Biological Computing

Stuart A. Kurtz
Stephen R. Mahaney
James S. Royer
Janos Simon

ABSTRACT Adleman's [Adl94] successful solution of a seven-vertex instance of the NP-complete Hamiltonian Path problem by recombinant DNA technology initiated the field of biological computing.
We propose a very different model of molecular computing based on the biochemistry of RNA editing and RNA translation. In our model, individual molecules become fully capable general purpose computers.

1 Introduction

Our goal in this paper is to show how chemically plausible modifications of well-understood biological systems can be used to construct general purpose processors out of individual molecules. To this end, we summarize basic biochemistry in Section 3, in order to make the technological proposals of Sections 4 and 5 understandable.

Scientific understanding of the molecular basis for cell biology has grown enormously in recent years. The cell is now understood as a computational system: its program resides in DNA , and its state in the distribution of chemical compounds and electrical charges. The cell's state and environment interact with an enormous variety of expression control and transport systems to determine subsequent states.

Adleman [Adl94] was the first computer scientist to implement a computational technology based on biological ideas. Adleman gave the best of all possible plausibility arguments: he performed a computation using the technology he described. In this, his work remains unique.

The computational technologies we describe in this paper are very different from Adleman's, not only in the particular biochemical systems involved, but in the nature of the computations each system can perform. As a result, the distribution of intellectual risks involved in each proposal is quite different. Adleman's proposal is based on sound chemistry, but he does not seem to give a general purpose computation machinery. Attempts by Adleman's successors to quantify the power of his ideas in terms of traditional complexity classes (such as NP or PSPACE) do not seem realistic (e.g., see Hartmanis [Har95] for a discussion of mass vs. instance size for

NP computations based on Adleman's approach.). Our approach is more speculative with respect to its chemistry, but is intrinsically conservative from a computational point of view: we seek from the outset to produce general purpose computers.

We contrast Adleman's technology with ours in Section 6, but understanding his work is not a prerequisite to understanding the most significant ideas in this paper, and we do not attempt a systematic presentation of his work. His original paper [Adl94] is clearly written and accessible, although many of our readers will benefit from reading our brief introduction to biochemistry (Section 3) first.

2 The One-Molecule Processor

The essence of computation is iterated state transformation.[1]

The Representation of State

The first step in developing a computational technology is to determine how state is to be represented physically. The state of a computation has been represented in many ways, e.g., by pebbles, by triangular marks pressed into a clay tablet, by pencil marks on a sheet of paper, by the setting of a gear, or by an electrical potential across a capacitor.

Polymers are molecules that consist of repeated structural units called monomers. Some polymers (including several important families of biopolymers) are linear, and so are strings over an alphabet of monomers. For example, proteins are linear polymers based on twenty amino acid monomers; hence, proteins are strings on a twenty letter alphabet. Thus, an individual protein molecule can represent the state of a computation.

Transformating State

The second step in developing a computational technology is to show how to transform state, i.e., how the physical representation of one computational state can be used to produce a physical representation of a[2] successive state. In mechanical and electronic computers, the physical representation of one state is altered to become the physical representation of a successive state:

---

[1] Turing [Tur36] in his seminal paper on computability introduced the concept of the *configuration* of a computation. He formalized the notion of a computation as a sequence of configurations, each of which followed from the previous by the application of a *move* of the machine. Turing contended [Tur36, p. 232], in what has become known as "Turing's thesis," that "these operations include all those which are used in the computation of a number." Subsequent experience in computing has tended to confirm this thesis. The opening sentence of this section is simply a restatement of Turing's thesis.

[2] We use the indefinite pronoun here to account for the possibility of nondeterministic computations.

gears rotate to new positions, or electrical charges are transported so as to create different potentials. In traditional pencil and paper computations, though, one usually writes out each step of the computation separately, so that creating the physical representation of one state does not entail the loss of the physical representation of its predecessor.

To accomplish this second step for polymer based computers, we need to devise a sufficiently rich set of transformations on them. This leads naturally to the consideration of biochemical polymers and biological processes, because the cell has evolved an incredible variety of chemical mechanisms for manipulating biopolymers. We propose to take advantage of these and similar mechanisms to perform the state transformations upon which our computational technologies will be based.

