed gives us a new way of thinking about information transport in neurons. Here are a few simple observations:

Chordata have developed nervous systems in order to be able to respond rapidly to changes in their environment. Since the organisms may be physically large – typical distances may be a meter or more – neuronal information transport needs to be physically *fast* (in terms of velocity; not necessarily cycle time). But the “final-*f*” relation – which relates diffusion constant, molecule size, and frequency (equation 2.27) – tells us that if molecularly encoded information is moving fast, then the information is encoded in *small* molecules. This may be why neurons use very simple molecules – *i.e.* sodium, calcium, and potassium ions – for fast information transport.

But if information is encoded in small molecules, then in order to preserve chemical distinguishability, very little information may be encoded in each molecule – *i.e.* the number of distinguishable signals must be small⁴. And indeed, the number of distinguishable signals used for information transport is small; see the above comment regarding the ions used by neurons.

In the case of isotropic diffusive transport of a small number of chemical species, the bisection bandwidth density is not very large⁵. So if one wishes to move large amounts of information from one place to another at high speed, it seems there is no escaping the need for a routing structure, in order to relax the condition of isotropy.

Stated another way: If we broadly categorize computing systems into “routed” systems (*e.g.* electronic computers) and “unrouted” systems (*e.g.* chemical computers), then we might make an argument – for instance based on bisection bandwidth density – that routed systems have a fundamental advantage over unrouted systems. If this is so, then it is quite clear why routed systems (*e.g.* nervous systems) have evolved in nature: They confer a fundamental fitness advantage in terms of speed and complexity of reaction to stimuli.

⁴this can easily be seen from the final- ϵ relation (equation), in which we see that if L is small, then this puts a smaller bound on $\log_q N$, and hence on N

⁵This is obviously a broad statement, but I believe one which could be put on a firm theoretical footing with a modicum of work. This would be a good area for further study.

4.3 Speculation: Why are cells the size they are?

As we have discussed above – in many different ways – spatial organization is essential to pushing chemical computation past certain performance limits. Put another way, subdivided chemical systems have an inherent advantage over uniform diffuse systems in terms of their computation capacity. This is because in subdivided systems physical sequestration of signals can increase information density (by increasing the amount of information represented by each signal molecule), and may also increase the speed of information propagation (by introducing anisotropy into the environment).

Let us assume for a moment that living cells represent single sites in a chemical cellular automaton, as described in the previous chapter. Obviously, this is a gross simplification for myriad reasons; it is by no means meant to be a complete model. But let's consider the implications for a few moments.

The question is posed: “How large should a cell be?”. If we try to answer this question *de novo*, using only the ideas developed in this thesis, the answer must be thus:

“Cells must be:

- large enough to store and process enough information to accomplish their information processing tasks,
- large enough to have enough redundancy to not make errors in said information processing,
- yet not so large as to waste excessive biomass on needless redundancy.”

If cells were too much smaller, their behavior would of necessity be either

- much simpler, since they have fewer signal molecule classes to work with, or
- much more random in their behavior, since their average chemical redundancy would be smaller.

It's interesting to note that living cells have evolved to have a small but greater than one redundancy for most of their “important” information processing operations. A few examples:

- Most genes are present in approximately a copy number of one (two for eukaryotes). Bacterial plasmids range in copy number from about 3 to 500.

- Many repressor proteins are present in numbers from about 5 to 50 per cell.
- mRNA's are present in small integer quantities, for genes being expressed.

On the other hand, if cells were too much larger, they might be wasting a lot of biomass on *excessive* redundancy – their tasks are not terribly complex on an individual basis, so they only need some reasonable redundancy (say 5 to 50) and a reasonable number of signals, say a few thousand (corresponding roughly to the number of structural genes).

Let's look at some rough numbers. Say we have a hypothetical cell with 1850 structural genes, and that each corresponding protein averages 370 amino acid residues, as in section 3.2.2. Let us assume further the cell needs an average redundancy of 10, to minimize its error rate without being profligate. And let's say α is about 0.2. What is the minimum size of this cell, assuming there is nothing else inside it?

$$\begin{aligned} \Delta V &\approx \frac{NLr}{\alpha} \cdot v_1 \\ &\approx \frac{(1850)(370)(10)}{0.2} \cdot v_{1(\text{protein})} \\ &\approx 0.008\mu\text{m}^3 \end{aligned}$$

So the diameter of this cell, assuming sphericity, is about $0.25\mu\text{m}$. While this is certainly a fair bit smaller than most actual cells, it seems reasonable to hypothesize that if one includes all the overhead *not* included in this model (*e.g.* genetic information storage, gene transcription machinery, structural elements, motile elements, the entire metabolic apparatus, *etc, etc*), that living cells have evolved to *near maximal* information processing capacity for their size.

Part II

Measuring Computation

Chapter 5

Introduction to Computation Density

A significant problem in comparing electronic to chemical computers, is that the shape of their performance envelopes is so radically different: Current electronic computers are very high speed, and very low error; whereas current schemes for chemical computation can reach very high signal densities, but only at the expense of reduced speed and increased error rate.

Although we can calculate roughly the shape and size of the performance envelopes for chemical and electronic computers, we cannot compare them directly, for want of a single “figure of merit” by which to do so. In part two of this thesis, I establish such a figure of merit, by bringing together and extending previous work in the fields of Information Mechanics, Physics of Computation, and Cellular Automata theory.

I start with reasonable definitions of total Computation Capacity and Computation Density, as suggested by Toffoli in his paper on the Fungibility of Computation. From this point, the work proceeds in three major steps:

- First, I take the idea of computation capacity – phrased by Toffoli as a counting problem – and rephrase it in terms of a statistic. Having done so, I can *estimate* the computation capacity of a system by using statistical estimation. This also allows me to estimate the computation *density* of an infinite system (which presumably has infinite computation *capacity*), since my proposed statistical measure does not explicitly take the system size into account.
- Second, I elucidate the link between computation density and a property of dynamic systems called “specific ergodicity”. This gives an alternate way of evaluating the computation density of a medium from first principles, rather than through statistical measurement. Also, more importantly for the analysis at hand, it gives us an upper bound on the computation density of a medium in terms of its information storage capacity and its speed of operation.
- Finally, I take my statistical formulation of computation capacity, and extend it to error-prone systems. I then extend the analysis to computation *density* of infinite systems, by paralleling the argument mentioned above for error-free systems.

The above arguments lead to a bound on computation density of an infinite error-prone system in terms of information density, speed of operation, specific ergodicity, and error rate. This formulation is general enough that I can reach the goal of

the second part of this thesis: Comparing the computation density of current electronic computers with the maximum achievable by proposed models of chemical computation.

5.1 Computational media

It is considerably difficult to understand computation systems at a fundamental enough level to be able to make direct comparisons between such different things as a workstation computer and a vat of chemicals in solution! So to set the tone, I will take a large step back, and consider computation systems from a much more general – and somewhat philosophical – perspective. I hope the reader will bear with me; and I promise the openness in perspective will bear fruit in the chapters to come.

I begin by asking: What do a clock, a punch-card loom, a digital computer, a cell, and a lava lamp have in common? One answer is that they are all computational systems, made of computational media. We might say that “computation” goes on in all of them; certainly in some more than others. But what is computation, and how should we measure it? Certainly the computation taking place in a punchcard loom is not directly comparable to that in a modern digital computer.

A definition general enough to encompass all the above items is elusive, to put it mildly. The relations among some groups is well understood, but a broader understanding is not pervasive, maybe because the progress in understanding these relations takes place on so many distinct frontiers.

It is the goal of this second part of this thesis to contribute, in some small measure, to such a broader understanding. First, by establishing a taxonomic framework for classifying computational systems. Second, by surveying existing, well understood relations within some classes of the taxonomy. And third, hopefully breaking some new ground on some of the relations not well understood.

The taxonomy I will propose comes from comparing pairs of computational systems which differ in one respect, and enumerating the ways in which they differ. From these differences, I establish a set of character traits which may or may not be present in a computational system.

I then consider all possible subsets of these traits, and consider which subsets are represented by well understood systems. By looking at the boundaries of these areas, I identify open problem areas within the taxonomy.

To understand those open areas better, I adapt existing techniques which have been used to study some types of computational systems, and try to apply those

techniques to the open areas.

5.2 Traits of computational media

I will introduce some traits here by comparing pairs of computational media or systems, and seeing how they differ. Be warned, this is a “fast and loose” comparison. We will define things more explicitly when we get down to the chosen problem areas.

finitude

What is the difference between a digital computer and a universal Turing Machine? One easy answer is that one is finite and bounded; the other is not. The computer is a finite system; a “thing”. The Turing Machine won’t fit in a finite amount of space; it is more like “stuff” than a thing, except that it is nonuniform (see “structure”, below).

We can simulate the action of a computer on a TM, and to a certain point, we can simulate the action of a TM on a computer. But clearly we can’t go all the way, and simulate an arbitrary TM on a computer of finite size, in a finite length of time. Some problems, astronomically huge, will be too difficult for our computer; we will have to keep looking for larger and/or faster computers.

structure

Now consider the difference between a Turing Machine and a Cellular Automaton. They are both unbounded systems, but the Turing Machine has a special component, the head, which is only in one place at a time. The Turing Machine thus has some structure to it, as contrasted by the uniformity of the cellular automaton. The TM more closely fits our conventional definition of “machine” than does the CA, whereas a Cellular Automaton has a “smart dust” quality, or crystalline form, about it. The TM is “structured”, whereas the CA could be said to be “uniform”.

It is also worth noting, that the TM processes a finite amount of information per unit time, whereas the CA, in the idealized case of infinite extent, processes an infinite amount of information per time step.

determinism

So far in this section I've mentioned only deterministic systems. Some of these have finite state (*e.g.* the computer), whereas some have infinite state (*e.g.* the CA). In some practical sense, though, it is nonsensical – except in idealization – to consider infinite state systems as deterministic over long times, since any actual experiment can clearly only account for a portion of the information in the initial state.

So, independent of whether we believe physics to be deterministic or not, it is often useful to consider systems as being stochastic at some level of observation. This in fact is one of the central premises of statistical mechanics.

In the abstract: If we compare a finite state machine to a Markov machine, we have the essence of what the difference is between deterministic – or “ideal” systems, I will call them – and stochastic, or “physical” systems.

As other examples, compare the structure of a huge array of fully tested FPGAs to the structure of the Teramac [36] – or the structure of a conventionally designed, massively parallel supercomputer, to the more radical design of IBM's BlueGene [8] project. Both Teramac and BlueGene were deliberately designed to be tolerant to faults (either spatially and/or temporally distinct) introduced during construction or operation. They are designed for construction in a physical environment, above absolute zero temperature, with interaction with the surroundings, hence prone to failures. This environment is distinct from the mathematically ideal environment of finite state machines. This distinction is not merely academic nor pedantic – for example, the number of components in BlueGene, and the length of the computations for which the machine was intended, dictated that the machine would likely have several component failures *during any single job*.

Clearly, the finite state machine is a very special case of the Markov machine. If we have a universal Markov machine, we can easily simulate a FSM. But to go the other way; to simulate a Markov machine with an FSM, for any appreciable length of time, will require larger and larger amounts of FSM resources. This is very similar to the case mentioned above, in “finitude”.

universality

The term universality is typically used when a member of a class of systems can exhibit the behavior of any other element of the class. This is most familiar from theoretical computer science, where we have the notion of a UTM being universal in the class of TMs. Likewise, in practical computer engineering, we have the

notion of a “complete” instruction set. Or in digital hardware design, the notion of “logical completeness” of a set of gate types.

One fundamental insight derived from universality theory, is that there exist distinct classes of problems, some of which are strictly harder than others. Hence a machine which can solve a problem from a harder class is considered “more powerful” than a machine which can only solve problems from a proper subset of that class.

But this is a coarse distinction. In many fields, we have a finer measurement: We have “benchmarks” to distinguish between machines in the same universality class, but with varying performance. There are few who would argue that an Alpha processor is not more powerful, in some measurable sense, than a 6502.

One issue with benchmarks, though, is that they tend to be problem-specific. This is because they are *performance* oriented rather than *possibility* oriented. If someone asks you whether you would rather have a UTM or a pushdown automaton, you would clearly prefer the UTM. But if you were asked your preference between a 1.2 GHz G4 and a network of 1000 Sun-3's, your informed answer would definitely depend on what kind of problem you wished to tackle.

There is an open question as to what universality means for finite systems. But this thesis is not primarily concerned with universality; more with performance measurement. So we will be content with this loose understanding of universality for the present discussion.

5.3 Proposed taxonomy of computational systems and media

Following is a proposed taxonomy of computational media and systems, in tabular format. In the table cells are listed some examples of each taxon. This is obviously not meant to be complete; some of the important examples will be discussed in detail later.

		Deterministic/ Ideal		Stochastic/ Physical	
		Programmable/ Special	Universal/ General	Programmable/ Special	Universal/ General
Thinglike/ Finite	Structured/ Machine-like	Loom/ Counter	Computer	Lava lamp?	Fault tolerant computer
	Uniform/ Dust-like	PROM/ finite CA	FPGA	Arkin gas	Teramac/ BlueGene
Stufflike/ Infinite	Structured/ Machine-like	TM	UTM	Stoch. TM	?
	Uniform/ Dust-like	physical CA	universal CA	Stoch. CA/ Gunk	Gunk?

5.4 Techniques

Wherein we survey some techniques which have historically helped to understand the relations between specific pairs of these taxa.

generalize

By this I mean the process of reasoning that a class of computers has a distinguished subclass of universal machines. The classic example is Turing's proof of the existence of a UTM which can simulate any TM, and the consequent realization of the existence of a whole class of UTMs, properly contained in the class of TMs.

Similar reasoning has been carried out in the field of cellular automata [13].

One might consider a logic-complete FPGA as a universal PROM, since it can simulate any PROM and any logic-complete FPGA can simulate any other¹.

Similarly, one might consider a digital computer with a complete instruction set as a kind of generalized punch-card loom, from which it descended.

real-ize

Most often a mathematical model of computation must be modified and/or restricted before a physical embodiment of it can be constructed. This can be either in spatial extent (since we can't build infinite systems with finite resources), in spatial resolution, in error-rate, or other ways. In this paper I will consider real-

¹Within some scale factor, of course, and subject to some constraints on routability. We have also ignored the distinction here between FPGAs whose cells have registered *vs.* unregistered outputs. Clearly a bit more work would be needed to make this statement more rigorous.

ization as the process of adapting the idealized notions of determinism and infinite size to physically realizable systems.

By this I do not mean to suggest that physics is nondeterministic; I simply mean that practically sized systems – in a world where information travels at finite speed, at nonzero temperature, imperfectly isolated from outside influence, and measured with finite precision – are effectively not deterministic at very long time scales.

So what I mean by “real-izing” is to explicitly take account of these uncertainties in our mathematical treatment of the system.

amortize and value

In some simple cases, it is easy to order finite-sized computational systems in terms of quality, or usefulness. For instance, I can say that a computer with two megabytes of memory is “better” than an (otherwise identical) computer with one megabyte. Or I can say that an FPGA with 20,000 gates is “better” than one with 10,000, all other factors being equal.

But how should we put values on these qualities? Is an FPGA with 20,000 gates *twice* as good as one with 10,000? That is, would I trade two FPGA’s with 10,000 gates each, for one of 20,000 gates? Probably, but not necessarily. Or worse yet: would I trade a computer twice as fast for one with twice as much memory? It depends on my utility function.

One way to attack this problem is to try and answer it for infinite systems first. This may seem strange, but we will argue that it is easier to define a “value density” for a computational medium, than it is to define a value for a computer.

Amortization can be thought of as a renormalization process by which we can, if we’re careful, assign intensive quantities to infinite media. For instance, we might say that one universal CA is “better” than another if it runs twice as fast, or has twice as many bits of state per cell, *etc.*

In some cases (discussed in more detail below), we can actually proceed from a rank-ordering on computational systems to a numeric quantity which is intensive to the medium. Continuing the financial metaphor, I will call this “valuation”: instead of simply comparing systems or media to order them – as we do in universality arguments – we can measure a property of each numerically.

An implied consequence of the existence of such an intensive quantity, is that there is a corresponding extensive quantity, or “value”, defined on finite computational systems, which is approximately additive under system composition.

pulverize

To “pulverize”² a structured system is to abstract the functionality of the system in such a way that it can be embedded in a uniform, unstructured system. For instance, we can embed the capabilities of a microprocessor in an FPGA, by carefully setting the initial state of the FPGA. Likewise, we can embed a Universal Turing Machine in a Universal CA, by carefully setting the initial state of the CA [78].

In order to find a medium capable enough to do such an embedding, we must analyze a carefully designed and assembled, highly structured system, and abstract some characteristic which allows a “powder” to do the same tasks. For instance, for the FPGA, logical completeness of the cell set, plus some reasonable condition on routability, guarantees that with a large enough FPGA we can build the microprocessor we want.

For embedding the UTM in a CA, the situation is more subtle. Universality for CAs is a slightly different concept than universality for TMs, since an ideal CA is infinite in extent, and can therefore execute an infinite number of logical operations per unit time. So when we think of the space-time *density* of logical operations, the CA is a clear winner. Clearly we can build a UTM within a UCA without any speed or space tradeoff, but the converse is almost certainly not true. Presumably, there is an intermediate class of CAs which is universal for TMs, but not for CAs.

But as I mentioned above, we are not primarily concerned with universality here. So we will deliberately leave that discussion aside in this work.

5.5 Roadmap: toward computation density of infinite, dust-like, stochastic media

Much is understood about performance in structured and/or finite systems, both deterministic and stochastic.

But to approach the problem of measuring computation in such diverse systems as electronic VLSI circuits *vs.* chemical reaction systems taking place in solution, we must try to understand computation in a more general way. There are at least three nontrivial problems facing us in this effort:

- **space *vs.* time tradeoffs:** Electronic and chemical computers operate

²from the germanic root, “pulver”, or powder

at vastly disparate speeds and signal densities. In order to compare them directly, we must develop applicable techniques for doing so. Fortunately, there is rich literature on this topic.

- **generalizing to infinite systems:** Electronic computers are generally structured, whereas the chemical computers envisioned thus far operate in fluid phase, or in an unstructured way. In order to compare them, we must develop a technique to “amortize and valueate”, as discussed above.
- **incorporating error:** Chemical computers may operate with substantially higher error rates than their electronic counterparts. Again, we must figure out how to incorporate this difference into our evaluation. This corresponds to the process labelled “real-ize”, above.

Tommaso Toffoli, in his papers on the “Fungibility of Computation” [71][74], has developed a structure for accomplishing the first two tasks above: Dealing with space *vs.* time tradeoffs, and amortizing infinite systems. This gives us already an approach for evaluating dust-like infinite deterministic systems. But to my knowledge, performance has not yet been well quantified for infinite dust-like stochastic systems. Also, much is understood about universality in the context of deterministic and/or finite systems; but not that much about infinite dust-like stochastic systems.