Iterated Transformations

The final step is to develop a process for iterating those state transformations. This step can be more difficult than it first seems. Consider an electronic computer. It is conceptually straightforward to build a "transformation circuit" that, given a physical representation of state in a vector of input registers, uses standard boolean components to set a vector of output registers so that they represent the successive state. The hard part is in getting those values out of the output registers and back into the input registers so that the process can be repeated. In practice, this often involves latching circuitry that adds considerable complexity to the overall processor design.

Somewhat similar difficulties will attend our efforts to base computers on RNA translation.

By reducing the size of a processor to a single molecule, we make possible an enormous increase in potential parallelism. A mole ($6.02 \cdot 10^{23}$) of one-molecule processors would weigh a couple hundred kilograms; if we were content with a mere 10^{20} processors, we'd be down to about an ounce in weight. As a comparison, few modern supercomputers have more than a thousand processors. Even if our individual processors are much less powerful than those of modern computers by a dozen orders of magnitude, we might still be able to outperform existing supercomputers by factors of hundreds of millions.

3 A Brief Introduction to Biochemistry

This section provides an overview of the basic biochemistry that supports our proposed technologies. Our presentation necessarily presents general principles as if they were universal. In reality, general principals of biology often have interesting and informative counterexamples. *Molecular Cell Biology* [LBB+95] offers a much more comprehensive and systematic presentation of this material.

3.1 DNA*,* RNA*, and Proteins*

Single stranded DNA (ssDNA) is a linear polymer, whose monomeric constituents are called *nucleotides*. Each nucleotide consists of a pentose (2-deoxyribose, a five-carbon sugar), to which is attached a base (one of adenine, guanine, cytosine, and thymine), and a phosphate group. We find it helpful to think schematically, as in Figure 1. Consecutive monomers in an

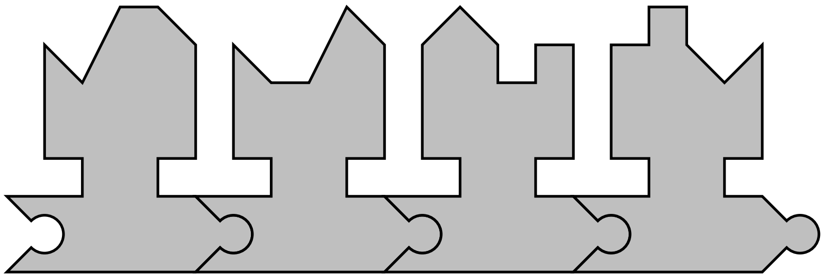

FIGURE 1. Schematic of Single Stranded DNA. Each individual "puzzle piece" represents a nucleotide. The covalent bonds between consecutive nucleotides are indicated by the "ball and socket" connections. The structures at the top of each nucleotide are the bases.

ssDNA molecule are linked by covalent bonds between the phosphate group of one monomer and the sugar group of its successor. The nucleotide bases hang off to the side. We think of single-strands of DNA as representing strings on the four letter alphabet $\{\mathsf{A}, \mathsf{G}, \mathsf{C}, \mathsf{T}\}$.

Within the cell, DNA is usually present in its double-stranded form (dsDNA). As its name suggests, double-stranded DNA consists of two single-strands, bound together by their bases. The nucleotide bases form *complementary pairs*: adenine bonds with thymine, and guanine bonds with cytosine. The Watson-Crick complementarity principle states that the two strands must be antiparallel complements of one another, cf. Figure 2. From a biological point of view, DNA has two critical properties: it can convey information, and it can be accurately replicated. Replication is accomplished by the following process: a molecule of dsDNA is separated into the two molecules of ssDNA, while suitable complements for each strand are assembled from nucleotide monomers. Replication of DNA is a crucial step in *mitosis* (cell division), and is the mechanism whereby each daughter cell receives a complete and accurate copy of the genetic information of its parent.

Ribonucleic Acid (RNA) is structurally similar to DNA, but differs in that the pentose is ribose rather than 2-deoxyribose, and that the nu-

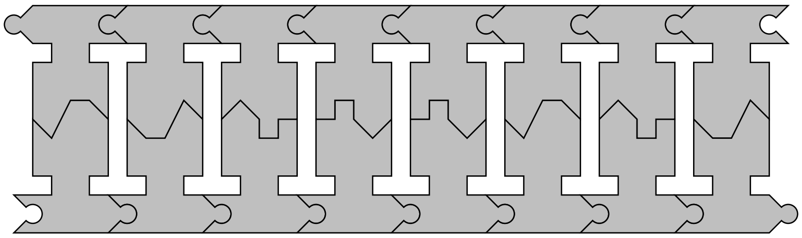

FIGURE 2. Schematic of Double Stranded DNA. In this picture, two complementary ssDNA molecules form a molecule of dsDNA. As the schematic suggests, the Watson-Crick bonding between strands is much weaker than the covalent bonding along the backbone of an individual strand.

cleotide base uracil (U) replaces thymine. In the cell, RNA is produced by *transcription* from DNA by a process similar to DNA replication. RNA plays several critical roles in the cell. Messenger RNA (mRNA) is a single-stranded linear molecule that conveys the description of a protein. We think of individual molecules of mRNA as representing strings on the four letter alphabet $\{\mathsf{A}, \mathsf{G}, \mathsf{C}, \mathsf{U}\}$. Transfer RNA (tRNA) and ribosomal RNA (rRNA) are single-strands of RNA that have been folded, taking on a secondary structure which enables them to function efficiently as enzymes. Both tRNA and rRNA play crucial roles in protein synthesis.

Proteins are polymers based on 20 amino acid monomers. Although proteins can and do associate themselves into multi-stranded complexes, the mechanism is very different from the Watson-Crick mechanism used by DNA and RNA. Unlike DNA and RNA, there is no biological process for direct replication of proteins, instead they must be synthesized from instructions encoded in mRNA.

3.2 Protein Synthesis

Protein synthesis begins within the cell nucleus, when a *gene* (a sequence of DNA that encodes a protein) is transcribed to a molecule of mRNA. The molecule of mRNA is then transported out of the nucleus to the cytosol[3]. During transportation, the mRNA molecule may undergo substantial alteration. The protein is assembled from the instructions encoded on the mRNA at the ribosome. The *central dogma of biology* (cf. Figure 3) [LBB$^+$95, p. 12] sums this whole machinery up concisely: DNA specifies RNA, which specifies protein.

We now consider this process in greater detail.

[3]The *cytosol* is the region of the cell outside of any organelle [LBB$^+$95]. An *organelle* is a structural unit of the cell, e.g., a ribosome or mitochondrion.

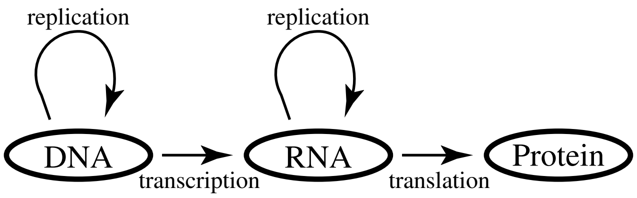

FIGURE 3. The Central Dogma of Biology

Gene Expression and Transcription

It is conventional—and for good reason—to view DNA as the cell's program. Still, meaning cannot be assigned to a program without fixing the computational system that will run it. For DNA, that computational system is the cell—the structure of which is specified by the DNA itself. The self-referential aspects of this machinery can scarcely surprise anyone familiar with computation.

To develop the computational metaphor of DNA, it is perhaps more instructive to view each individual gene as a program in a distributed multiprocessing system. This forces us to view the cell as a parallel computer with a distributed operating system. As a computer, the cell determines the result of running the program specified by a gene, which is typically the construction of a protein molecule. As a distributed operating system, the cell determines when and how often to execute a particular program.

A biologist would describe this machinery using somewhat different terminology. Genes may or may not be *expressed*. The sum of biological activities that determine cellular activity is *regulation*, and an important regulated activity is *transcription initiation*.