We would like to move into the area of performance/universality of infinite dust-like stochastic systems; called “amorphous computation”, or “gunk”, by Abelson, Sussman, *et al.* [1], since chemical computation systems are reasonably well represented by this model. To do this, we will use Toffoli’s work as a starting point, and extend his measurement techniques to include the effect of error rate on usable computation power. We will accomplish this in a series of steps:

1. **real-ize:** from deterministic, finite to stochastic, finite (see section 5.5.1, below).
2. **amortize:** from deterministic, finite to deterministic, infinite (see section 5.5.2, below).
3. attempt to combine the results of the above two, to evaluate stochastic, infinite systems (see section 5.5.3, below).

A warning: some steps of this argument are clearer than others. Some are well supported; others are what I consider reasonable conjectures, given with supporting examples. Whatever one’s evaluation of the believability of the claims made

here, it is worth something to note that the final result is simple, and intuitively reasonable.

Such a result was not originally in the scope of this thesis, but I was pressed by the lack of results in this field to work in this direction, so that I could have reasonable basis on which to base performance comparisons of highly dissimilar computational media. There is more-than-ample ground here for further inquiry; my purpose here has been simply to get the results to a maturity sufficient to evaluate the performance of the dissimilar computational media discussed in part I of this work.

5.5.1 Real-ization: computation density of error prone systems

functional effectiveness and stochastic functional effectiveness

(Wherein I define the notion of functional effectiveness, and generalize it from deterministic functions to stochastic functions)

Toffoli has proposed [71][74] what I believe is a reasonable performance metric on deterministic, dust-like systems— that is, CAs. He begins with a definition of “computation density” for finite CAs, and presents experimental numerical evidence for the notion that this density is amortizable for infinite systems, and so in his terms, computational media is “fungible”.³

In his discussion of computation density, Toffoli is treating only deterministic systems. But he hints at a possible method for extending his definition to stochastic systems.

To close the argument reasonably rigorously, it’s necessary to do a few steps of work. I start with a quite restrictive case: replacing a computational system with a simple deterministic function mapping a discrete finite domain to a discrete finite range. I then ask, if we have a machine to compute this function, how much “computation capacity” can I say is inside the machine?

I treat this problem with an information theoretic/statistical mechanical argument: I define a functional value on a deterministic system, by an expected value over an ensemble of deterministic systems, randomly chosen from a suitable set of initial conditions. I call this measure “functional effectiveness”.

One nice aspect of the development of functional effectiveness, is the definition can be easily extended to stochastic functions (a function whose output is a

³or, in the parlance of finance theory, we might say it was a high-liquidity asset, for which the Law of One Price holds

random variable). I follow Toffoli's hint down this path, and develop a measure of how much computation we can expect, on average, from such a function. A simple example, for instance, is a deterministic function with an error term introduced into the input or the output.

I call this measure "stochastic functional effectiveness", and illustrate it for several simple examples.

steps toward stochastic computation density

To close the chapter, I will give a hint of things to come by suggesting that we can divide functional effectiveness by the log of the size of the input space, to arrive at normalized measures I will call functional density and stochastic functional density, respectively. This is a simpler parallel to Toffoli's normalization argument for obtaining computation density (a property intrinsic to a *medium*) from computational capacity (a property of a *system*).

This will lay the groundwork for migrating the results from the chapter on finite, error-prone computational systems, to infinite, error prone computational media. This result will be the basis of all applications given in this thesis, including Amorphous Computation, DNA computation, and generalized chemical computation.

5.5.2 Measurement: relation of computation capacity to specific ergodicity for deterministic systems

(Wherein I take a shew the relation between computational capacity, an extensive quantity, and specific ergodicity, an intensive quantity)

A significant obstacle to practical application of results on Toffoli's computation density, is the difficulty of evaluating it. It is defined as a limit, and each term of the limit sequence is exponentially more expensive to evaluate experimentally. He hints at a way past this, by reintroducing the notion of specific ergodicity of a dynamics, and saying that it is "intimately related" to computation capacity. He does not elucidate further on the relation between the two, at least in all the papers I have been able to find.

In this chapter, I try to shed some light on this relation. The approach parallels the previous section, where we measured functional effectiveness by choosing the expectation of a statistic over a randomly chosen input. Here, we proceed similarly by generalizing computation capacity to an expectation over a set of spacetime point-observations of the computational process.

The central idea is to consider the spacetime evolution of the entire system for all time, and to observe it at a sparse set of spacetime points. The question then becomes: How incrementally “surprising” can the system be? That is, if I start with some sparse set of observations, and I make a single other observation, what is the expected amount of information I will gain about the overall state evolution of the system?

In this case, the statistic I use is the conditional entropy of my next observation, conditioned on the values of my current set of observations. I will show, in turn, that

1. For a reasonably general class of systems, this value reduces to Toffoli’s computation capacity, as defined, and
2. if we take the expectation of this value over a reasonable distribution of starting states and observation points, the expectation reduces exactly to the specific ergodicity, multiplied by the log of the size of the state space.

5.5.3 How good is my gunk?: computation density of infinite, dust-like, error-prone media

(Wherein I combine the above results, and propose a quality measure for infinite, dust-like, stochastic computational media)

Once we have in hand a believable measure for computation capacity of finite, stochastic systems, and a way to measure it in terms of specific ergodicity, we can try and generalize to infinite, dust-like systems.

First, and most easily, for dust-like systems, the size of the state space goes exponentially with spatial size. Hence the error-free term in the stochastic functional capacity – expressed previously in terms of specific ergodicity and the size of the state space – can be rewritten in terms of the *information density* of the state space. In this way, the spatial extent of the system cancels, and we have the error-free term in our computation density, in terms of specific ergodicity.

It remains only to evaluate the error term. The error term can be thought of as the spacetime rate of entropy production, due to errors (hence *not* due to useful computation).

If we take errors to be discrete events, sparse in spacetime; and if we make the assumption that they occur almost independently⁴, then we can approximate the coarse-grained average of the spacetime entropy error rate as the product of two

⁴not an unreasonable assumption, if they are sparse

factors: The spacetime error arrival rate, and the expected entropy introduced at each error.

5.5.4 Discussion

These results, taken together, put us in a position to be able to approximate the computation density of infinite, dust-like, stochastic media. Admittedly, some of the steps I have taken are on stable footing, and others are more tenuous. However, none of them are unreasonable, and further work could no doubt improve the rigour, clarity, precision, and scope of the results. I will note, also, that the intuitive appeal and the simplicity of the final form of the results, speaks in some measure for itself.

5.6 Applications

classical electronic computers In the field of conventional digital electronic computers, much attention is paid to maximizing performance, while trying to keep error rates to an acceptable level. “Acceptable” generally means the computer is capable of performing typical tasks set before it, while maintaining an acceptably small global error probability. Error rates are, at least historically, extremely low, and even when they start to become a problem, application of modest amounts of highly localized error detection and/or correction generally reduces errors to astronomically small rates.

But this cannot be the case forever. As digital switching devices become ever smaller, the amount of energy required to spontaneously (hence erroneously) switch them becomes increasingly small, hence error rates per gate increase. Indeed, in modern silicon VLSI switching devices, spontaneous alpha-particle decay in the silicon substrate giving rise to erroneous switching is “embarrassingly frequent” [9].

Additionally, a large part of the reason to be reducing gate sizes and increasing speed, is to be able to tackle larger problems. But without error correction, the likelihood of completing a large problem without errors becomes vanishingly small. Indeed in massively parallel next generation supercomputers such as BlueGene [8], the aggregate MTBF of all components of the machine will be far less than the expected time to complete a typical problem!

So we have two confluent factors driving us to apply ever more error correction: The use of smaller hence more error-prone gates, and the need to solve larger

hence more error-prone problems. Since we are faced with the opportunity of using smaller, faster, yet more error-prone computational media, it is apparent that we need reasonable ways to compare the quality of such media on a rational basis. In other words, we wish to develop a “figure of merit” which measures comparatively how good or bad an error-prone computational medium is.

I will argue that computational density is such a figure of merit. If this is in fact the case, then it becomes important to be able to evaluate computational density for modern electronic computers. I will show how this can be done, and will provide some numerical estimates on computational densities achievable with state-of-the-art VLSI processors.

Cellular Automata machines Toffoli and Margolus have built a series of Cellular Automaton Machines, which are fast hardware-based implementations of finite-state CA’s [73], [75], [48], [76]. These machines are quite general, but are intended primarily for CA modeling of physics.

More recently, Margolus has proposed a next generation CA machine [50], [49], based on a very closely coupled (single chip) DRAM/processor architecture.

The concepts presented here will be directly applicable to evaluating the computation density of such machines.

biocomputation Finally, in the concluding chapter, we will give some analysis of the models of biochemical computation presented in chapter 3, with an eye toward evaluating their effectiveness as a competitor to electronic computers.

Chapter 6

Computation Capacity and Computation Density

6.1 Toffoli's computation capacity and computation density

Computation capacity

In trying to find a reasonable figure of merit for a computing system, we will start with Toffoli's "*computation capacity*" [71][74]. He defines capacity as the log of the number of distinct behaviors – or input/output transfer functions – a system might display. He gives the number of distinct behaviors the symbol n . Since systems larger in space and/or time can be expected to have an increasing number of possible behaviors, Toffoli parametrizes n by space and time: $n = n(X, T)$. He then writes the computation capacity as:

$$A_{\text{toffoli}}(X, T) \equiv \lg n(X, T).$$

He is careful to note that some computers have degenerate programs; that is, more than one program which can give rise to the same behavior.

Example: the PROM

Toffoli uses a PROM as a simple example of a computational system. Imagine a PROM with m address lines, and k data lines. The number of bits stored in the PROM is $k \cdot 2^m$. The behavior is the function programmed into the PROM; in other words, the mapping from input to output. Clearly, this system does not have program degeneracy, so n in this case is simply the number of programs, or $2^{k \cdot 2^m}$. Hence, $A_{\text{toffoli}} = k \cdot 2^m$, or simply the number of program bits in the PROM.

Computation density

If $n(X, T)$ were to grow exponentially with spacetime volume, then $A_{\text{toffoli}}(X, T)$ could be written more simply as

$$A = CXT,$$

for some constant C . We might then call C the "*computation density*", as A is now an extensive quantity, and C is its corresponding intensive quantity.

However, things are not that simple: The number of behaviors cannot be expected to grow *exactly* exponentially with spacetime volume. So, in order to define computation density, Toffoli makes some restrictions:

- he restricts his view to cellular automata,
- he redefines computation density as being a function of the spacetime volume of the cellular automaton: $C(X, T) = \frac{1}{XT} A_{\text{toffoli}}(X, T)$, and
- finally, he defines his computation density to be the limit (if it exists) of $C(X, T)$ as the spatial volume of the CA goes to infinity, and the time volume of the CA goes to zero:

$$C_{\text{toffoli}} \equiv \lim_{X \rightarrow \infty, T \rightarrow 0} C(X, T) = \lim_{X \rightarrow \infty} \frac{1}{X} \frac{\partial A(X, T)}{\partial T}. \quad (6.1)$$

He goes on to present numerical evidence that the limit seems to exist, in all cases he has (computationally) explored.

6.2 A statistical approach to computation capacity

In Toffoli's approach, we *count* the number of possible mappings – or transfer functions – from input to output, and take the log of that as the computation capacity of the box. But this may not be possible or desirable. For instance, the number of transfer functions may be huge, and may not have a regular combinatorial structure – so it may be intractable to count exhaustively, and difficult or impossible to compute analytically. Or as another example, say we do not have available all different transfer functions to count, but we can sample them at random. In this case, we may wish to simply *estimate* their number.

What of these cases? To approach this question, let's consider the system under study as a black box which is given us "preprogrammed", and on which we may do experiments. Our experiments are of the form: Apply an input, and observe the output.

A reasonable approach we could use to try and count the transfer functions, is to consider the transfer function of each box we are given as a sample of a random variable, and then try to estimate the amount of *information* in the box, by estimating the *entropy* of the random variable.

More concretely: Let us denote a transfer function $w : I \rightarrow O$ by a vector ω , composed of the outputs in response to each possible input:

$$\omega = \langle o_i \rangle, i \in I.$$

If we regard P , the program in the box, as a uniform random variable (call it \tilde{P}), and we regard the vector ω as a function of P , then the capacity is the entropy of the vector-valued random variable $\tilde{\omega}$, induced by applying ω to \tilde{P} :

$$A_{\text{stochastic}} \equiv H(\tilde{\omega}) = H(\omega(\tilde{P})).$$

6.2.1 Estimation techniques

If we have an agent presenting us with randomly programmed examples of the black box, then how might we estimate $H(\tilde{\omega})$, by experimentation? We have some set of examples, or programs, and we have some set of inputs (the components of ω). Should we spend more effort on measuring individual components of ω , for *all* examples we're given, or should we measure *all* the components of ω , for a smaller set of examples? Put another way, should we measure the *entire* transfer function for some small set of boxes? Or should we sample as many boxes as we can, and only spot-check the transfer functions on a small set of inputs?

example: the PROM under random sampling

Let's say our adversary is giving us randomly programmed PROMs, and let's imagine we elect to do only a single experiment per example given us (*i.e.*: per black box). This means we apply only a single address as input. Clearly we *cannot* arrive at a good measure of the capacity of the PROM in this way, since most of the program is being wasted for *every* example we test: Many possible programs are degenerate under this experiment (that is, don't produce observably different results). This happens for all programs which have the same value at the given address. So in this case, we have done too little experimentation to accurately measure the capacity of the system.

What of the other extreme case: Where we do *lots* of experiments? If we were to do more and more measurements, we could get more information out each time (if we chose our inputs intelligently), until we knew the entire PROM contents, or "program" in the box.

In the PROM case, when we do this, the transfer functions we obtain from independent program samples will be uncorrelated. That is, the transfer function for one sample gives us no information about the transfer function for any other sample.

Contrast this with the case in which someone gives us a black box with the same number of input and output lines; but which contains only a binary adder with

one of the operands preprogrammed (*i.e.*: the “program” in this case consists of specifying one of the operands to the addition). In this case, the transfer functions of multiple samples are correlated: The first one may look completely random, but the more samples we inspect, the less new information is gathered each time. In the limit, a single input (say, all zeroes) will suffice to tell us the entire transfer function of a new sample presented to us.

With a PROM in the black box, there is much to be discovered, and each experiment tells us very little. But with the simple adder in the black box, there is not as much to be discovered.

Keep in mind that we are trying to estimate the amount of information in a randomly chosen transfer function of the *generic* system, or of the *medium*. If we are given a *medium* for which the vast majority of programs are “boring”, then those vast majority are not worth considering compared to the few which are interesting, and we might say that the medium is “weak”. On the other hand, a medium which does surprising things no matter which way we prepare it, is a better general-purpose computer. For example, a medium which can count to some large number N *if we prepare it correctly* is not as interesting as a medium which can generate random permutations of N elements. For measuring computation capacity, we don’t actually care about the *detailed* manner in which the medium responds to different programs; we only care about how “surprising” its behavior can be over the set of all possible programs. That is, how diverse is the class of functions it can compute.

So, let’s try to estimate $H(\tilde{\omega})$ for the PROM. Consider a set of black boxes prepared with all possible programmings of the PROM. Now, let an agent choose one uniformly at random and present it to us. We may now do as many measurements on the box as we wish. In this case the outcomes of any set of measurements on different inputs are independent; the outcome for one input gives us no information about any other input. In the parlance of probability theory:

Let I be an “input space”, O be an “output space”. Let P be the space of programs, or functions from I to O . If I and O are finite, we can regard a member of P as a vector in the vector space $\Omega \equiv O^I$. An agent chooses a member of P and gives it to us for “experimentation”. Let’s say the program given us is a sample of the vector-valued random variable $\tilde{\omega} = \langle \tilde{o}_i \rangle$. Our “experimentation” consists of applying some inputs i to the black box and observing the outputs. Since $\tilde{\omega}$ is a vector-valued random variable, we can treat it as a collection of random variables \tilde{o}_i , and we can consider an “experiment” as sampling \tilde{o}_i , for some i .

In the case of the PROM, for any chosen program in P , all components of the vector are independent. Moreover, for the PROM, the statistical entropy for each

component is the same:

$$\begin{aligned}
 H(\widetilde{o}_k | \widetilde{o}_j) &= H(\widetilde{o}_k) && \text{for } j \neq k \\
 &H(\widetilde{o}_k) && \text{is constant over } k
 \end{aligned}$$

So, if we consider the ensemble of PROMs programmed all possible ways, presented all possible inputs, of course they will generate all possible outputs. But a stronger statement is: for each input, the information contained in the distribution of outputs will be the number of output bits, k ; the output distribution is independent of input; and the number of possible inputs is 2^m . So, the total information is $(\lg ||O||) \cdot ||I|| = k \cdot 2^m$, or precisely the number of programming bits in the PROM.

So the PROM is a very good computing medium indeed: It has a large program space, and each program gives rise to a distinct behavior. Its computation capacity is equal to its number of programming bits.

6.3 Eliding the distinction between program and input

In attempting to abstract away the concreteness of existing models of computation, the question arises: How should we define “program”, and how “input”, so as to arrive at a believable definition of computation capacity? One problem which comes up when trying to work more deeply with Toffoli’s definitions, is that the capacity is defined only for a given partition of program from input. Who is to say that the capacity will not be different for a different choice of which lines are “program” and which are “inputs”. Ideally, a good definition of capacity will be invariant under this choice.

Let us return to our friend the PROM. What should we consider its “input”, and what its “program”?

Neglecting the obvious answer for a moment, let’s ask if there is any other way we can split up the program from the input. If there are other reasonable ways to do so, that should not change the total computation capacity of the PROM.

Let’s consider swapping our definition of program and input, and see what happens. The “input” now will be the entire state of the bits stored in the PROM. So there are $2^{k \cdot 2^m}$ possible inputs. And the “program” will be the state of the m address lines. So there can be no more than 2^m distinct behaviors. Clearly

something is wrong here: Most of the input is being summarily ignored, while the number of programs is way too small to elicit the full range of behaviors we would expect from the PROM. By changing the definition of which lines we designate as program, and which as input, we have dramatically reduced the capacity of the PROM as Toffoli defines it.

What can be done? To go further, as stated above, we'd like a definition which is invariant under the partition of program *vs.* input. Let us investigate whether, by modifying or expanding the definition of capacity, we can arrive at such a measure.

a proposal

Let's try redefining capacity to be the supremum:

Let \mathcal{P} be a partition of the input space into a "program" set P and an "input" set I . Without loss of generality, assume $I = \{1, 2, 3, \dots, ||I||\}$. For a given \mathcal{P} , we have

$$\begin{aligned} A_{\mathcal{P}} &= H(\omega(\tilde{P})) \\ &= H[\langle o_1, o_2, \dots, o_{||I||} \rangle (\tilde{P})]. \end{aligned}$$

Intuitively, if we vary the partition \mathcal{P} , the following effects occur:

- If we take \mathcal{P} such that I is large and P is small, there aren't enough distinct programs to produce very many independent output vectors, so H becomes small.
- Conversely, if we take \mathcal{P} such that I is small and P is large, there aren't enough output vectors to distinguish the behaviors of all the programs, so H again becomes small.