The cell has a wide variety of mechanisms for regulating transcription initiation. Typically, there are regulatory proteins (repressors and/or initiators) that undergo conformal changes when bound to certain chemicals. In this way, feedback mechanisms are established that link the concentration of reagents to the production of enzymes that catalyze the associated reactions. For example, expression of β-galactosidase in *E. Coli* is controlled by a repressor protein called the *lac* repressor. The *lac* repressor ordinarily binds to the gene that encodes β-galactosidase and prevents its transcription. Transcription can occur only when the *lac* repressor binds to a lactose molecule, which causes it to detach from the gene. The end result is that the rate of production of β-galactosidase, a protein used to digest lactose, is coupled to the concentration of lactose in the cell.

This feedback control mechanism can be modeled by a simple system of differential equations (with time shifts), and so looks like a special purpose analog computer. Our interest, though, is in digital computers, and we leave the further development of the computational capabilities of these systems to others.

Post-Transcriptional Processing of mRNA

A molecule of mRNA may undergo substantial alteration between transcription and translation.

The best understood alteration is the removal of noncoding sequences called *exons* from the molecule of mRNA. Curiously, some of these exons spontaneously form a secondary structure that catalyzes their own excision. It is not clear what benefit, if any, the presence of exons conveys to the organism that contains them. In some cases, the mechanism whereby exons are removed may function as a regulatory device. In others, the exon seems to be nothing more than a well-adapted genetic parasite.

From a computational point of view, it is difficult to take advantage of this particular process: state is not so much altered as it is thrown away.

A less well understood kind of post-transcriptional processing goes by the name of RNA *editing*. In RNA editing, nucleotide substitution takes place. In animal cells, only point alterations (i.e., the substitution of a single nucleotide for another nucleotide) are known, but in the mitochondrial mRNA of plants and protozoans, more than half of the original nucleotides may be replaced [LBB$^+$95, p. 535]. It is not completely clear what sort of substitutions occur, or whether or not they are all necessarily length preserving.

While this particular process may be poorly understood from a biological point of view, it could lead in a very straightforward way to the implementation of a one-molecule processor. Most likely, there are enzymes that match certain specific mRNA sequences, and then catalyze replacements within that sequence. If a sufficient variety of these enzymes exist, it is easy to see how the vision of a one-molecule processor can be realized. Significantly, such a processor would have a real advantage over our standard models of serial computation (such as Turing and von Neumann machines), in that multiple replacements could be occurring simultaneously along the state molecule: in effect, we have a one-molecule asynchronous array!

Translation

Proteins are assembled at the *ribosomes*. We think of a ribosome as a "jig" upon which a protein is assembled according to instructions provided by a molecule of mRNA. Significantly, the active portions of the ribosome are not proteins, but rather RNA.

The mRNA molecule is read as a sequence of three-letter *codons*. Thus, the language of protein synthesis—the genetic code—contains sixty-four words, each of which consists of a three letter sequence on the alphabet $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{U}\}$. There are more words in the language (64 base triplets) than there are objects to be denoted (20 amino acids). This discrepancy is resolved in two ways: the coding is degenerate, i.e., most amino acids are encoded in more than one way; and some of the codons are control instructions, specifically start and stop.

The association of amino acids to codons is *not* determined by the ribosome, but rather by molecules of transfer RNA. The nature and role of tRNA is central to our development, and so we will consider it in greater detail.

There are at least twenty different varieties of tRNA—one for each amino acid. An individual tRNA molecule is shaped like a lower-case letter "t" (albeit with a straight lower arm). The long, lower arm of the "t" is the *anticodon*, which contains a nucleotide base sequence complementary to the corresponding codon(s)[4]. The upper *acceptor arm* of the "t" binds to an amino acid. If there is an amino acid bound to the acceptor arm, the molecule is known as an aminoacyl-tRNA. The left and right arms of the "t" are known as the D and TψCG loops respectively. Molecular biologists do not currently assign to the D and TψCG loops a role in decoding the mRNA, but there is some biochemical evidence that they might, and it will be useful for our development to assign them a speculative role. Figure 4 gives a schematic view of an aminoacyl-tRNA.

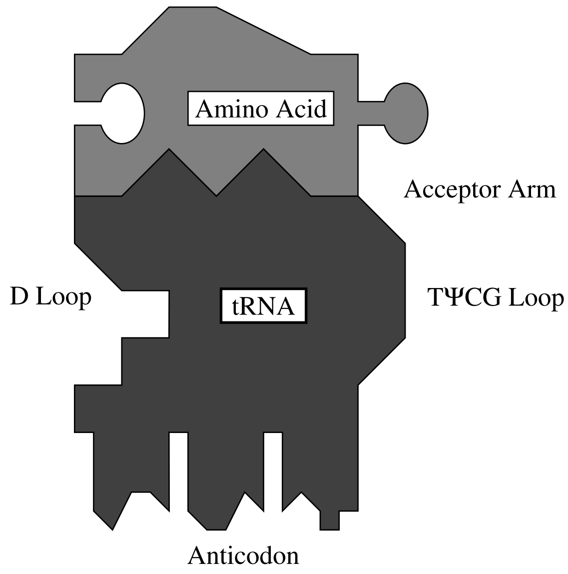

FIGURE 4. Schematic of an Aminoacyl-tRNA

At the ribosome, a molecule of protein is constructed with the sequence encoded by the molecule of mRNA. The ribosome has three active sites at which tRNA is temporarily bound: the aminoacyl-tRNA site (A), the

[4]Some tRNA molecules contain modified bases that can bind with more than one of the RNA bases, giving rise to a weaker complementarity. This particular nuance is the source of most of the degeneracy in the genetic code, but it is unimportant to us.

peptidyl-tRNA site (P), and the ejection site (E). At the beginning of a single step of protein construction, the ribosome is holding a tRNA molecule at its P site. The acceptor arm of the tRNA molecule is bound to the tail of the protein being constructed (the peptide), while the anticodon is held against a complementary site on the mRNA molecule. A step begins when an aminoacyl-tRNA complementary to the next codon of the mRNA molecule binds to the A site. The bond between the peptide and the peptidyl-tRNA molecule is then broken, and a new bond is established between the peptide and the amino acid bound to the acceptor arm of the tRNA molecule at the A site. The tRNA molecules then shift: the now unaminoacylated tRNA molecule at the P site moves to the E site, from where it is ejected; while the now peptidylated tRNA molecule at the A site moves to the P site, concluding the step. Figure 5 gives a schematic view of this process.

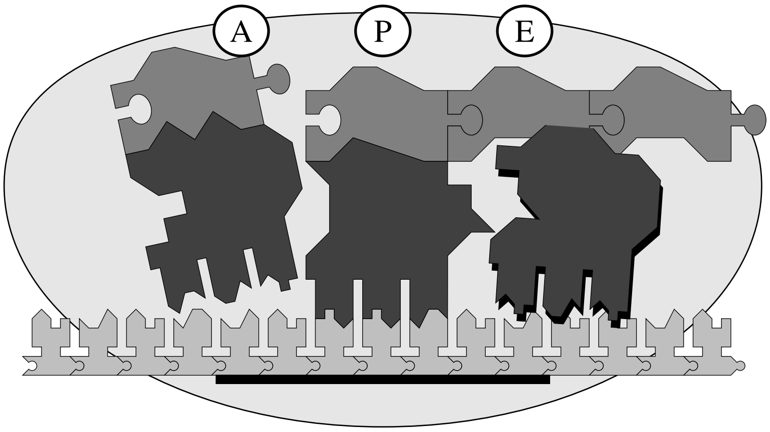

FIGURE 5. Schematic of Protein Synthesis at the Ribosome. The diagram shows an aminoacyl-tRNA approaching the A site of a ribosome, a peptidyl-tRNA at the P site, and an unaminoacylated tRNA that was recently ejected from the E site.

This system has considerable computational promise. The transcription of a molecule of mRNA into a protein molecule closely resembles a finite state transducer, i.e., a Post correspondence system. We can easily represent the moves of a Turing machine by finite state transductions, and so the possibility of constructing a general purpose computer based on transcription is very plausible.

Against this promise must be weighed two substantial objections. First, the protein molecule contains only one third as many monomers as the molecule of mRNA from which it was transcribed. Although such a system can lead to recognizable computations, they are of necessity quite brief! The second objection comes from the simple fact that our input and output alphabets do not coincide. This is even more substantial, as the output from one round of transcription cannot serve as input for the following round. If this restriction cannot be removed, only one-step computations are possible.