In general, H will be maximized when \mathcal{P} is somewhere between all-input and all-program.

In lieu of trying to evaluate the optimal partition in general (a hopeless task), we choose for now to simply define A to be the maximum of H , over all possible partitions:

$$A_{\text{generalized}} = \max_{\mathcal{P}} H(\omega(\tilde{P})).$$

example: 2 input gate

Consider the example of an AND gate: it has 2 inputs and one output. The possible partitions of the input bits are:

- both inputs are “program” bits
- both inputs are “input” bits
- one input is a “program” and the other is an “input”

The table shows a calculation of $H(\tilde{\omega})$ in each case¹:

both program		both input		one program, one input	
p	ω	p	ω	p	ω
< 0, 0 >	< 0 >	<>	< 0, 0, 0, 1 >	< 0 >	< 0, 0 >
< 0, 1 >	< 0 >			< 1 >	< 0, 1 >
< 1, 0 >	< 0 >				
< 1, 1 >	< 1 >				
$H(\tilde{\omega}) = \frac{1}{4} \lg \frac{1}{4} + \frac{3}{4} \lg \frac{3}{4} \approx 0.84$		$H(\tilde{\omega}) = 0$		$H(\tilde{\omega}) = 1$	

So, the computation capacity of an AND gate is 1 (as we might have guessed).

a less tractable but more restricted case

Consider the logic cell shown in figure 6.1. This is a unit cell from the QuickLogic pASIC I series of FPGAs². When I worked at QuickLogic, it was an open question for us in the electronic CAD department, as to how many distinguishable logic functions were achievable with this cell, by tying the inputs high or low or together. We never solved this problem; we determined an exact counting was computationally infeasible with the resources available to us at the time.

The approach above gives us an alternate way of finding the number of functions achievable with this cell:

¹Here we take the shorthand of denoting $\omega(\tilde{P})$ by $\tilde{\omega}$. We will often do this in the following sections.

²The pASIC cell actually had a registered output, but that is beyond the scope of the present discussion.

6.3. ELIDING THE DISTINCTION BETWEEN PROGRAM AND INPUT

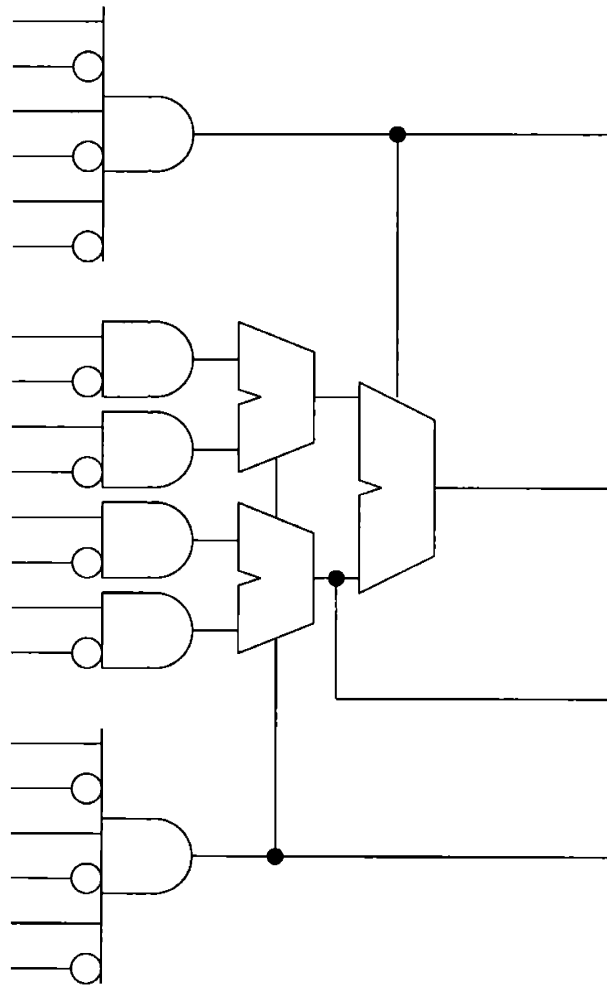


Figure 6.1: A unit cell from the QuickLogic pASIC I series of FPGAs.

1. for each partition of program and input bits,
2. for each possible state of the input bits,
3. compute the entropy of the output, as we change the program bits.
4. finally, take the maximum over the choice of program/input partition

A side observation, is that this gives a sketch of a Monte Carlo algorithm for estimating the computation capacity of a system³.

“null-input” computation capacity

Since the number of functions over an input set is exponentially larger than the number of members in the input set, it seems reasonable to conjecture that in most cases, the (program, input) partition which gives the largest $H(\tilde{\omega})$ is likely to be skewed heavily in favor of programming bits, and away from input bits. For example, recall the comparison between the PROM and the PROM with its program and input bits exchanged: Having too few programming bits and too many input bits radically reduced $H(\tilde{\omega})$.

For this reason, I will introduce the notion of “null-input” computation capacity. This is just $H(\tilde{\omega})$, for the case in which all initial state is taken to be program, and none is considered input. This simplification will become more useful when, in the following section and the following chapter, we are dealing predominantly with spatially uniform systems such as cellular automata.

It is also worthwhile to note that the set of gates having maximal capacity in the null-input case, contains the set of *reversible* gates⁴. And indeed, the set of gates having a capacity equal to the total number of inputs, is exactly the set of

³Various Monte Carlo algorithms are possible. The simplest to envision is to estimate the transfer function entropy $H(\tilde{\omega})$ for each possible (program, input) partition by statistical sampling techniques, then finding the maximum over the partition. If there are too many possible partitions, we could either statistically sample from the space of partitions; or in some cases, we can reduce the number of partitions by finding equivalence classes under equivalence of $H(\tilde{\omega})$, counting the size of the equivalence classes, and sampling only one point within each equivalence class.

⁴proof sketch:

First, realize $H(\tilde{\omega})$ cannot be larger than the number of program+input bits $\|P\| + \|I\|$. For a reversible gate, $\|O\| = \|P\| + \|I\|$, so if $H(\tilde{\omega})$ can reach this number, then it is maximal. But indeed, for a reversible gate, each output is distinct (otherwise it's not reversible), so $H(\tilde{\omega})$ is exactly $\|O\| = \|P\| + \|I\|$.

reversible gates⁵. For instance, the capacity of the Fredkin gate [27], or “controlled not” gate, by this measure is exactly 2; and that of the Toffoli gate, or “controlled swap” gate is exactly 3.

6.4 Computation density in spatially distributed systems

density of space-decomposable automata

Imagine we have a spatially distributed array of N identical cells, like the cell in figure 6.1, above. Then, the size of the $Program \otimes Input$ space is $(2^{n_i})^N = 2^{N \cdot n_i}$, instead of 2^{n_i} . So if the capacity of a cell is denoted by A_{cell} , then it is conceivable that the total capacity of the array is as large as $N \cdot A_{\text{cell}}$.

If we wish to solve larger problems by composing a number of smaller cells in this way, then we must couple the cells in some way (by taking some inputs of some cells as being dependent on inputs or outputs of other cells). But intuitively, coupling decreases the available entropy of the input and/or program, by restricting our choice to a subfield (of the Borel field which is the cross product of all the individual input domains). That is, if we restrict the program set to a subfield (by programming the cells not completely independently), then the entropy of the output vector can only go down. Likewise, if we restrict the input to a subfield, the entropy of the output vector can only go down.

In fact, it can be rigorously shown that capacity is sub-additive. That is, any combination of two systems yields a system with *at most* the sum of the computation capacities of the two individual systems. For a proof of this, see appendix A. It is therefore apparent that if we divide the total capacity by the size of the array to get the aggregate density of the array, we will wind up with $C_{\text{array}} \leq C_{\text{cell}}$. If we wish to maximize C_{array} , we will need to allocate the state of the array to be mostly programming bits. In this case, the size of I will be about the same, but $H(\tilde{\omega})$ can be about N times as large.

⁵proof sketch:

A gate generates at most $2^{|P|}$ output vectors, so it has a capacity no more than $|P|$. In order for it to have capacity $|P| + |I|$, then, $|I|$ must be zero. And indeed, for it to have maximal capacity, it must have a full complement of $2^{|P|}$ distinct output vectors. So $H(\tilde{\omega}) = H(P) = |P|$. But if $H(\tilde{\omega}) = |P|$ and $|I| = 0$, this means all inputs to the gate (I , which is null, and P , which can be obtained by reverse mapping the output) can be obtained from the output. Which is to say, the gate is reversible.

coupling

To solve larger problems, of course in general we will need more computation capacity. But as we have just seen, the price we will pay for increasing capacity is decreasing density – because to tackle larger problems, in general we must couple smaller computers to obtain larger computers.

example: coupled *vs.* noncoupled systems

Consider an array of N decoupled 2-input gates, each of unit capacity. The array has $2N$ inputs and N outputs. If we choose one input from each gate to be a programming bit, and the other input to be an input bit, then the program space is 2^N , and each program gives a distinct transfer function. So the aggregate capacity is at least N . Since capacity is sub-additive (shown in appendix A), and each gate has capacity 1, then the total capacity of the array must be exactly N .

But this is not a very useful computer, even though it has high capacity. To make it useful (that is to say, to make it compute a more “interesting” function), we must sacrifice some capacity by coupling the parts of the system together. Let us consider another system with the same number of inputs and outputs, ($2N$ and N , *resp.*), but which is made up of components internally coupled to perform a more useful function. As an example, we will use the N -bit binary adder.

The N -bit binary adder can be built from about $4N$ two-input gates, in any one of a number of obvious ways. Taken separately (or as an uncoupled array), these $4N$ gates have capacity $4N$. But because of the coupling, the total system capacity is reduced to somewhere between N and $2N$ (see note ⁶).

6.5 Computation density in time-evolving systems

Thus far we have discussed only combinatorial networks, which map any given input to a uniquely determined output. This is fine as far as it goes, but is a limited way to look at general purpose computers. In order to get a better understanding of computational dynamics, we have to consider time evolution.

⁶To see $A \leq 2N$, note that the capacity cannot be larger than the number of programming bits, which in turn cannot be larger than the total number of inputs, $2N$. To see $A \geq N$, note that in the case we take one N -bit summand as the “program” and one N -bit summand as the “input”, we have 2^N available transfer functions, so we at least have shown a partition for which $H(\bar{\omega})$ is N .

The simplest way to think about this is to treat the network as a pipelineable computation resource.

In this approach, we assume that information “flows” through the network in a coherent wavefront. If we then build deeper and deeper networks, eventually the length of time to produce a result will be much greater than the rate at which arguments can be presented. At this point, we can think of the network as an logical “pipeline”, or a sort of transmission line in which logical operations take place. We can then say that a faster pipeline with an identical set of available functions has a higher computation *density* (see Toffoli’s definition, above)⁷.

Pipeline machines are of some interest, but most computations of interest are those which have deep dependencies. If we were to try to build pipeline machines to solve problems with deeper and deeper dependencies, the machines would take unbounded physical resources. In addition, we may only want to solve a single instance of the problem. In this case, building a very deep machine with many pipeline stages is a waste of materials, since at any given time, only a tiny fraction of the machine is being used for relevant computation. To work around this problem, we build machines which feed their own outputs back to their inputs.

time-composed automata and feedback automata

Consider an automaton which is not necessarily spatially cellular in nature, but whose range is identical to its domain. If we continue to apply this automaton’s output back to its input, we have what we call a “dynamics”, in the parlance of physics, or an “iterative map”, in the parlance of mathematics. We will call such a machine (pictured in figure 6.2) a “feedback automaton”.

How should we measure the computation density of a feedback automaton? Since the introduction of a time parameter, the output is no longer a function of only the input bits. This is because, unlike the pipeline machine, there is no defined “end”. We can continue a computation for as long as we wish.

loss of strict time-independence

At first glance, we might naïvely think that, because we can continue the operation of a feedback automaton for as long as we wish, then we could get an arbitrarily large number of functions from it, just by running it longer. Clearly this cannot be the case for a machine of any finite size: a time-composed automaton with

⁷For more discussion of treating combinatorial circuits like transmission lines, see *e.g.* references [22] and [49].

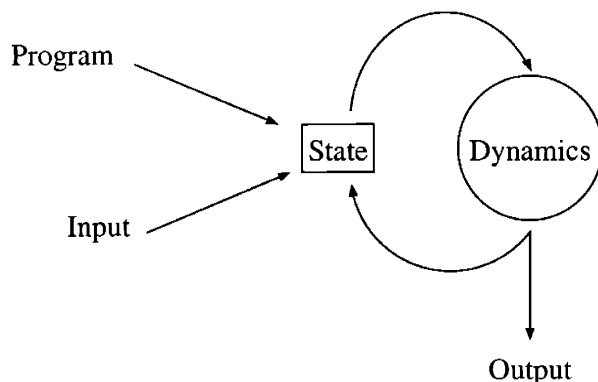


Figure 6.2: A feedback automaton

a finite size is a finite state machine. Since there are a finite number of initial states, there is a finite length of time before the dynamics cycles. The number of functions is limited by the size of the state space.

Put another way: For the pipeline machine, we can present a fresh, independent input to the machine at every time step, so the outputs at each time can also be independent. But with a feedback automaton, we sacrifice this independence: since the machine is deterministic, if we know the entire machine state for any given time, then its subsequent behavior is completely determined.

Since the behavior of a feedback automaton is completely determined by a single function on its state space, we might say that it has zero computation capacity. It has, in some sense, no programmability. But this is similar to the case with the combinatorial logic network, in which we had to split the state space of the machine into an “Input” space, and a “Program” space: once we give up some variety available to us in inputs, we can recover some programmability. It is no surprise, then, that this can be done also with feedback automata.

computation capacity of finite feedback automata

Say we have a feedback automaton; how should we compute its computation capacity? One approach is to introduce an additional input parameter τ to our transfer function, which will be the number of timesteps for which to run the machine. We now have, instead of $w_P(I) \rightarrow O$, $w_{P,\tau}(I) \rightarrow O$.

Let’s consider only the trivial input space (zero entropy), and the universal program space⁸. This will give us a lower bound on C .

⁸indeed, since the “program” and the “input” are combined into a single state variable, *before*

Table 6.1: the tableau matrix of outputs of a finite, deterministic, null-input feedback automaton.

	0	1	2	3	...	k	...
P_1	$\omega(P_1, 0)$	$\omega(P_1, 1)$	$\omega(P_1, 2)$	$\omega(P_1, 3)$...	$\omega(P_1, k)$...
P_2	$\omega(P_2, 0)$	$\omega(P_2, 1)$	$\omega(P_2, 2)$	$\omega(P_2, 3)$		$\omega(P_2, k)$	
P_3	$\omega(P_3, 0)$	$\omega(P_3, 1)$	$\omega(P_3, 2)$	$\omega(P_3, 3)$		$\omega(P_3, k)$	
...							
P_m	$\omega(P_m, 0)$	$\omega(P_m, 1)$	$\omega(P_m, 2)$	$\omega(P_m, 3)$		$\omega(P_m, k)$	

Proceeding along the previous lines, let us try and evaluate the entropy of the output vector $H(\tilde{\omega}_{P,\tau})$ over random P and τ . Since the input is trivial, $\tilde{\omega}$ has only one component, call it $\omega(P, \tau)$. This is just the state obtained by running the state P through τ timesteps of the dynamics. The possible outputs of the machine are shown schematically in the tableau matrix in table 6.1.

Let us try to evaluate $H(\tilde{\omega})$. One problem which arises here, is taking an expectation over a τ chosen “uniformly” from $[0, \infty >$. We will define this as the limit

$$H(\tilde{\omega}) \equiv - \lim_{t \rightarrow \infty} E_{P \in \mathcal{P}} [E_{\tau \in [0,t]} [\lg \Pr[\omega(P, \tau)]]] \tag{6.2}$$

To show this limit always exists for this class of machines, I make the following argument:

Consider that the dynamics is finite and deterministic. This means that the sequence of machine states starting with any starting state, eventually enters a cycle. Moreover, the cycle is repeated *ad infinitum*; so for large enough t , only the states in the cycle will count in the expectation over τ . That is, the initial transient becomes less significant as t becomes large.

Also, since the dynamics is deterministic, each cycle is simple. That is, each state appearing on the cycle appears *exactly once* on the cycle. So, for any given P , as t becomes large, the $\Pr[\omega(P, \tau)]$ term approaches either zero (for those states on a transient), or $\frac{1}{\text{orbit}(P)}$ (for those states not on a transient), independent of τ (see note ⁹).

So, for each P , the following limit exists:

$$\lim_{t \rightarrow \infty} E_{\tau \in [0,t]} [\lg \Pr[\omega(P, \tau)]]$$

we start running the automaton, it makes perfect sense to completely ignore any distinction between program and input

⁹here, $\|\text{orbit}(P)\|$ denotes the length of the cycle eventually entered from state P

and its value is $\lg \frac{1}{\|\text{orbit}(P)\|}$ if P is on an orbit, and 0 if P is a transient state.

Since the limit exists, we may interchange the order of the limit, and the expectation over \mathcal{P} in equation 6.2 to obtain:

$$A_{\text{stochastic}} = H(\tilde{\omega}) \tag{6.3}$$

$$= -E_{P \in \mathcal{P}} \left[\lg \frac{1}{\|\text{orbit}(P)\|} I_{\text{orbit}}(P) \right], \tag{6.4}$$

where $I_{\text{orbit}}(P)$ is the indicator function of whether P lies on an orbit, or is a transient state.

We will revisit this relation later, in the section on Specific Ergodicity and its relation to computation density.

To obtain the computation density, we should divide by X and differentiate with respect to T , as shown in equation 6.1. But differentiating with respect to T no longer makes sense – since we’re dealing with feedback automata, the capacity no longer depends on a time parameter. The correct substitution is to replace the differentiation by T with division by τ , the cycle time of the computer. This is equivalent to multiplying by f , and guarantees that the computation densities of functionally equivalent media scale in direct proportion to the execution speeds.

Putting this all together, we have a proposal for stochastic computation density of a feedback automaton:

$$C_{\text{stochastic}} = f \cdot \lim_{X \rightarrow \infty} \frac{1}{X} H(\tilde{\omega}) \tag{6.5}$$

$$= -f \cdot \lim_{X \rightarrow \infty} \frac{1}{X} E_{P \in \mathcal{P}} \left[\lg \frac{1}{\|\text{orbit}(P)\|} I_{\text{orbit}}(P) \right]. \tag{6.6}$$

6.6 Infinite systems: Cellular Automata

Let's try and extend these ideas from finite systems ("thing-like" computers), to infinite media ("stuff-like" computers). This is more difficult than the analysis of the previous section, in that we don't have a finite set of examples for our agent to choose from; so we must somehow sample from an infinite set of examples. Additionally, there is a normalization problem: Presumably the computation *capacity* of an infinite system is infinite, so reducing it to computation *density* seems a bit difficult.

This seems fairly hopeless at first glance. But if we restrict ourselves to *uniform* ("dust-like") systems, then we have a chance. Recall infinite uniform deterministic systems are cellular automata. Also recall in these systems the speed of information propagation is finite, and the medium is uniform. So instead of being presented with differently programmed examples, we can sparsely sample *just one* system and learn all we need to about the dynamics¹⁰, in order to judge its computation density.

We start with the following assumptions:

1. We have a certain computational substance available for a given spacetime extent (X, T) .
2. The substance is uniform – that is, we can relocate computations by spacetime translation¹¹.
3. The complete dynamics is given by the combination of substance and initial state.
4. The dynamics is deterministic.
5. The dynamics is reversible.
6. Information flow is local – that is, the speed of information propagation is finite.

a different approach: spacetime entropy rate as computation density

Recall what we've done so far: Whereas Toffoli is concerned with counting the *number* of possible functions of a system, we have generalized this definition to how

¹⁰assuming it has enough randomness in the initial state

¹¹note this also implies that the information density of the system state is uniform

surprising a randomly chosen function can be. This is our new way of estimating the number of distinct behaviors the system can display.

A deterministic dynamics maps an initial state to an infinite sequence of output states. Usually we consider a dynamics as being a function from states to states. For this analysis, though, it is convenient to consider instead the function from states to infinite sequences of states induced by the dynamics. The task before us then becomes: We want to know the spatial density of how “surprising” this function can be, per unit of time.

examples:

- **identity dynamics:** The identity dynamics is as boring as they get: If we look at one site, at any particular moment, we know everything about it, for all time. So however many sites we observe, we know all about them for all time. Our rate of surprise is zero.
- **random states at each time:** This system presents a new, random state at each time (hence, note, it is *not* actually a dynamics in the sense we are considering). It is very surprising: we get one site’s worth of information per unit space, per unit time. But it’s not a “function”: the output bears no relation to the input. No deterministic finite-state computer can generate this amount of “surprise” over long periods of time (since it will eventually exhaust the state space).

So, to go any further, we need to define “surprising”, for a deterministic system. Fortunately, the concept of statistical entropy again comes to the rescue.

“Surprising” implies incomplete knowledge or nondeterminism. Since at the moment we’re not considering nondeterminism, we need to relax our knowledge requirements. We do this by observing only a part of the system – for instance, by spatial subsampling; temporal subsampling; spatial windowing; temporal windowing; random sampling; *etc...* The point is that, in general, at only a subset of spacetime points is the state known *a priori*. This is perfectly appropriate and reasonable anyway, since we’re dealing with infinite spacetime.

Consider Toffoli’s expression for computation capacity:

$$\lim_{X \rightarrow \infty} \frac{1}{X} \left[\frac{\partial}{\partial T} \log n(X, T) \right]_{T=0} .$$

Consider first the inner expression:

$$\left[\frac{\partial}{\partial T} \log n(X, T) \right]_{T=0} .$$

This is a function of X , the spatial volume of the system. And it is, for that spatial volume, the *initial arrival rate* of new information about the system, as we evolve it through one time step. Now, the whole expression

$$\lim_{X \rightarrow \infty} \frac{1}{X} \left[\frac{\partial}{\partial T} \log n(X, T) \right]_{T=0}$$

can be thought of as the *spatial density* of the inner expression; or the spatial density of the initial arrival rate of new information.

So, let's imagine we have an initial set of state observations at some set of spatial locations, for a particular moment¹². We can treat the value of subsequent observations on the same set of locations as a random variable, and compute its entropy arrival rate. Further, we can divide this entropy rate by the number of locations we're observing, to reach a quantity analogous to Toffoli's computation density.

There are two important questions which may be nagging the reader:

1. How should we treat the subsequent observations as a random variable?
2. Why should the entropy arrival rate be independent of the details of the location set?

The first is easy. Since we don't have any *a priori* information regarding the state outside our location set, we choose the random variable as being uniformly distributed over all states which are consistent with our observations on the location set.

The answer to the second question is it *isn't*: The entropy arrival rate on our location set depends on what the set looks like. For instance, if the location set is a concentrated region larger than the neighborhood¹³, then sites near the middle of the region are initially independent of the sites outside (see figure 6.3). In other words, since information only travels at a finite speed, subsequent values of points near the middle of the region depend only on things we already know (the state *within* the region), so the initial entropy rate at such points is obviously lower than it would be at isolated points.

Let's look at an easy upper bound. Clearly the entropy rate per site is bounded above by the information content of a site, since that's all there is to know about

¹²we can later relax this to be a set of observations of some sites at *possibly different* moments in time

¹³that is, the size of the neighborhood used in computing the subsequent value at any site

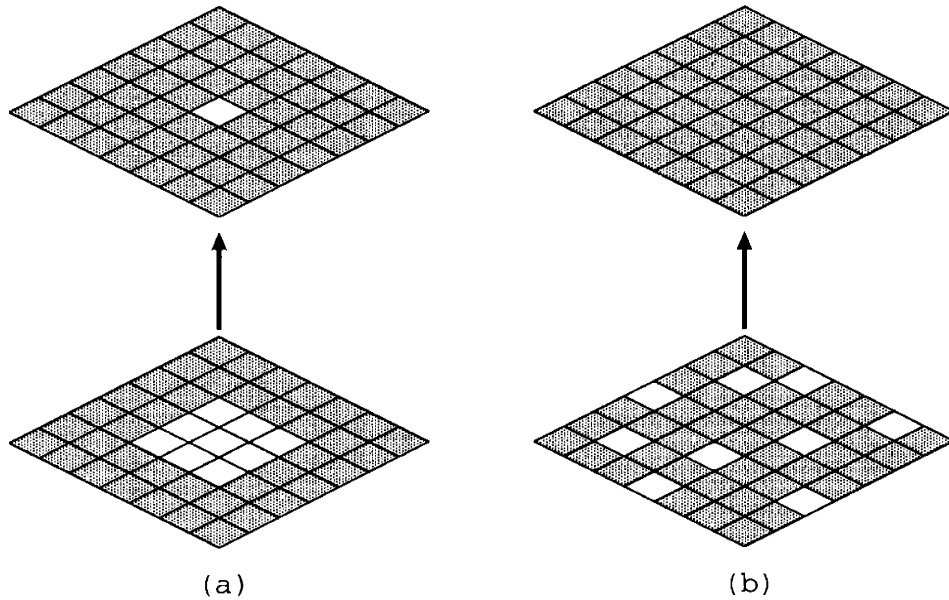


Figure 6.3: Two location sets of the same size, undergoing one timestep. Shaded areas represent unknown state, and unshaded areas represent known state. The location set is the known state in the bottom of each column.

(a): Information about unknown state “flows” into the location set only around the edges. Some elements of the location set (in this case, namely, the center square) receive no information at all in the first timestep. In general, information enters the location set slowly.

(b): Information about unknown state flows into the location set from many more sites, since there is more edge. In general, information enters the location set rapidly.

a site for any point in time (*cf.* the random “dynamics”, above) – No dynamics will be able to sustain this rate of surprise, for an indefinitely long time. So in the case of a real dynamics, if we want the *maximum* conditional entropy rate near the beginning of our observations, we will pick spacetime sites which are causally disconnected, or are loosely connected. This suggests we pick sites which are spacelike separated (*i.e.* further apart in space than they are in time), such as is shown in figure 6.3(b).

This approach also simplifies the calculation greatly, since this bound can be computed by calculating the entropy arrival rate *at a single site*.

Notice that I have been lazy about the transition from a set of observations at a single moment, to a set of observations at distinct spacetime points. This is okay, since because computations can be relocated in time (*cf.* assumption 2, above), as long as our observations are far enough apart in *space*, it doesn’t matter if they are exactly at the same *time*; only that they are causally disconnected (*i.e.* spacelike separated).

So the question becomes: what is the expected entropy rate for a single site? For the moment, I will call this quantity h_1 , and put as my proposal without further justification,

$$C_{\text{stochastic}} \equiv \frac{f}{\Delta V} h_1,$$

for an infinite, dust-like, deterministic medium (*i.e.* a CA)¹⁴.

We will come back to the quantity h_1 in considerably more detail in chapter 8, where we shall relate it to the concept of *specific ergodicity* – a concept developed by Toffoli and Margolus.

¹⁴Note the additional division by ΔV , not reflected in equation 6.6. This is because equation 6.6 assumes X is unitless. In order to account for the actual physical size of the sites, we write $\Delta X = \Delta V \cdot X$, where X is unitless, and ΔX is the total physical size of the automaton. Then division by ΔX instead of X results in an extra factor of $\frac{1}{\Delta V}$.

Chapter 7

Computation Density of Error-prone Systems

7.1 Error-prone systems

Toffoli's analysis is only for deterministic systems. But he gives a hint that, for error-prone systems, one could generalize n to the number of behaviors per space-time volume *due to deliberate programming*, as opposed to "errors" or randomness. Since we wish to evaluate error-prone computers against each other, most important for us now is to learn how to partition computation capacity into useful and non-useful parts – in other words, how to generalize computation capacity and computation density to error-prone systems.

We wish to analyze computation systems in which the output, o , is not uniquely determined by the input i .

To guide us, we have a couple of conditions:

- (i) for a system with error rate zero, computation capacity should degenerate to A_{toff} , and
- (ii) for a system so error-prone that its output is not causally related to its input, computation capacity should go to zero.

In terms of the discussion in chapter 6, we may say the computation density of an error-*free* computer is the Shannon entropy associated with a uniform random choice of transfer function, each of which renders a distinct output state vector. But what of an error-*prone* computer? Clearly if noise is introduced into the system, the effective computation capacity will be smaller; but by how much?

7.2 Functional effectiveness

To address this question, we introduce the concept of *functional effectiveness*. This will allow us to talk about how much *effective* computation we can get out of a function, if it is noisy. The effectiveness of a noisy function should be inversely related to the unpredictability of its output, if we know the ideal function¹; and should be progressively related to the unpredictability of its output, if we don't know the ideal function².

More rigorously, say we have a function f which we can evaluate, but only with error. That is, we have a stochastic function, call it \tilde{g} , over the same domain as f . We wish to be able to quantify *how close* \tilde{g} is to f . Clearly if \tilde{g} is deterministic (equal to the value of some function $g(i)$ on each input i), and if $g(i) = f(i)$ for all i , then \tilde{g} is *effectively* equal to f . That is, a sample of \tilde{g} can be used to compute a sample of f *exactly*. But in the more general case, a sample of \tilde{g} leaves some uncertainty as to the corresponding value of f .

How much uncertainty? If there is no uncertainty at all, we wish to say the *functional effectiveness* of \tilde{g} is equal to that of f . If there is no information at all gained about f from the sample of \tilde{g} , we will say the *functional effectiveness* of \tilde{g} as a model for f is zero. And we wish to have some suitable measure for the intermediate cases – the most interesting cases – in which some information about f is gleaned from taking samples of \tilde{g} .

Let's call the domain D , and for the moment let us assume D is finite. This allows us to talk about vectors instead of functions: We will denote a function $f(\cdot)$ by a vector \vec{f} , of output values taken over the points of D . That is, if $D = \{i_1, i_2, \dots, i_m\}$, then we will denote $f(\cdot)$ by $\vec{f} = \langle f(i_1), f(i_2), f(i_3), \dots, f(i_m) \rangle$.

Now if, as above, we have a function $f(\cdot)$ over a finite domain D , and a stochastic function \tilde{g} , let us define the *functional effectiveness of \tilde{g} as a model for f* as follows:

$$\mathcal{E}(\tilde{g}, f) \equiv H(\tilde{g}) - H(\tilde{g}|\vec{f}), \quad (7.1)$$

where $H(\cdot)$ denotes Shannon entropy³.

If we compare with the informal terminology at the beginning of this section, $H(\tilde{g})$ is the 'usefulness' term, and $H(\tilde{g}|\vec{f})$ is the 'noise' term.

¹let us call this the 'noise' term

²let us call this the 'usefulness' term

³In this paper, when we speak of the Shannon entropy of a stochastic function, we mean the entropy of the vector-valued random variable induced by sampling the stochastic function once at each domain point: $H(\vec{g}) = H(\tilde{g}(i_1), \tilde{g}(i_2), \tilde{g}(i_3), \dots, \tilde{g}(i_m))$, where the $\tilde{g}(i_k)$ are distinct outcomes.

If we rewrite this in a more symmetric form, it immediately suggests itself as the *mutual information* of \tilde{g} and \vec{f} :

$$\mathcal{E}(\tilde{g}, f) = H(\tilde{g}) - H(\tilde{g}|\vec{f}) \quad (7.2)$$

$$= H(\tilde{g}, \vec{f}) - H(\tilde{g}|\vec{f}) - H(\vec{f}|\tilde{g}) \quad (7.3)$$

$$= I(\tilde{g}, \vec{f}) \quad (7.4)$$

This seems reasonable: If $\tilde{g}(\cdot)$ is deterministic and identical to $f(\cdot)$, then the noise term is zero, and $\mathcal{E}(\tilde{g}, f) = H(\tilde{g}) = H(\vec{f})$. Conversely, if samples of \tilde{g} are causally unrelated to (independent of) \vec{f} , then $H(\tilde{g}|\vec{f}) = H(\tilde{g})$, and so $\mathcal{E}(\tilde{g}, f) = 0$.

7.2.1 Useful specializations

One quite restrictive, but useful specialization is when:

- all points of \tilde{g} are independent ($\tilde{g}(i_a)|\tilde{g}(i_b)$, for $a \neq b$)
- all points of \tilde{g} have identical entropies ($H(\tilde{g}(i_a)) = H_{output} = \text{const}$)
- all points of \tilde{g} have identical entropies, conditioned on their input values ($H(\tilde{g}(i_a)|\vec{f}) = H_{output|function} = \text{const}$)

In this case we have

$$\begin{aligned} H(\tilde{g}) &= H(\tilde{g}(i_1), \tilde{g}(i_2), \dots, \tilde{g}(i_m)) \\ &= H(\tilde{g}(i_1)) + H(\tilde{g}(i_2)) + \dots + H(\tilde{g}(i_m)) \\ &= mH_{output}, \end{aligned}$$

and by a similar argument, we have

$$H(\tilde{g}|\vec{f}) = mH_{output|function}.$$

So in this restricted case, we have

$$\mathcal{E}(\tilde{g}, \vec{f}) = m(H_{output} - H_{output|function}). \quad (7.5)$$

7.3 Functional effectiveness as a measure of computation

7.3.1 A proposed measure

computation capacity as a statistical entropy over inputs

Now that we have a relation between computation capacity and entropy (as discussed in chapter 6), and a relation between entropy and functional effectiveness (from the previous section), we can propose and evaluate a generalized definition of computation capacity – for error-prone systems – in terms of functional effectiveness.

Our first step is to encapsulate the computation capacity of a system in terms of a single function. Our approach will be to consider a *computer* as a mapping \mathcal{C} from the space of all (program \otimes input) tuples, to outputs. Of course, all of this will be parametrized by the space and time volume of the computer, as above.

So, we start with⁴:

$$\mathcal{C}_{X,T} : \mathcal{P} \otimes \mathcal{I} \rightarrow \mathcal{O}.$$

This induces a family of transfer functions ω_P , parametrized by the program P , as follows:

$$\omega_P(I) \equiv \mathcal{C}(P, I).$$

Previously, we had $A_{\text{toff}}(X, T) = \log n(X, T)$, where $n(X, T)$ is the number of transfer functions achievable in a spacetime volume (X, T) . We then demonstrated that $A(X, T) = H(\tilde{\omega}_{X,T})$, where $\tilde{\omega}_{X,T}$ is (the vector associated with) a transfer function chosen uniformly at random from the space of all achievable transfer functions, Ω . However, this relies on knowing which transfer functions are achievable. We can modify the definition slightly, to obviate that requirement: Instead of picking a *transfer function* $\tilde{\omega}$ uniformly at random, we instead pick a *program* \tilde{P} uniformly at random. This produces an induced stochastic function $\omega_{\tilde{P}}(I) = \mathcal{C}(\tilde{P}, I)$. We can measure the entropy of $\omega_{\tilde{P}}$ by realizing that the entropy of a vector-valued random variable is simply the entropy of the joint probability distributions of its components:

$$H(\omega_{\tilde{P}}) = H(\omega_{\tilde{P}}(i_1), \omega_{\tilde{P}}(i_2), \dots, \omega_{\tilde{P}}(i_m)).$$

⁴In the following discussion, we will often drop the X and T subscripting, and leave it implicit.

We will propose this as a more generalized measure of computation capacity:

$$A_{\text{proposed}}(\mathcal{C}_{X,T}) = H(\omega_{\tilde{P}}).$$

Clearly, since each transfer function is a result of a distinct program, the entropy of the transfer function of a randomly chosen program is at most the entropy of a randomly chosen transfer function:

$$H(\omega_{\tilde{P}}) \leq H(\tilde{\omega}).$$

functional effectiveness of a perturbed computer

To treat capacity as a question of functional effectiveness, we first introduce an error-prone computer which is a perturbation of \mathcal{C} – let’s call it $\tilde{\mathcal{C}}$. $\tilde{\mathcal{C}}$ will have the same domain and range as \mathcal{C} , but it is an error-prone computer which we propose as a model for \mathcal{C} :

$$\tilde{\mathcal{C}}_{X,T} : \mathcal{P} \otimes \mathcal{I} \rightarrow \mathcal{O}.$$

Now, just as we had the induced family of transfer functions ω_P , above, we can find an induced family of error-prone transfer functions, defined as follows:

$$\tilde{\omega}_P(I) \equiv \tilde{\mathcal{C}}(P, I).$$

Now, again taking \tilde{P} uniformly at random, we can propose the computation capacity of $\tilde{\mathcal{C}}$ as being the functional effectiveness of $\tilde{\omega}_{\tilde{P}}$ as a model for $\omega_{\tilde{P}}$:

$$\begin{aligned} A_{\text{proposed}}(\tilde{\mathcal{C}}_{X,T}) &\equiv \mathcal{E}(\tilde{\omega}_{\tilde{P}}, \omega_{\tilde{P}}) \\ &= H(\tilde{\omega}_{\tilde{P}}) - H(\tilde{\omega}_{\tilde{P}} | \omega_{\tilde{P}}). \end{aligned}$$

7.3.2 Discussion

reduction to A_{toff} in deterministic case

In the deterministic case, the model $\tilde{\omega}_{\tilde{P}}$ is identical to the ideal $\omega_{\tilde{P}}$. So $H(\omega_{\tilde{P}} | \tilde{\omega}_{\tilde{P}}) = H(\tilde{\omega}_{\tilde{P}} | \omega_{\tilde{P}}) = 0$, so the effectiveness of the model is just the entropy in choice of function:

$$\begin{aligned} A_{\text{proposed}}(\mathcal{C}_{X,T}) &= \mathcal{E}(\tilde{\omega}_{\tilde{P}}, \omega_{\tilde{P}}) \\ &= H(\omega_{\tilde{P}}) - H(\omega_{\tilde{P}} | \tilde{\omega}_{\tilde{P}}) \\ &= H(\omega_{\tilde{P}}) \\ &= A_{\text{toff}}(X, T) \end{aligned}$$

zero in completely random case

In the case where the output of the computer is completely uncorrelated with (*i.e.*: statistically independent of) the input, we have $H(\tilde{\omega}_{\bar{P}}|\omega_{\bar{P}}) = H(\tilde{\omega}_{\bar{P}})$, and so

$$\begin{aligned} A_{\text{proposed}}(\mathcal{C}_{X,T}) &= \mathcal{E}(\tilde{\omega}_{\bar{P}}, \omega_{\bar{P}}) \\ &= H(\tilde{\omega}_{\bar{P}}) - H(\tilde{\omega}_{\bar{P}}|\omega_{\bar{P}}) \\ &= 0 \end{aligned}$$

Recall our criteria at the beginning of this chapter. These two boundary conditions illustrate that our proposed measure meets the criteria.

More examples

We will continue to use our friend the PROM, but now with different types of error behavior, in order to illustrate that our new definition of A is both reasonable, and general enough to be useful.

Example i: the base case: Recall our PROM has k address lines and m output lines. The number of programming bits is thus $m \cdot 2^k$, and the number of programs is $2^{m \cdot 2^k}$. Each program results in a distinct transfer function, so $H(\omega_{\bar{P}}) = m \cdot 2^k$.

Example ii: some output bits are stuck: Let us imagine that in our perturbed PROM $\tilde{\mathcal{C}}$, some of the m output lines are permanently stuck at some value, and that only m' of the output lines are still functioning normally. In this case, only the m' programming bits of each row which correspond with the working output lines make any difference at all to the transfer function. This induces a partition of the program space into $2^{(m-m') \cdot 2^k}$ equal-sized partitions, each of size $2^{m' \cdot 2^k}$, such that all programs in any partition give rise to the same transfer function. Hence $\tilde{\omega}_{\bar{P}}$ is uniformly distributed, but over a set of size $2^{m' \cdot 2^k}$, and $H(\tilde{\omega}_{\bar{P}}) = m' \cdot 2^k$. Now, since $\tilde{\omega}_{\bar{P}}$ is completely determined by $\omega_{\bar{P}}$, we can conclude that $H(\tilde{\omega}_{\bar{P}}|\omega_{\bar{P}}) = 0$. So finally,

$$\begin{aligned} A(\tilde{\mathcal{C}}) &= H(\tilde{\omega}_{\bar{P}}) - H(\tilde{\omega}_{\bar{P}}|\omega_{\bar{P}}) \\ &= H(\tilde{\omega}_{\bar{P}}) \\ &= m' \cdot 2^k. \end{aligned}$$

That is, the computation capacity of the PROM is reduced by a factor of $\frac{m'}{m}$, as we would expect.

Example iii: some (fixed set of) output bits are random: Let us say m' outputs still function correctly, but the remainder, instead of being stuck, are random (with some distribution independent of the input). We take a sample of the perturbed transfer function $\tilde{\omega}$, by applying each input and collecting the vector of outputs. We compute the entropy of $\tilde{\omega}$ by the entropy of the joint distribution of the components of $\tilde{\omega}$. Each component of $\tilde{\omega}$ has m' bits given by the correctly functioning outputs, and $m - m'$ bits which are independently distributed, according to the same distribution for each sample (let us call this distribution f_1). So the joint entropy of all components of $\tilde{\omega}$ is the joint entropy of all the bits given by correctly functioning outputs, plus 2^k times the entropy of the distribution of a sample of f_1 :

$$H(\tilde{\omega}_{\bar{p}}) = m' \cdot 2^k + h_1 \cdot 2^k,$$

where h_1 denotes the entropy of f_1 .

Now, to compute $H(\tilde{\omega}_{\bar{p}}|\omega_{\bar{p}})$, we make the following observation: If we know $\omega_{\bar{p}}$, then we know m' bits of each sample of $\tilde{\omega}_{\bar{p}}$. The remaining $m - m'$ bits in each component of $\tilde{\omega}_{\bar{p}}$ are independent of $\omega_{\bar{p}}$, and independent of each other, and identically distributed with entropy h_1 . So,

$$H(\tilde{\omega}_{\bar{p}}|\omega_{\bar{p}}) = h_1 \cdot 2^k,$$

hence

$$\begin{aligned} A(\tilde{\mathcal{C}}) &= H(\tilde{\omega}_{\bar{p}}) - H(\tilde{\omega}_{\bar{p}}|\omega_{\bar{p}}) \\ &= (m' \cdot 2^k + h_1 \cdot 2^k) - h_1 \cdot 2^k \\ &= m' \cdot 2^k. \end{aligned}$$

Again, the computation capacity of the PROM is reduced by a factor of $\frac{m'}{m}$.

Note that example (ii) above is just a special case of example (iii), in which $h_1 = 0$.

Example iv: all output bits err independently with probability ϵ : In this case, we can treat each output line individually, since they are independent. Since the outputs already have full entropy of 2^k bits each, perturbing the output cannot introduce any additional entropy. So $H(\tilde{\omega}_{\bar{p}}) = m \cdot 2^k$. To compute $H(\tilde{\omega}_{\bar{p}}|\omega_{\bar{p}})$, consider the perturbed output as a vector of $m \cdot 2^k$ bits, each of which matches the corresponding unperturbed bit with probability $1 - \epsilon$. So with probability $1 - \epsilon$ per bit, the perturbation introduces zero entropy into the output

vector, and with probability ϵ per bit, the perturbation introduces an expected entropy of 1 bit into the output vector. $H(\tilde{\omega}_{\tilde{P}}|\omega_{\tilde{P}})$ is thus:

$$H(\tilde{\omega}_{\tilde{P}}|\omega_{\tilde{P}}) = m \cdot 2^k \cdot \epsilon.$$

This gives

$$\begin{aligned} A(\tilde{\mathcal{C}}) &= H(\tilde{\omega}_{\tilde{P}}) - H(\tilde{\omega}_{\tilde{P}}|\omega_{\tilde{P}}) \\ &= m \cdot 2^k - \epsilon \cdot m \cdot 2^k \\ &= (1 - \epsilon) \cdot m \cdot 2^k. \end{aligned}$$

So the computation capacity is reduced in this case by a factor of $1 - \epsilon$. Again, this is not surprising.

7.3.3 Applied to feedback automata

To treat computation capacity of an error-prone feedback automaton as a question of functional effectiveness, we adopt an argument similar to that of section 6.5. We restrict ourselves to the null-input case, and we think of the state evolution ω as a function of time and the program:

$$\omega_X : \mathcal{P} \otimes \{1, 2, 3, \dots, T\} \rightarrow \mathcal{O}.$$

Note that the T subscript has been dropped from the transfer function, ω . This is because the “size” of the computer in the feedback automaton case is no longer a function of T . We should, however, note that the computation density must be linearly related to the operating frequency, f , since f determines how many states the computer can reach in a given amount of time.

Let us choose a noisy experimental model for our computer \mathcal{C}_X . Let us call it $\tilde{\mathcal{C}}_X$, and let $\tilde{\omega}_X(s_0, T)$ be the corresponding stochastic transfer function, defined by applying the given initial state to the “noisy” computer, and evaluating the output state at time T . Since we are considering only the null-input case, the vector $\tilde{\omega}$ has only one component, so we will just equate the vector and its single component.

We may ask: How *effective* is $\tilde{\mathcal{C}}_X$ as a model for \mathcal{C}_X ? It turns out that the functional effectiveness $\mathcal{E}(\tilde{\omega}_X, \omega_X)$ meets the criteria posed above at the beginning of this section, for our generalized definition of computational capacity.

We propose the following formal definition of A , in the nondeterministic feedback automaton case:

$$\begin{aligned} A_{\text{proposed}}(X) &\equiv \mathcal{E}(\tilde{\omega}_X, \omega_X) \\ &= H(\omega_X) - H(\tilde{\omega}_X|\omega_X). \end{aligned}$$

Hence, we can directly compute the corresponding computation density:

$$C_{\text{proposed}} = \frac{1}{\Delta T} \cdot \lim_{X \rightarrow \infty} \frac{1}{\Delta X} [H(\omega_X) - H(\tilde{\omega}_X|\omega_X)].$$

Since $\frac{1}{\Delta T} = f$ and $\Delta X = \Delta V \cdot X$, where ΔV is the physical size of a site, we may put this in the slightly more agreeable form:

$$C_{\text{proposed}} = \frac{f}{\Delta V} \cdot \lim_{X \rightarrow \infty} \frac{1}{X} [H(\omega_X) - H(\tilde{\omega}_X|\omega_X)].$$

We will treat this expression in more detail in the following section.

7.4 Estimating computation density of error-prone media

Let's see how we might be able to estimate or evaluate the expression for the computation density of an infinite, uniform, error prone feedback automaton, in a real-world case. We start with

$$C_{\text{proposed}} = \frac{f}{\Delta V} \cdot \lim_{X \rightarrow \infty} \frac{1}{X} [H(\omega_X) - H(\tilde{\omega}_X | \omega_X)].$$

First, note the two terms inside the limit. We can think of the two terms as the error-free term and the error term, respectively. That is, the $H(\omega_X)$ term measures the total amount of “unpredictability” in the system, due to both errors and computation, given limited observations as discussed in section 6.6; and the $H(\tilde{\omega}_X | \omega_X)$ term measures the amount of unpredictability due to errors alone. So, assuming the limit exists of each term individually, we may write

$$C_{\text{proposed}} = \frac{f}{\Delta V} \cdot \lim_{X \rightarrow \infty} \frac{1}{X} H(\omega_X) - \frac{f}{\Delta V} \cdot \lim_{X \rightarrow \infty} \frac{1}{X} H(\tilde{\omega}_X | \omega_X). \quad (7.6)$$

We will revisit the first term in the following chapter, when we discuss specific ergodicity and its relation to computation capacity. The second term – the error term – we will pay closer attention to now.

Imagine errors occur as discrete events, and that they are sparse in spacetime. That is to say, the rule of the stochastic CA is such that, given any input state at a site and its surrounding neighborhood, a single “correct” outcome state carries most of the probability for the next state at that site. The correct output state may of course depend on the input state; indeed it must if the CA is to do anything remotely interesting. For errors to be sparse in spacetime, this means that the probability of the output state being incorrect is small: $\text{Pr}[\text{error}] \ll 1$.

Let us say the probability of an error at each state transition is δ , and let us say the expected entropy increase per error occurrence is given by H_δ . Since errors are sparse in spacetime, that means they are nearly independent. So we may tally up the total expected entropy per timestep to be introduced into the system thusly:

$$H_{\text{total}} \approx X \cdot h(\delta) \cdot H_\delta.$$

Here, $h(\delta)$ denotes the binary entropy function $h(\delta) \equiv -[\delta \lg \frac{1}{\delta} + (1 - \delta) \lg \frac{1}{1-\delta}]$. Substituting this into the second term in equation 7.6, we obtain

$$C_{\text{proposed}} = \frac{f}{\Delta V} \cdot \lim_{X \rightarrow \infty} \frac{1}{X} H(\omega_X) - \frac{f}{\Delta V} \cdot h(\delta) \cdot H_\delta. \quad (7.7)$$

CHAPTER 7. COMPUTATION DENSITY OF ERROR-PRONE SYSTEMS

So, we have reexpressed almost all the terms in C_{proposed} in terms of *local* properties of the medium. The only term left untreated is the $\lim_{X \rightarrow \infty} \frac{1}{X} H(\omega_X)$ term. That is the subject of the next chapter.

Chapter 8

Computation Density, Information Density, and Specific Ergodicity

8.1 Introduction

In this chapter we will build off the results in chapters 6 and 7 to find a relation between Margolus and Toffoli's *specific ergodicity*, and stochastic computation density, for infinite, dust-like, deterministic systems. Following this, we will generalize the result to infinite, dust-like, stochastic systems.

Let us begin by summarizing what we've done so far:

1. In section 6.2, we generalized A_{toffoli} to $A_{\text{stochastic}}$:

$$A_{\text{stochastic}} \equiv H(\tilde{\omega}) = H(\omega(\tilde{P})).$$

This gave us a basis from which to work on further extensions of the definitions of computation capacity and density, which is most of the work in part II of this thesis.

2. In section 6.5, for feedback automata, we generalized the notion of "behavior" to include time-dependent behavior. We redeveloped the definition of $H(\tilde{\omega})$ from a classical definition of statistical entropy, to an expectation over a time variable which ranges over $[0, \infty >$. We noted a relation between this and the expected size of orbits of the system:

$$A_{\text{stochastic}} = -E_{P \in \mathcal{P}} \left[\lg \frac{1}{\|\text{orbit}(P)\|} I_{\text{orbit}(P)} \right],$$

and said we would come back to the topic later, which we now do.

3. In section 6.6, for infinite dust-like systems, we redeveloped C_{toffoli} from a spatial derivative of a time derivative, to the initial entropy production rate per site, subject to a suitable *a priori* probability distribution:

$$C_{\text{stochastic}} = \frac{f}{\Delta V} h_1.$$

The problem with (2) is that it only applies thus far to finite systems.; and the problem with (3) is that it hasn't been evaluated for feedback systems. In this chapter we will consider a CA, which is *both* an infinite dust-like system, and a feedback system, and see what we can learn about its computation density.

8.2 Review of specific ergodicity

We will start with a finite feedback automaton, and show the relation between its value of $A_{\text{stochastic}}$ and its specific ergodicity η . But first, let us review the definition of specific ergodicity.

Consider a reversible dynamics on a state space \mathcal{P} . The dynamics implicitly partitions \mathcal{P} into a disjoint set of orbits. The intuitive idea behind specific ergodicity is this: If we call a system which visits all of its state space in a single cycle *ergodic*¹, then a system with more smaller cycles is somehow “less ergodic”. Toffoli makes this more concrete as follows:

If we choose a state from \mathcal{P} with random incidence, we can consider that we have chosen two values implicitly: an *orbit*², and an *index* in that orbit. The total amount of information in the state we’ve chosen is the log of the size of the state space:

$$H_{\text{total}} = \lg \|\mathcal{P}\|.$$

Of this total, some of the information is which orbit the state is on, and the remainder is the expected information contained in the index.

To see this, let o denote an orbit chosen uniformly at random, and s denote a state chosen uniformly at random. Also, for any state s , let $o(s)$ denote the orbit on which s lies. Then we may write:

$$\begin{aligned} H_{\text{orbit}} &= - \sum_o \Pr[o] \lg(\Pr[o]) \\ &= - \frac{1}{\|\mathcal{P}\|} \sum_o \|o\| [\lg(\|o\|) - \lg(\|\mathcal{P}\|)] \\ &= - \frac{1}{\|\mathcal{P}\|} \sum_o \|o\| \lg(\|o\|) + \frac{\lg(\|\mathcal{P}\|)}{\|\mathcal{P}\|} \sum_o \|o\| \\ &= \lg(\|\mathcal{P}\|) - \frac{1}{\|\mathcal{P}\|} \sum_o \|o\| \lg(\|o\|), \end{aligned}$$

and

$$\begin{aligned} H_{\text{index}} &= \frac{1}{\|\mathcal{P}\|} \sum_s \lg(\|o(s)\|) \\ &= \frac{1}{\|\mathcal{P}\|} \sum_o \|o\| \lg(\|o\|), \end{aligned}$$

¹note this is *not* the mathematical definition of ergodicity of a dynamics on a continuous state space

²or an attractor basin, in the case of an irreversible dynamics

Note that $H_{\text{index}} + H_{\text{orbit}} = H_{\text{total}}$, as one would expect.

We now define the *specific ergodicity* of the dynamics as

$$\eta \equiv \frac{H_{\text{index}}}{H_{\text{total}}}.$$

We can think of η as the portion of information in a randomly chosen state, which specifies only the *index* of the state on its orbit.

Let us examine some extreme cases: If there is only one orbit and it covers the entire state space, then $H_{\text{index}} = H_{\text{total}}$, and $\eta = 1$. In this case the system is completely ergodic. If at the other extreme, each state is its own trivial orbit, then $H_{\text{index}} = 0$, hence $\eta = 0$. In this case the system is as far from ergodic as is possible.

8.3 Finite feedback automata

Recall from section 6.5, our proposed definitions of computation capacity and computation density for finite feedback automata:

$$A_{\text{stochastic}} = -E_{P \in \mathcal{P}} \left[\lg \frac{1}{\|\text{orbit}(P)\|} I_{\text{orbit}}(P) \right],$$

and

$$C_{\text{stochastic}} = -f \cdot \lim_{X \rightarrow \infty} \frac{1}{X} E_{P \in \mathcal{P}} \left[\lg \frac{1}{\|\text{orbit}(P)\|} I_{\text{orbit}}(P) \right].$$

Let us rewrite the definition of $A_{\text{stochastic}}$ in the terminology of section 8.2, above. We will first treat the case of a reversible automaton (that is, an automaton whose dynamics is invertible), and then give a sketch of an argument for an irreversible automaton.

case i: reversible automaton For a *reversible* automaton, I_{orbit} is always 1. So

$$\begin{aligned} A_{\text{stochastic}} &= -E_{P \in \mathcal{P}} \left[\lg \frac{1}{\|\text{orbit}(P)\|} \right] \\ &= E_{P \in \mathcal{P}} \left[\lg \|\text{orbit}(P)\| \right]. \end{aligned}$$

Rewritten in terms of section 8.2,

$$A_{\text{stochastic}} = \frac{1}{\|\mathcal{P}\|} \sum_s \lg \|\text{o}(s)\|.$$

But notice this is exactly H_{index} ! So we can write

$$\begin{aligned} A_{\text{stochastic}} &= H_{\text{index}} \\ &= \eta H_{\text{total}}, \end{aligned}$$

where H_{total} , as always, denotes the total amount of state information of the system.

This of course is valid for finite CA's also, but in this case we can say something more about the total state information: Since the automaton is made up of X equivalent cells, each containing the same amount of initial state information (let's call it H_1), we have $H_{\text{total}} = XH_1$. So in this case

$$A_{\text{stochastic}} = XH_1\eta,$$

and

$$C_{\text{stochastic}} = \frac{f}{\Delta V} H_1\eta.$$

case ii: irreversible automaton For an irreversible automaton, the specific ergodicity is defined slightly differently. D'Souza [23] takes as a starting assumption that the number of states which should be counted from any starting state in the *irreversible* case is the total number of states *reachable from* the starting state. Since the set of reachable states is the orbit, plus some initial transient, we may write the set of states reachable from s as

$$r(s) \equiv o(s) \cup t(s),$$

where $o(s)$ is the orbit, and $t(s)$ is the transient. D'Souza then defines the specific ergodicity as $\frac{H'_{\text{index}}}{H_{\text{total}}}$, where H'_{index} has the modified definition:

$$H'_{\text{index}} \equiv \frac{1}{\|\mathcal{P}\|} \sum_s \lg(\|r(s)\|).$$

Clearly, $H'_{\text{index}} \geq H_{\text{index}}$, since $\|r(s)\| \geq \|o(s)\|$ for any s .

We may now revisit our expression for $A_{\text{stochastic}}$ in the irreversible case. In particular, we can easily show it is *less* than $XH_1\eta$:

$$A_{\text{stochastic}} = -E_{P \in \mathcal{P}} \left[\lg \frac{1}{\|\text{orbit}(P)\|} I_{\text{orbit}(P)} \right]$$

$$\begin{aligned}
 &\leq -E_{P \in \mathcal{P}} \left[\lg \frac{1}{\|\text{orbit}(P)\|} \right] \\
 &= \frac{1}{\|\mathcal{P}\|} \sum_s \lg \|o(s)\| \\
 &\leq \frac{1}{\|\mathcal{P}\|} \sum_s \lg \|r(s)\| \\
 &= H'_{\text{index}}
 \end{aligned}$$

So in the irreversible case we have $A_{\text{stochastic}} \leq H'_{\text{index}} = \eta H_{\text{total}} = X H_1 \eta$, and correspondingly $C_{\text{stochastic}} \leq \frac{f}{\Delta V} H_1 \eta$. In other words, introducing irreversibility into a system by “kicking out” some states from orbits, onto transients, can only have a *negative* impact on computation capacity.

Recall the goal of Part II of this thesis: we seek an *upper* bound on computation capacity and computation density. Since reversible automata are “best in class”, it will suffice to treat only the reversible case in the search for an upper bound. For brevity and clarity, from this point forward we will consider only reversible automata, allowing us to drop the inequality notation with the understanding that the bound also holds for irreversible automata.

8.4 Infinite feedback automata (CA’s)

Note that the two expressions for $A_{\text{stochastic}}$ and $C_{\text{stochastic}}$, above, both implicitly depend on X , the size of the CA. I have left the subscripts out throughout this chapter, for brevity, but in general we must write

$$A_{\text{stochastic}}(X) = X H_1 \eta(X),$$

and

$$C_{\text{stochastic}}(X) = \frac{f}{\Delta V} H_1 \eta(X).$$

If η actually approaches some fixed value as $X \rightarrow \infty$, then of course $C(X)$ will also approach a fixed value asymptotically: If

$$\lim_{X \rightarrow \infty} \eta(X) = \eta$$

then

$$\lim_{X \rightarrow \infty} C_{\text{stochastic}}(X) = \frac{f}{\Delta V} H_1 \eta. \tag{8.1}$$

Now, in [74], Toffoli presents numerical evidence that η may actually converge for almost all CA's. To prove this is, as far as I know, an open problem. What can be said is that, if this conjecture is true, then the limit in equation 8.1 exists; hence the medium has a well-defined computation density, given by

$$C_{\text{stochastic}} = \frac{f}{\Delta V} \eta H_1. \quad (8.2)$$

8.5 Error-prone infinite feedback automata (stochastic CA's)

We can combine the results from section 7.4 and section 8.4 to arrive at an expression for computation density of stochastic, infinite, dust-like feedback automaton (*i.e.* a randomized cellular automaton):

We substitute the expression for $C_{\text{stochastic}}$ from equation 8.2 for the deterministic term in equation 7.7, yielding

$$C_{\text{stochastic}} \approx \frac{f}{\Delta V} \eta H_1 - \frac{f}{\Delta V} \cdot h(\delta) \cdot H_\delta \quad (8.3)$$

$$= \frac{f}{\Delta V} [\eta H_1 - h(\delta) \cdot H_\delta] \quad (8.4)$$

for a stochastic CA in the sparse error limit.

8.5.1 Discussion

Let's take a moment to reflect on this equation. The two principle objectives of part II of this thesis are

1. to arrive at an expression for computation density of an error-prone, infinite, unstructured medium, and
2. to recast the expression entirely in terms of other *intensive* properties, which are measurable.

It appears that we have reached this goal; in 8.4 we have $C_{\text{stochastic}}$ for such a medium, in terms which are both easily understood and easily measured³. With

³Perhaps with the exception of η , which may be difficult to measure in a reasonable amount of time. Keep in mind, however, that η is bounded above by 1, so putting an *upper* bound on computation density doesn't require an explicit evaluation of η .

this as basis, we can form a model for comparing many various types of computational media. We will do some of this comparison in the next chapter, between DNA computers and electronic computers. But for now, think about equation 8.4 more broadly, as a good starting point for measurement and comparison of computational systems of many disparate – and seemingly incomparable – types.

8.5.2 Examples

amorphous computers, or “gunk” : Abelson *et al.* [1] have proposed a model of computation comprising massive numbers of unsynchronized, randomly distributed, identically programmed, error-prone computational elements. These elements can each communicate to their “neighbors” ; that is, the other elements within a certain range; but they must do so isotropically – there is no *a priori* routing. Typical algorithms in this model involve a “discovery” phase, in which all processors investigate the topology of their local connections; and gradient-like indices are set up by means of message passing algorithms, by which individual processors may estimate their global position. This location process allows for the establishment of a kind of “virtual routing” structure.

The typical algorithm then proceeds to the next phase, which I will call the “execution” phase, in which the problem at hand is solved – usually with greatly increased simplicity, speed, and efficiency, thanks to the virtual routing structure laid out in the discovery phase.

Certainly this model appears amenable to analysis using our measure of computation density: the information density is easily computable, the cycle time is known, and presumably the error rate and H_δ are easy to measure or estimate. The problem remaining is to estimate or measure η . Keep in mind: it is trivial to bound η , since it may not be greater than one. But in many cases, especially those with very large H_1 and low communication bandwidth, η may be very small indeed. In fact, for most if not all algorithms which have been written for this computation model, H_1 is much, much larger than the typical message size passed from peer to peer.

In any case, clearly this is fertile ground for future research.

***in vitro* biocomputers:** *In vitro* biocomputers – in particular, DNA computers – are quite amenable to analysis using the techniques developed here. In fact, DNA computation (and *in vivo* biocomputation – see below) was one of the original motivations for pursuing this line of inquiry.

In chapter 3, we gave some analysis and results on DNA computation. It will be the goal of the next chapter to revisit those results in more detail, and to compare them with performance results of typical electronic computers.

***in vivo* biocomputers:** In addition to the brief (and somewhat sloppy) analysis of a hypothetical protein interaction computational model given in chapter 3, we might envision other uses for the theory given here, in analyzing and predicting results from *in vivo* biocomputation efforts.

Recent work in functional genomics and molecular cell biology suggests that functional complementation of cells becomes increasingly difficult to achieve, as the computational complexity of the complementation increases. It is interesting to speculate whether wild-type cells have, in fact, adapted to using *all* their computational resources for self-maintenance and for reproduction. If so, this would suggest that any functional complementation to wild-type cells is likely to reduce evolutionary fitness relative to unmodified cells.

Theoretical analysis of this question might be attempted through a differential, or “small signal” analysis of variation of computation density with signal density, in a closed reactor cell. And in turn, experimental verification and suggestions for theory might be garnered from work on “minimal organisms”, currently being done in the Knight lab [43] and elsewhere.

CA machines / high density array processors: It is also worth noting that, in addition to applications of this theory to more “outrageous” models of computation such as biocomputation and amorphous computation, it is also applicable to slightly more mainstream hardware – such as Cellular Automaton machines, high density SIMD array processors, FPGA’s, *etc.* Application of this theory to such systems gives us an objective basis for comparing theoretical maximum performance of particular instances of these various types of machines.

It also seems likely that marginal analysis – suggested above for biocomputers – might also be applicable to analysis of distributed hardware resource allocation problems. Examples might include:

- predicting optimal allocation of real-estate for routing resources *vs.* computation resources in FPGA cell design, to maximize customer benefit
- predicting optimal allocation of real-estate for memory resources *vs.* computation resources in hybrid memory/high density gate array chip designs, such as those described in [50] and [49]

- optimizing the tradeoff between transistor density and error rate, in extremely high density VLSI designs

Chapter 9

Conclusions

Part I of this thesis was concerned with estimating the performance envelope of chemical computers. Part II has been concerned with the fungibility of computation, and of establishing a single, intensive valuation function for all computational media. I have argued that computation density is such a valuation function.

This final chapter will attempt to synthesize these two parts, by establishing limits on the computation density of chemical computers.

9.1 Review of part II

We begin with a brief review of what we've done so far in part II:

1. In chapter 6, we rephrased Toffoli's counting formula for computation capacity into a statistical expectation (a statistical entropy, in fact). The results are valid for finite, deterministic, combinatorial systems.
2. In sections 6.5 *et seq.*, we extended this statistical definition of computation capacity from combinatorial functions, to feedback automata. The results in this case are valid for finite, deterministic feedback automata.
3. Finally, in section 6.6, we further extended the statistical definition of computation capacity from a global (extensive) to a local (intensive) definition, in an attempt to establish a measure of computation density¹ for infinite, unstructured (uniform) systems. The results in this chapter are valid for infinite, deterministic, uniform feedback automata (CA's).
4. Moving on, in chapter 7, we extended the statistical definition of computation capacity from deterministic to nondeterministic systems. The results here are valid for finite, error-prone combinatorial systems.
5. In sections 7.3 *et seq.*, we applied similar generalizations to (2) and (3), above, to establish a measure of computation density for infinite, error-prone feedback automata.
6. And finally, in chapter 8, we have shown a relation between Toffoli's "specific ergodicity", information density, and our computation density of infinite uniform systems. This allows us to state computation density of infinite uniform systems both deterministic and stochastic, *entirely* in terms of intensive quantities.

¹stated entirely in terms of intensive quantities

All the above work has put us in a position to estimate the computation density of infinite, uniform, error-prone computation systems; in fact, this has been the ultimate goal of part II of this work, up to and including chapter 2.

Now we will move on to more specific applications: first to chemical computers in general, and finally to DNA computers. This last effort will allow us to compare the performance of DNA computers with that of their more conventional cousins – modern electronic computers – on a firmer theoretical footing than previously possible.

9.2 Computation density of a chemical computer

Recall our analysis of the performance envelope of chemical computers from chapter 2. There, we computed bounds on f and $\langle \varepsilon \rangle$ as a function of N , L , and ΔV . This gave roughly the shape of the performance envelope, but didn't directly address the issue of comparability or valuation of radically different computers. By contrast, the entire goal of part II of this thesis has been comparability: We now have in hand a reasonable measure of the specific value of a computational medium – the computation density. So it is here, finally, that we can use the fruits of part II to put a valuation on chemical computation media.

To review: We treat a chemical computing medium as a stochastic cellular automaton, via the embedding outlined in chapter 2. The fixed parameters of the medium are determined by the chemistry we are using. The variable parameters, or “design parameters”, are chosen by us as we embed a particular computational model into our chemistry. These design parameters will be, for the purposes of this discussion, N , L , and ΔV , or equivalently, N , L , α , and r .

We begin by combining the performance bounds on f and $\langle \varepsilon \rangle$ – as developed in chapter 2 – with the valuation equation: equation 8.4 from section 8.5.

The valuation equation for chemical computers is:

$$C = \frac{f}{\Delta V} [\eta H_1 - h(\delta) \cdot H_\delta].$$

We will assume $\eta = 1$ for simplicity², so in this case

$$C = \frac{f}{\Delta V} [H_1 - h(\delta) \cdot H_\delta].$$

²at any rate, η cannot be larger than 1

Recall H_1 is the amount of stored information at each site. For N binary signals per site, as in our chemical computation model, H_1 is simply the number of signals:

$$H_1 = N.$$

Next, recall $h(\delta) \cdot H_\delta$ is the expected number of bits of error per cell, per time step. Since there are N signals, and each signal represents one bit of information, this is approximately $N \cdot h(\delta) \cdot 1$ (see note ³). So

$$C = \frac{f}{\Delta V} N[1 - h(\delta)].$$

Now to combine the results: We substitute our expression for f from section 2.7, to yield

$$C = \frac{f}{\Delta V} N[1 - h(\delta)] \tag{9.1}$$

$$\leq ND_L(k\Delta V)^{-2/3}(\Delta V)^{-1}(1 - h(\delta)) \tag{9.2}$$

$$= ND_1L^{-\gamma}k^{-2/3}(\Delta V)^{-5/3}(1 - h(\delta)). \tag{9.3}$$

Referring to section 2.4, we have $\Delta V \geq \frac{Nrv_L}{\alpha}$ from equation 2.10, so

$$\Delta V^{-1} \leq \frac{\alpha}{Nrv_L} = \frac{\alpha}{Nrv_1L^{3\gamma}},$$

or

$$\Delta V^{-5/3} \leq \left(\frac{\alpha}{Nrv_1}\right)^{5/3}L^{-5\gamma}.$$

Hence we can express a bound on C for a chemical computer, in terms of the design parameters N , α , r , and L :

$$\begin{aligned} C &\leq N^{-2/3}D_1L^{-6\gamma}k^{-2/3}\left(\frac{\alpha}{rv_1}\right)^{5/3}(1 - h(\delta)) \\ &= D_1(Nk)^{-2/3}\left(\frac{\alpha}{rv_1}\right)^{5/3}L^{-6\gamma}(1 - h(\delta)). \end{aligned}$$

Recall that here, δ is the specific error rate per site per timestep. In particular, it is bounded below by $N \langle \varepsilon \rangle$. So in our mean-field approximation in which we replace ε by its expectation over signals, we obtain an approximate bound on C as:

$$C \leq D_1(Nk)^{-2/3}\left(\frac{\alpha}{rv_1}\right)^{5/3}L^{-6\gamma}(1 - h(N \langle \varepsilon \rangle)). \tag{9.4}$$

³in the limit in which errors are sparse in spacetime, hence nearly independent

9.2.1 Discussion

Several things are worth noting about equation 9.4:

- C scales *down* with N
- C scales *down* with r
- C scales *down* with L

These three, taken together, indicate that trying to improve the computation density of a chemical computing medium by increasing N , L , or r , is doomed to failure. As we argued before in section 6.4, the key to solving large problems is applying enough error correction to the computation, to guarantee a high probability that the computation will complete without encountering an uncorrectable error. Doing this requires coupling; and as we stated earlier, coupling – while necessary for solving large problems – actually *reduces* the computation capacity of the medium. A similar corollary obtains here: To solve large problems, we must reduce the probability of uncorrectable errors. To do so, we must increase either N , L , or r . But doing so, while it may reduce error probability, will also necessarily reduce computation density.

A couple of other things worth noting about equation 9.4 are:

- **α is bounded:** Since α cannot, by definition, be larger than 1, increasing α without bound to improve computation density is not physically realizable.
- **For small error rates, C is not heavily dependent on $\langle \varepsilon \rangle$:** If the average expected error rate $\langle \varepsilon \rangle$ is quite small, note that C is not affected very severely. This makes sense, for a couple of reasons:
 - in the presence of mildly imperfect operation, application of a small amount of error correction goes a long way; and
 - we would expect that computation capacity should be a continuous function of the design parameters – it is *prima facie* disbelievable that there would somehow be a discontinuity in computation capacity as the error probability went from zero to some very small value.

r	L	N	ΔV (μm^3)	f (Hz)	$\langle \varepsilon \rangle$ bound (conjecture)	C ($\frac{\text{bits}}{\text{s} \cdot \mu\text{m}^3}$)
10	40	100	0.7	9.5	10^{-440}	1400
4	10	100	0.022	220	10^{-32}	$1 \cdot 10^6$
4	7	100	0.012	400	10^{-18}	$3.3 \cdot 10^6$
1	720	2^{70}	$1.4 \cdot 10^{20}$	$5 \cdot 10^{-20}$	≈ 0	$4.2 \cdot 10^{-19}$
1	14	10000	1.0	13	$3 \cdot 10^{-9}$	$1.3 \cdot 10^5$
1	12	1000	0.075	85	$3 \cdot 10^{-9}$	$1.1 \cdot 10^6$
1	9	16	0.00075	2200	$3 \cdot 10^{-9}$	$4.7 \cdot 10^7$

Table 9.1: Bound on computation density for DNA computers with various values of r , N , ΔV , and L . α is assumed to be 0.0006 in all cases.

9.2.2 DNA computers revisited

In section 3.1.6, we discussed the shape and size of the performance envelope of DNA computers. This came in the form of a functional dependence of f and $\langle \varepsilon \rangle$ on the design parameters of the computer: r , L , N , and ΔV . Recall, the results of these computations for a few different design points were shown in table 3.1.

Now, let us augment table 3.1 with another column, in which we evaluate the computation density for each row in the table. We have:

$$C = \frac{f}{\Delta V} N [1 - h(N \langle \varepsilon \rangle)].$$

Since the error rate is small for the cases listed in the table, we will just consider for the moment that $h(N \langle \varepsilon \rangle) \approx 0$. Evaluation for the first case proceeds as follows:

$$\begin{aligned} C &\leq \frac{9.5\text{Hz}}{0.7\mu\text{m}^3} (100) \\ &\approx 1400 \frac{\text{bits}}{\text{s} \cdot \mu\text{m}^3}. \end{aligned}$$

And so forth for the other rows in the table.

The augmented table is shown in figure 9.1.

9.2.3 Comparison with electronic computers

Before we compare these performance limits with conventional electronic computing media, a couple of notes are in order:

dimensionality of integration: In typical electronic integrated circuits, the dimensionality of the medium is for all intents and purposes, two. This is for several reasons:

1. such circuits are produced using lithographic processes, hence are inherently two dimensional. Integration in the third dimension would require a method of routing signals in the third dimension, as well as a method of fabricating with high density in the third dimension.
2. such circuits typically dissipate (relatively) large amounts of energy per equivalent gate operation. This gives rise to a cooling problem – in typical cases we must dedicate the third dimension to the removal of heat⁴. For more in-depth discussion of this and related issues, consult [25], [14], [51], *etc.*

narrowness of optimal design region: If we review the sets of design parameters given above in section 9.2.2, it is intuitively apparent that as we reduce r and L , the error rate is effectively zero for much of the time, and then rises *rapidly* to an unacceptably high level. This is because, if we are trying to solve extremely large problems, we have two competing goals: We want speed, but we also want a *lower* error rate as the problems become larger. The combination of these factors makes the cutoff of the “acceptable” region of design parameters very narrow: By the time the computer becomes fast enough, it is also starting to have a very high error rate. This effect will already be familiar to those who have worked on extremely high density, high speed VLSI circuit design.

programmability: In the chemical computer case, there are no dedicated routing resources; whereas for instance in a high density electronic Cellular Automaton machine, there are most likely at least one – and possible more – dedicated and distinct routing channels for each spatial dimension⁵. The lack of such dedicated

⁴There are several levels at which the heat dissipation problem is relevant. So-called “adiabatic circuit design” [82], [77], [26] avoids, to the extent possible, passing current across any nontrivial voltage difference. This technique, by avoiding commutation power, holds the promise of minimizing switching power dissipation to near the thermodynamic limit.

At a finer level is a fundamental thermodynamic limitation on irreversible computation processes, given by Landauer [44]. This limit states that any logically irreversible operation *must* dissipate free energy at least equal to $k_B \ln 2$ per bit lost in the computation.

⁵or more general routing structures, such as routing which behaves locally as hypercube routing, *etc.*

routing channels can be thought of as a source of information loss, or dissipation. For instance, if we compare a two-dimensional system in which each site has four *distinct* information channels – each coming from one ordinal direction and carrying one bit per timestep – *vs.* a system which has four *indistinct* information channels, each coming from one direction and carrying one bit per timestep, but which are indistinguishable, we can calculate that we have lost more than half of the incoming information at each site, per timestep⁶.

On the one hand, this makes the problem of programming a chemical computer very foreign to us; developing algorithms for such computers is likely to be difficult. On the other hand, presumably the anisotropy in routing in the electronic case comes at some cost – wires occupy space; they are not free. An interesting area for further research would be to delve more deeply into this implicit tradeoff: Certainly anisotropic (“routable”) computers can simulate isotropic (“nonroutable”) computers. Presumably the reverse is also true; but the question is: at what cost to computation density? A more rigorous treatment of the tradeoffs in computation density between routable and nonroutable systems would be a theoretically interesting problem to tackle, and would also have direct application to the theory of programming chemical computers.

numbers: Let’s try and put some approximate numbers to this comparison.

We will take $1\mu\text{m}^2$ as the size of a transistor in an electronic computing medium. We will assume the switching time of such a transistor is on the order of 1ns. We will assume planes of transistors may be stacked (for instance, in liquid-cooled stacks), with an interplanar spacing of 10mm. Lastly, we assume a logic gate of unit computation capacity requires four transistors to implement. All of these assumptions should be regarded as pessimistic for modern electronic computational media.

We compute the computation density thusly: The cell size is

$$\begin{aligned}\Delta V &= 4 \cdot 1\mu\text{m}^2 \cdot 10\text{mm} \\ &= 40000\mu\text{m}^3.\end{aligned}$$

⁶Let’s assume each information channel is modeled by a Bernoulli trial per timestep, in the absence of any *a priori* information. Then the incoming information in the distinguishable-channel case is four bits per timestep. But the indistinguishable-channel case has only five possible outcomes: {0, 1, 2, 3, 4}, corresponding to the number of ones arriving at each timestep. The probability distribution is binomial: $\frac{1}{16} \cdot \{1, 4, 6, 4, 1\}$. The incoming information is approximately 1.86 bits. So we have lost about 2.14 bits, or 53% of the information.

And the number of signals per cell (N) is one. So

$$\begin{aligned} C_{\text{electronic}} &= \frac{1\text{GHz}}{40000\mu\text{m}^3} \cdot (1\text{bit}) \\ &= 2.5 \cdot 10^{-5} \frac{\text{bits}}{\text{ns} \cdot \mu\text{m}^3}. \end{aligned}$$

conclusion: The best computation density obtainable from the idealized DNA computers in table 9.1 is about $0.047 \frac{\text{bits}}{\text{ns} \cdot \mu\text{m}^3}$ (see note⁷). If we set α all the way up to 0.05, the chemical computer may be able to run about 20 times faster (as discussed in section 3.1.7), giving a capacity of almost $1 \frac{\text{bit}}{\text{ns} \cdot \mu\text{m}^3}$. In contrast, the computation density achieved by commodity desktop workstations is about $2.5 \cdot 10^{-5} \frac{\text{bits}}{\text{ns} \cdot \mu\text{m}^3}$.

So we can conclude that current electronic technology comes within a factor of 40000 of a hypothetical best-case chemical computer using DNA, operating at $\alpha = 0.05$. Since we have been not so generous with our electronic model of computation, and very generous with our chemical model, it is almost certain that the margin is much narrower, even with *current* electronic technology. Add to this the following factors:

- electronics can easily improve by another factor of 100 to 1000 in speed-density terms before statistical limitations become crippling, and
- we have no idea what η actually is for DNA computers, but 1 is certainly very generous.

Given these factors, we can conclude that chemical computers and electronic computers are closer to equivalent in maximum achievable computation density, than is generally appreciated in the chemical computation community.

9.3 Incompleteness of work

This section summarizes the major assumptions and areas of incompleteness in this work. The next section discusses possible areas for generalization and expansion.

⁷note the change of units from the figures quoted in the table

9.3.1 Theoretical treatment of error correction

I have not explicitly treated error correction in this discussion. The believability of the arguments presented here – in particular, the coupling argument in section 6.4 – would be greatly improved by such a treatment.

To do this, one might argue roughly that error correction, giving an exponential reduction in error rate with a linear increase in number of signals used, does not affect the computational density of a computing medium. If one could show that if we have a medium X of computational density A , and we transform X to X' by applying error correction, that the computational density of X' is *at most* A . Perhaps by an argument that transforming X to X' amounts to coarse-graining spacetime, resulting in decreased error rate, but with correspondingly decreased spacetime density.

One might also try to make an argument that my expression for computational density implies that if a “good” [60] family of error correcting codes is used, computational density can be *at most* preserved, because it is asymptotically defined. This certainly would not mean that error correction is worthless; it simply puts it in perspective: the programmer seeking to complete a computation of given finite size T with a probability of error p in a medium of density C must choose to apply error correction of a quality sufficient that her computation will terminate with the correct answer, with probability at least $1 - p$. If the medium is very good, it can complete her computation with high probability with no error correction. The worse the medium is (that is, the smaller C is), the higher the quality of the error correction she must choose, and hence the more of the medium she must use in order to complete the fixed size computation with her desired probability of success.

9.3.2 Enhancement of error model

As noted in our discussion in section 3.1.6, the bound we put on d_{exp} – the expected minimum separation distance between nearest neighbor codewords – could stand to be a lot better. For review, the shortcomings listed there include:

- **crosstalk, false positives, and false negatives:** In our calculation of the free energy of an erroneous binding, we assumed standard concentrations for all signals, and only calculated the difference in free energy for a “crosstalk” type error; that is, an incorrect molecule binding to a signaling site. We did not explicitly account for false positives (*i.e.* a site which should not be bound, is bound), or false negatives (*i.e.* a site which should be bound,

is not bound). As discussed in the conjectures section, this effect probably roughly squares the error probability, but certainly a much more thorough treatment is warranted.

- **sloppy combinatorics in calculating $\langle \epsilon \rangle$ from ϵ :** In calculating the probability of a signal error from the probability of a molecule error, we took a very stringent condition; that is, a signal error occurs if and only if *all* signal molecules for that class are erroneously bound. This also roughly squares the error probability, but again, better analysis is warranted.
- **nonzero conformational entropy:** We have ignored conformational entropy of signal molecules, and conformational entropy is presumably much lower in the correctly bound state than in an unbound or erroneously bound state. Hence, we have substantially overestimated the difference in free energy between a correctly bound state, and an unbound or erroneously bound state. So we have underestimated the error probability. Currently I have no method of estimating the magnitude of this effect.

9.3.3 Weakness in specificity

The model developed in this thesis is cumbersome, to put it mildly. It is quite possible, even likely, that additional insight might lead to a simpler, more elegant, and more general approach to the problem. Such an approach might take the form of a general theory regarding expected predictability of stochastic processes of the form found in chemistry (see section 9.4.7, below).

9.4 Future Work

9.4.1 Gene Expression Logic

If we could develop reasonable insight on scaling of hybridization mismatch energies between DNA and DNA binding proteins, we could probably obtain a similar bound for Gene Expression Logic. The mechanism of this bound may eventually shed light on the reasons underlying the experimental difficulty of “functional complementation” of living organisms.

9.4.2 Abstract models of chemistry

One might formalize this approach for *all* chemistries, by adopting a “toy model” of chemistry, and showing how it applies to real chemistry within some bounds. Ample precedent for this can be found in Hart and Istrail’s work [34] on protein folding in a lattice model. Such a toy model might be a model in which chemicals were represented by oddly-shaped, fully connected cubic sublattices in 3-space.

A dangling question here is how to reduce the blocks model produced in this way, to a formal mathematical abstraction. This would probably entail describing the interaction surfaces as bit strings, defining an interaction energy between pairs of bit strings, and defining the scaling of the “size” property of a molecule in relation to the length of the bit string required to describe the binding site.

9.4.3 Simulations

Given the energy and size derivations from such an abstract chemistry model, one could write the chemical kinetic equations describing the (stochastic) dynamics of blocks model chemical systems. One need not write a complete simulator, since they have already been written by Arkin[11], Lyons *et al.* [46], Gibson[29], and others. One need only write a “logical preprocessor”, which takes as input a desired set of logical relations and a specified signal size, and chooses a set of blocks-model “species” which supposedly interact in the desired logical pattern. It will output as simulation parameters, rate constants for all the reactions between the chosen species. The rate constants will be computed from the implicit temperature, the calculated “size” of the species, and the interaction energies computed from the blocks model.

The theory predicts that the computational density of a blocks-model computational medium is almost-independent of the specific computation, and the specific signals chosen. The following set of experiments could provide numerical support for this hypothesis:

- Run a parallelizable computation on a short, fat computer and a long, thin computer and verify the aggregate error rate is similar.
- Run a parallelizable computation on a short, fat computer and a short, thin computer, and verify the aggregate error rate scales exponentially in thickness.
- Run a parallelizable computation on a long, thin computer and a short, thin computer. Verify the aggregate error rate scales exponentially in length.

- Run a parallelizable computation on a short, thin computer. Note a bad error rate. Then rescale the computer by applying error correction, and rerun the computation. Note a better error rate. Compare the calculated computational capacities of both computers.

9.4.4 Performance bounds for electronic computers

It is conceivable that an analysis similar to that in Part I could be applied to electronic computers. That is, one could define a design tuple which was appropriate for electronic computers, and carry through a similar set of arguments to place bounds on the accessible performance envelope. Such an analysis would involve defining assumptions on the physical construction and mode of operation of the class of computers being considered.

For instance, one might start by formalizing the structure of standard lithographically etched CMOS integrated circuits, and take as the design tuple a set such as:

- minimum feature size
- capacitance density of active circuit elements to substrate
- dopant concentration
- operating voltage

The analysis might proceed as follows:

- compute information storage density from feature size
- compute expected variance in device parameters from feature size and dopant concentration
- compute error probability per gate operation from variance in device parameters and operating voltage (and temperature)
- compute operating speed from feature size, capacitance density, and operating voltage

Then one could combine the resulting bounds on error probability and operating speed into a bound on the computation density of this class of electronic computer, similar to the analysis I have done for chemical computers in Part II.

Other such analyses could be done, of course, for differently defined classes of electronic computers.

9.4.5 More thorough investigation of multicellularity

Taking a closer look at the amount of computation performed by single cells, and trying to carry through the argument suggested in section 4.3, regarding the evolutionary necessity of multicellularity.

9.4.6 Sensitivity analysis of GEL systems

Consider two reactor vessels, one of which is already heavily computationally loaded, and the other of which is not. It is intuitively clear that we can build a better computer in the less-loaded reactor than in the more-loaded. To give more substance to this argument and quantify the sensitivity of chemical computers to “functional supplementation” would be a valuable contribution. Specifically, if cells are already heavily functionally loaded as suggested above in the evolution argument, then it is not surprising that some GEL systems which are predicted to work, simply do not when added to a cell’s existing gene expression mechanism.

9.4.7 Generalization to arbitrary stochastic processes

A much more general goal is suggested by interpreting this work in a much broader context: That of stochastic processes. More specifically, to try and quantify the amount of useful computation which can be performed by a stochastic process. Fredkin and Toffoli [71] propose a measure of the amount of useful computation extractable from a dynamical system, but theirs is deterministic. D’Souza *et al.* [23] also treat deterministic and reversible systems, citing the determinism and reversibility of physical law. But they don’t explicitly treat the question of how errors (for instance, those introduced by thermal interaction with the environment) can be practically treated, other than to suggest using more degrees of freedom of the system to represent each logical quantity. The weakness with this approach is one of dynamical mixing: The internal degrees of freedom initially inside one bit of information will likely cross couple to other degrees of freedom over time, causing the initial state of the bit to be eventually lost. To directly treat the mixing problem, for instance by fundamentally quantifying the rate of logical entropy produced by the mixing of the dynamics, *vs.* the density of stored information in the computation, would be a valuable addition to the state of knowledge in this field.

Part III
Appendices

Appendix A

Sub-Additivity of Computation Capacity

Definition:

let f and g be binary combinatorial circuits with stochastic computation capacities A_f and A_g , respectively. Let f be described by $O_f = F(I_f)$, and let g be described by $O_g = G(I_g)$. Then we will say a *stateless combination* of f and g is a wiring together of f and g in which:

1. inputs may be wired together
2. outputs may *not* be wired together
3. inputs of f may be determined by (wired to) outputs of g
4. inputs of g may be determined by outputs of f , and
5. finally, the entire circuit remains combinatorial in nature; that is, the combined output is still a function of the reduced input set.

Lemma (sub-additivity for null-input systems):

Let f and g be binary combinatorial circuits with null-input stochastic computation capacities A_f and A_g , respectively. Let fg be a stateless combination of f and g , and let its null-input stochastic computation capacity be denoted A_{fg} . Then $A_{fg} \leq A_f + A_g$.

Proof:

Any wiring together of inputs may be expressed as a restriction on the input space. Or, taken another way, the inputs I_f and I_g to the two subsystems are *not* independent.

Also, any wiring of inputs of f to outputs of g or *vice versa* can again be expressed as a restriction on the allowable input space. If inputs of f are wired to outputs of g , for example, then we have I_f is dependent on O_g , which is in turn dependent (in fact, determined by) I_g . So I_f is dependent on I_g , or I_f and I_g are not independent.

The definition of statistical computation capacity gives us:

$$\begin{aligned}
 A_f = I(I_f, O_f) &= H(I_f) - H(I_f|O_f) \\
 A_g = I(I_g, O_g) &= H(I_g) - H(I_g|O_g) \\
 A_{fg} = I(I_{fg}, O_{fg}) &= H(I_{fg}) - H(I_{fg}|O_{fg}) \\
 &= H(O_{fg}) - H(O_{fg}|I_{fg})
 \end{aligned}$$

where the final equality follows from the symmetry of $I(\cdot, \cdot)$.

case i: independent If I_f and I_g are independent, then the first term is trivially: $H(O_{fg}) = H(O_f) + H(O_g)$. The second term can be written as follows: $H(O_{fg}|I_{fg}) = H(O_f|I_{fg}) + H(O_g|I_{fg})$. Since O_f is independent of I_g and *vice versa*, $H(O_f|I_{fg}) = H(O_f|I_f)$, and $H(O_g|I_{fg}) = H(O_g|I_g)$. So we have

$$\begin{aligned} A_{fg} &= H(O_f) + H(O_g) - H(O_f|I_f) - H(O_g|I_g) \\ &= A_f + A_g \end{aligned}$$

case ii: dependent In the dependent case, we have

$$H(O_{fg}) = H(O_f) + H(O_g) - I(O_f, O_g)$$

and

$$\begin{aligned} H(O_{fg}|I_{fg}) &= H(O_f|I_{fg}) + H(O_g|I_{fg}) - I(O_f, O_g|I_{fg}) \\ &= H(O_f|I_f) + H(O_g|I_g) - I(O_f, O_g|I_{fg}) \end{aligned}$$

(because O_f is uniquely determined by I_f and O_g is uniquely determined by I_g). So,

$$\begin{aligned} A_{fg} &= H(O_f) + H(O_g) - I(O_f, O_g) \\ &\quad - H(O_f|I_f) - H(O_g|I_g) + I(O_f, O_g|I_{fg}) \\ &= A_f + A_g - [I(O_f, O_g) - I(O_f, O_g|I_{fg})] \end{aligned}$$

But, since $I(O_f, O_g|I_{fg}) \leq I(O_f, O_g)$, the term in square brackets is at least zero, hence

$$A_{fg} \leq A_f + A_g$$

QED.

Lemma (sub-additivity for generalized systems): Let f and g be binary combinatorial circuits with generalized stochastic computation capacities A_f and A_g , respectively. Let fg be a stateless combination of f and g , and let its generalized stochastic computation capacity be denoted A_{fg} . Then $A_{fg} \leq A_f + A_g$.

Proof: First, let \mathcal{P}_f be a partition of I_f , and \mathcal{P}_g be a partition of I_g . Then we will denote the composition of these two partitions as \mathcal{P}_{fg} . We have the definition of generalized stochastic computational capacity for the combined system *with no restrictions*:

$$A_{fg} = H(O_{fg}) + \sup_{\mathcal{P}_{fg}} H(\overrightarrow{O_{fg\mathcal{P}_{fg}}}|O_{fg}).$$

If some inputs and/or outputs are tied together, this may be written as a restriction on the space of possible outputs O_{fg} (since O_f and O_g are not independent). Let us write this restriction in terms of an arbitrary function:

$$\Theta(O_f, O_g) = 0.$$

Now we can write the expression for A_{fg} with the restriction applied:

$$A_{fg} = H(O_{fg} | \Theta(O_f, O_g) = 0) + \sup_{\mathcal{P}_{fg}} H(\overrightarrow{O_{fg\mathcal{P}_{fg}}} | O_{fg}, \Theta(O_f, O_g) = 0).$$

Since the restriction $\Theta(O_f, O_g) = 0$ can only *reduce* entropies of distributions to which it is applied, we can remove the restriction and replace equality with inequality:

$$A_{fg} \leq H(O_{fg}) + \sup_{\mathcal{P}_{fg}} H(\overrightarrow{O_{fg\mathcal{P}_{fg}}} | O_{fg}),$$

where systems f and g are independent. But where they are independent, the entropy terms can be split:

$$\begin{aligned} A_{fg} &\leq H(O_f) + H(O_g) \\ &\quad + \sup_{\mathcal{P}_{fg}} [H(\overrightarrow{O_{f\mathcal{P}_{fg}}} | O_{fg}) + H(\overrightarrow{O_{g\mathcal{P}_{fg}}} | O_{fg})] \\ &= H(O_f) + H(O_g) \\ &\quad + \sup_{\mathcal{P}_{fg}} [H(\overrightarrow{O_{f\mathcal{P}_f}} | O_{fg}) + H(\overrightarrow{O_{g\mathcal{P}_g}} | O_{fg})] \\ &= H(O_f) + H(O_g) \\ &\quad + \sup_{\mathcal{P}_{fg}} [H(\overrightarrow{O_{f\mathcal{P}_f}} | O_f) + H(\overrightarrow{O_{g\mathcal{P}_g}} | O_g)] \\ &= H(O_f) + H(O_g) \\ &\quad + \sup_{\mathcal{P}_f} H(\overrightarrow{O_{f\mathcal{P}_f}} | O_f) + \sup_{\mathcal{P}_g} H(\overrightarrow{O_{g\mathcal{P}_g}} | O_g) \\ &= A_f + A_g \end{aligned}$$

where the first step follows from the independence of O_f on \mathcal{P}_g , the second from the independence of O_f on O_g , and the third again from the independence of O_f on \mathcal{P}_g .

Hence, $A_{fg} \leq A_f + A_g$

QED

Appendix B

Relation between d_{\min} and d_{opt}

lemma:

If the optimal spacing of N and \mathcal{S} is d_{opt} , then for *any* code C in \mathcal{S} of size N ,

$$\begin{aligned} d_{\text{exp}}(C) &\equiv E[\underline{d}(w)] \\ &\leq 2(d_{\text{opt}}(\mathcal{S}, N) + 1). \end{aligned}$$

proof:

We use a refinement argument:

Let $d(\cdot, \cdot)$ be the Hamming metric. Assume $d_{\text{opt}}(N, \mathcal{S}) = d$. That is, any code of N codewords in \mathcal{S} must have a minimum separation distance at least equal to d . Now consider an arbitrary code C of N codewords in \mathcal{S} . We want to show an upper bound on the expected value of the separation distance of a randomly chosen codeword in C .

We start by rewriting $E[\underline{d}]$:

$$\begin{aligned} E[\underline{d}] &\equiv \frac{1}{N} \sum_{w \in C} \underline{d}(w) \\ &= \frac{1}{N} \sum_{i=1}^{\infty} i n_i, \end{aligned}$$

where n_i denotes the number of codewords in C with $\underline{d} = i$.

Define m_j and q_j as follows:

$$\begin{aligned} m_j &\equiv \sum_{i=j(d+1)}^{(j+1)(d+1)-1} n_i \\ q_j &\equiv \sum_{i=j(d+1)}^{(j+1)(d+1)-1} i n_i. \end{aligned}$$

Then

$$\begin{aligned} q_j &= \sum_{i=j(d+1)}^{(j+1)(d+1)-1} i n_i \\ &< (j+1)(d+1) \sum_{i=j(d+1)}^{(j+1)(d+1)-1} n_i \\ &= (j+1)(d+1) m_j. \end{aligned}$$

So,

$$\frac{1}{N} \sum_{j=0}^{\infty} q_j < \frac{d+1}{N} \sum_{j=0}^{\infty} (j+1)m_j. \quad (\text{B.1})$$

But notice the following: $m_0 = n_d$, since no codeword has $\underline{d} < d$. So,

$$E[\underline{d}] = \frac{1}{N} \sum_{i=1}^{\infty} im_i = \frac{1}{N} \sum_{j=0}^{\infty} q_j. \quad (\text{B.2})$$

We combine equations B.1 and B.2 to obtain:

$$E[\underline{d}] < \frac{d+1}{N} \sum_{j=0}^{\infty} (j+1)m_j.$$

Now, for the moment assume d is odd. Each codeword in the set corresponding to m_j has \underline{d} between $j(d+1)$ and $(j+1)(d+1) - 1$, inclusive. So, we may circumscribe every codeword with $d+1 \leq \underline{d} \leq 2d+1$ by a ball of radius $\frac{d+1}{2}$; every codeword with $2d+2 \leq \underline{d} \leq 3d+2$ by a ball of radius $\frac{2d+2}{2}$; *etc.*, and these balls will touch each other only at their boundaries (by the triangle inequality).

Now, if there is one codeword with a ball of radius $\frac{2d+2}{2}$, we can fit two balls of radius $\frac{d+1}{2}$ within it, and they will touch only at their boundaries. Likewise, if there is a codeword with a ball of radius $\frac{3d+3}{2}$, we can fit three balls of radius $\frac{d+1}{2}$ within it, and they will touch only at their boundaries, and so on. Imagine there were enough space in balls of radius at least $\frac{2d+2}{2}$ and greater, to fit an extra ball of radius $\frac{d+1}{2}$ in for each codeword with $\underline{d} = d$. This condition is:

$$m_2 + 2m_3 + 3m_4 + \dots \geq m_0.$$

If this condition were to obtain, then we could replace the code C with a code of the same size, but with a minimum separation of $d+1$. But we know this to be impossible, by the definition of d . So the condition must not be true. In other words,

$$m_2 + 2m_3 + 3m_4 + 4m_5 + \dots < m_0.$$

So,

$$m_0 + 2m_1 + 3m_2 + 4m_3 + \dots < 2m_0 + 2m_1 + 2m_2 + \dots$$

But the left side is $\sum (j+1)m_j$, and the right side is just $2N$. So,

APPENDIX B. RELATION BETWEEN D_{MIN} AND D_{OPT}

$$\begin{aligned} E[d] &< \frac{d+1}{N} \sum_{j=0}^{\infty} (j+1)m_j \\ &< \frac{d+1}{N} \cdot 2N \\ &= 2(d+1). \end{aligned}$$

QED

Bibliography

- [1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Communications of the ACM*, 43(5):74–82, 2000.
- [2] J. Ackermann and F. U. Gast. Word design for biomolecular information processing. *Z. Naturforsch. A*, 58:157–161, 2003.
- [3] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021, 1994.
- [4] Leonard M. Adleman.
- [5] Hatim T. Allawi and John SantaLucia Jr. Thermodynamics and NMR of internal G-T mismatches in DNA. *Biochemistry*, 36:10581–10594, 1997.
- [6] Hatim T. Allawi and John SantaLucia Jr. Nearest neighbor thermodynamic parameters for internal G-A mismatches in DNA. *Biochemistry*, 37:2170–2179, 1998.
- [7] Hatim T. Allawi and John SantaLucia Jr. Thermodynamics of internal C-T mismatches in DNA. *Nucleic Acids Research*, 26(11):2694–2701, 1998.
- [8] F. Allen, G. Almasi, W. Andreoni, D. Beece, B. J. Berne, A. Bright, J. Brunheroto, C. Cascaval, J. Castanos, P. Coteus, P. Crumley, A. Curioni, M. Denneau, W. Donath, M. Eleftheriou, B. Fitch, B. Fleischer, C. J. Georgiou, R. Germain, M. Giampapa, D. Gresh, M. Gupta, R. Haring, H. Ho, P. Hochschild, S. Hummel, T. Jonas, D. Lieber, G. Martyna, K. Maturu, J. Moreira, D. Newns, M. Newton, R. Philhower, T. Picunko, J. Pitera, M. Pitman, R. Rand, A. Royyuru, V. Salapura, A. Sanomiya, R. Shah, Y. Sham, S. Singh, M. Snir, F. Suits, R. Swetz, W. C. Swope, N. Vishnumurthy, T. J. C. Ward, H. Warren, and R. Zhou. Blue gene: A vision

- for protein science using a petaflop supercomputer. *IBM Systems Journal*, 40(2):310–327, 2001.
- [9] IBM T.J. Watson Research Center anonymous.
- [10] A. Arkin and J. Ross. Computational functions in biochemical reaction networks. *Biophysical Journal*, 67:560–578, August 1994.
- [11] Adam Arkin, John Ross, and Harley H. McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage λ -infected *escherichia coli* cells. *Genetics*, 149:1633–1648, August 1998.
- [12] Richard J. Bagley and J. Doyne Farmer. Spontaneous emergence of a metabolism. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II, SFI Studies in the Sciences of Complexity*, volume 10, pages 93 – 140. Addison-Wesley, 1991.
- [13] Edwin Banks. *Information Processing and Transmission in Cellular Automata*. PhD thesis, Massachusetts Institute of Technology, 1971.
- [14] Charles H. Bennett. The thermodynamics of computation – a review. *Intl. J. Theor. Phys.*, 21(12):905 – 940, 1982.
- [15] William Bialek. stability and noise in biochemical switches. *LANL Condensed Matter Physics Archive*, 2000.
- [16] Dennis Bray. Protein molecules as computational elements in living cells. *Nature*, 376:307–312, July 1995.
- [17] Arwen Brennenman and Anne Condon. Strand design for biomolecular computation. *Theor. Computer Science*, 287:39–58, 2002.
- [18] Pierre-Gilles de Gennes. *Scaling concepts in polymer physics*. Cornell University Press, Ithaca, N.Y., 1979.
- [19] R. Deaton, J. Chen, H. Bi, and J. A. Rose. A software tool for generating non-crosshybridizing libraries of DNA oligonucleotides. In *DNA Computing*, volume 2568 of *Lect. Notes in Computer Sci.*, pages 252–261. Springer-Verlag, Berlin, 2003.
- [20] R. Deaton, M. Garzon, R. E. Murphy, J. A. Rose, D. R. Franceschetti, and S. E. Stevens. *Phys. Rev. Lett.*, 80:417–, 1998.

-
- [21] R. Deaton, J. W. Kim, and J. Chen. Design and test of noncrosshybridizing oligonucleotide building blocks for DNA computers and nanostructures. *Applied Physics Letters*, 82:1305–1307, 2003.
- [22] M. Denneau. The Yorktown simulation engine. In *19th Design Automation Conference*, pages 55–59. IEEE, 1982.
- [23] Raissa M. D’Souza. *Macroscopic order from reversible and stochastic lattice growth models*. PhD thesis, Massachusetts Institute of Technology, 1999.
- [24] Paul J. Flory. *Statistical mechanics of chain molecules*. Interscience Publishers, New York, 1969.
- [25] Michael P. Frank. *Reversibility for Efficient Computing*. PhD thesis, Massachusetts Institute of Technology, 1999.
- [26] Michael P. Frank, Carlin Vieri, M. Josephine Ammer, Nicole Love, Norman Margolus, and Jr. Thomas F. Knight. A scalable reversible computer in silicon. In *Proc. of the ISCA Workshop*, Barcelona, 1998.
- [27] Edward Fredkin and Tommaso Toffoli. Conservative logic. *Int. J. Theor. Phys.*, 21:219–253, 1982.
- [28] A. Frutos, Q. Liu, A. Thiel, A. Sanner, A. Condon, L. Smith, and R. Corn. *Nucleic Acids Research*, 25:4748–, 1997.
- [29] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A*, 104(9):1876–1889, 2000.
- [30] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22:403–434, 1976.
- [31] Daniel T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A*, 188:404–425, 1992.
- [32] Peter J. E. Goss and Jean Peccoud. Quantitative modeling of stochastic systems in molecular biology by using stochastic petri nets. *Proc. Natl. Acad. Sci. USA*, 1995:6750–6755, June 1998.
- [33] Harpaz, Gerstein, and Chothia. *Structure 2*, pages 641–649, 1994.

- [34] W. E. Hart and S. Istrail. Lattice and off-lattice side chain models of protein folding: Linear time structure prediction better than 86 *J. Comput. Biol.*, 4(3):241–259, 1997.
- [35] A. Hartemink and D. Gifford. In H. R. Rubin and D. H. Wood, editors, *DNA Based Computers III*, pages 25–. American Mathematical Society, Providence, RI, 1999.
- [36] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, 280(5370):1716–1721, 1998.
- [37] A. Hjelmfelt and J. Ross. Mass coupled chemical systems with computational properties. *J. Phys. Chem.*, 97:7988–7992, 1993.
- [38] A. Hjelmfelt, E. D. Weinberger, and J. Ross. Chemical implementation of neural networks and turing machines. *Proc. Natl. Acad. Sci.*, 88:10983–10987, December 1991.
- [39] A. Hjelmfelt, E. D. Weinberger, and J. Ross. Chemical implementation of finite state machines. *Proc. Natl. Acad. Sci.*, 89:883–387, 1992.
- [40] John SantaLucia Jr. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proc. Natl. Acad. Sci. USA*, 95:1460–1465, February 1998.
- [41] Thomas F. Knight Jr. and Gerald Jay Sussman. Cellular gate technology. In *Unconventional Models of Computation*, pages 257–272, Singapore, New York, 1998. Springer.
- [42] R. M. Karp, C. Kenyon, and O. Waarts. Error-resilient DNA computation. *Random Structures and Algorithms*, 15:450–466, 1999.
- [43] Thomas F. Knight. Microbial engineering. <http://www.ai.mit.edu/people/tk/ce/microbial-engineering.html>.
- [44] Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM J. Rsch. Devel.*, 5:183–191, 1961.
- [45] R. J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–545, 1995.

- [46] Eric Lyons, Larry Lok, and Drew Endy. Stochastirator: Using a stochastic mathematical framework to simulate chemical and biological reaction networks. <http://opnsrcrio.molsci.org/stochastirator/stoch-main.html>.
- [47] Marcelo O . Magnasco. Chemical kinetics is turing universal. *Phys. Rev. Lett.*, 78(6):1190–1193, 1997.
- [48] Norman Margolus. CAM-8: A computer architecture based on cellular automata. In A. Lawniczak and R. Kapral, editors, *Pattern Formation and Lattice Gas Automata*, pages 167–187. American Mathematical Society, 1996.
- [49] Norman Margolus. An FPGA architecture for DRAM-based systolic computations. In Arnold *et al.*, editor, *Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, pages 2–11. IEEE Computer Society, 1997.
- [50] Norman Margolus. An embedded DRAM architecture for large-scale spatial-lattice computations. In *Proc. 27th Annual Intl. Symposium on Computer Architecture*, pages 149–160. IEEE Computer Society, 2000.
- [51] Norman H. Margolus. *Physics and Computation*. PhD thesis, Massachusetts Institute of Technology, Laboratory for Computer Science, 1988. MIT/LCS Tech Report 415.
- [52] Harley H. McAdams and Adam Arkin. Stochastic mechanisms in gene expression. *Proc. Natl. Acad. Sci.*, 94:814–819, February 1997.
- [53] Harley H. McAdams and Adam Arkin. Simulation of prokaryotic genetic circuits. *Annu. Rev. Biophys. Biomol. Struc.*, 27:199–224, 1998.
- [54] Harley H. McAdams and Lucy Shapiro. Circuit simulation of genetic networks. *Science*, 269:650–656, August 1995.
- [55] J. Monod and F. Jacob. *Cellular Regulatory Mechanisms*, pages 389–401. Cold Spring Harbor, New York, 1961.
- [56] Frederick C. Neidhardt and Michael A. Savageau. Regulation beyond the operon. In Frederick C. Neidhardt, editor, *Escherichia Coli and Salmonella*, pages 1310–1324. ASM Press, Washington, D.C., 2 edition, 1992.
- [57] Frederick C. Neidhardt and H. Edwin Umbarger. Chemical composition of *escherichia coli*. In Frederick C. Neidhardt, editor, *Escherichia Coli and Salmonella*, pages 13–16. ASM Press, Washington, D.C., 2 edition, 1992.

- [58] Jean Peccoud and Bernard Ycart. Markovian modelling of gene product synthesis. *Theoretical Population Biology*, 48:222–234, 1995.
- [59] R. Penchovsky and J. Ackermann. DNA library design for molecular computation. *J. Comput. Biol.*, 10:215–229, 2003.
- [60] Oliver Pretzel. *Error-correcting codes and finite fields*. Oxford University Press, New York, 1992.
- [61] J. H. Reif. Parallel molecular computation. In *Seventh Annual ACM Symp. on Parallel Algorithms and Architectures*, pages 213–223, 1995.
- [62] J. A. Rose, R. J. Deaton, D. R. Franceschetti, M. Garzon, and S. E. Stevens Jr. Hybridization error for DNA mixtures of N species. <http://engronline.ee.memphis.edu/molec/Misc/ci.pdf>, 1999.
- [63] J. A. Rose, R. J. Deaton, M. Hagiya, and A. Suyama. Equilibrium analysis of the efficiency of an autonomous molecular computer. *Phys. Rev. E*, 65, 2002.
- [64] J. A. Rose, M. Hagiya, R. J. Deaton, and A. Suyama. A DNA-based in vitro genetic program. *J. Biol. Phys.*, 28:493–498, 2002.
- [65] S. Roweis and E. Winfree. On the reduction of errors in DNA computation. *J. Computational Biol.*, 6:65–75, 1999.
- [66] S. Roweis, E. Winfree, R. Burgoyne, N. V. Chelyapov, M. F. Goodman, P. W. K. Rothmund, and L. M. Adleman. A sticker-based model for DNA computation. *J. Computational Biol.*, 5:615–629, 1998.
- [67] W. D. Smith. In R. J. Lipton and E. B. Baum, editors, *DNA Based Computers*, pages 121–. American Mathematical Society, Providence, RI, 1996.
- [68] Warren D. Smith and Allan Schweitzer. DNA computers in vitro and vivo. Technical Report 983, NECI, Princeton, NJ, April 1995.
- [69] M. Sugita. *J. Theor. Biol.*, 4:179–192, 1963.
- [70] R. Thomas. Boolean formalization of genetic control circuits. *J. Theor. Biol.*, 42:563–585, 1973.

- [71] T. Toffoli. Four topics in lattice gases: Ergodicity, relativity, information flow, and rule compression for parallel lattice-gas machines. In R. Monaco, editor, *Discrete Kinetic Theory, Lattice Gas Dynamics and Foundations of Hydrodynamics*, pages 343–354. World Scientific, 1989.
- [72] Tommaso Toffoli. *Cellular Automata Mechanics*. PhD thesis, University of Michigan, 1977. Comp. Comm. Sci. Dept. Tech Report 208.
- [73] Tommaso Toffoli. CAM: A high-performance cellular automata machine. *Physica D*, 10:117–127, 1984.
- [74] Tommaso Toffoli. Action, or the fungibility of computation. In Anthony J. G. Hey, editor, *Feynman and Computation*, chapter 21, pages 349 – 392. Perseus Books, Reading, MA, 1999.
- [75] Tommaso Toffoli and Norman Margolus. The CAM-7 multiprocessor: A cellular automata machine. Technical Report LCS-TM-289, MIT Laboratory for Computer Science, 1985.
- [76] Tommaso Toffoli and Norman Margolus. *Cellular Automata Machines: A New Environment for Modeling*. MIT Press, 1987.
- [77] Carlin Vieri, M. Josephine Ammer, Michael Frank, Norman Margolus, and Jr. Thomas F. Knight. A fully reversible asymptotically zero energy micro-processor. In *Proc. of the ISCA Workshop*, Barcelona, 1998.
- [78] John von Neumann. *Theory of Self-Reproducing Automata*. Univ. of Illinois Press, 1966. completed and edited by Arthur Burks.
- [79] Ron Weiss, George Homsy, and Thomas F. Knight. Toward *in vivo* digital circuits. In *Proceedings of the DIMACS workshop on Evolution as Computation*, Princeton, NJ, January 1999.
- [80] Ron Weiss, George Homsy, and Radhika Nagpal. Programming biological cells. In *Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, Wild and Crazy Ideas Session*, San Jose, California, October 1998.
- [81] Erik Winfree. Simulations of computing by self-assembly. In *Prelim. Proc. Fourth Internat. Meeting on DNA Based Computers*, pages 213–239. University of Pennsylvania, 1998.

- [82] Saed G. Younis and Thomas F. Knight. Practical implementation of charge recovering asymptotically zero power CMOS, 1992. <http://www.ai.mit.edu/people/tk/lowpower/crl.ps>.
- [83] B. T. Zhang and S. Y. Shin. In *Proceedings of the Third Annual Genetic Programming Conference, University of Wisconsin at Madison, 1998*, pages 735–, San Francisco, 1998. Morgan Kaufman.