4 Computational Molecules

As we have just seen, the biochemical systems involved in protein synthesis have some very promising computational aspects, together with some possibly debilitating defects. In this section, we will describe a speculative chemical system that retains the computationally favorable aspects of protein synthesis while dispensing with its defects.

This approach is not without risk. Unlike Adleman, we cannot go into the lab tomorrow and perform a computation. We are not aware of any molecular system that has all of the properties we require, although the biochemistry of protein synthesis has most of them. Still, we can see no fundamental chemical or physical objection to the system we propose, and informal discussions of our ideas with our colleagues in biochemistry have thus far been most encouraging.

4.1 CNA

We propose a model of a computational molecule, which we'll call CNA, for "computational nucleic acid." CNA is our primary candidate for the one-molecule processor. We intend for CNA be a simultaneous generalization of DNA, RNA, and protein, although from a mathematical standpoint, it is just a string over a finite alphabet Σ. A CNA molecule can replicate, like DNA. A CNA molecule can also be interpreted, like mRNA, to synthesize a new molecule, in this case another CNA molecule rather than a protein. This synthesized CNA molecule can, in turn, be interpreted to synthesize a third CNA molecule, and so on.

Lest this seem fanciful, RNA is a potential candidate for implementing CNA. Although we usually think of DNA as double stranded and RNA as single stranded, double strands of RNA can form according to the Watson-Crick complementarity rules, so RNA can be replicated by essentially the same process as DNA. In modern cells, RNA is not interpreted by ribosomes to synthesize RNA, but many biologists believe that the first forms of life were purely RNA based. In this hypothetical "RNA world," enzymes were constructed from the RNA nucleotides rather than from amino acids by structures analogous to ribosomes. Recent research has illustrated that RNA molecules form unexpectedly efficient enzymes [ESB95], and biochemical researchers hope to be able to reconstruct an RNA-building ribosome analogue.

4.2 tCNA

Our mechanism for handling CNA translation introduces a generalization of tRNA called tCNA. The usual model for protein synthesis is altered, giving the D and TψCG loops a computationally significant role. We assume that there is a finite set of possible D loops, and a complementary set of TψCG loops. A tCNA module can participate in a step of protein synthesis only if both its anticodon is the complement of the codon of the CNA being interpreted, and its D loop is the complement of the TψCG loop of the previous tCNA.

In fact, it is known that there are codons in mRNA which can be transcribed either as an amino acid or as a control instruction. We hypothesize that interactions between the D and TψCG loops of consecutive cellular tRNA are responsible for controlling which of these possible interpretations occurs during protein synthesis. The transcription of a codon is not independent of context—it also depends on the codons that preceded it.

There are rare occasions in vivo in which a letter of the mRNA instruction stream is skipped. This suggests another way in which tRNA can be generalized. The three-letter genetic code is an evolutionary contingency, not a chemical necessity. We can readily imagine versions of tRNA that have anticodons of arbitrary length, and we permit this with tCNA. In the same way, we permit tCNA to contribute more than a single nucleotide to the CNA molecule being synthesized[5].

Definition 1 *Fix a finite set Q of states, and a finite alphabet Σ.*

*A $\langle Q, \Sigma\rangle$-*tCNA *molecule is a four-tuple $\langle \alpha, a, \beta, b\rangle$ where $a, b \in Q$ and $\alpha, \beta \in \Sigma^*$. We denote this $\langle Q, \Sigma\rangle$-*tCNA *molecule by β-*$\mathrm{tCNA}^{\alpha}_{a,b}$ *in analogy with the standard biochemical notation for varieties of* tRNA*. If Q and Σ are understood, we will speak of* tCNA *molecules, rather than $\langle Q, \Sigma\rangle$-*tCNA *molecules. The* anticodon *of β-*$\mathrm{tCNA}^{\alpha}_{a,b}$ *is β, its* contribution *is α, its* entry state *is a, and its* exit state *is b.*

*A finite set T of $\langle Q, \Sigma\rangle$-*tCNA *molecules is called a $\langle Q, \Sigma\rangle$-*genetic code*,* a genetic code on Q and Σ*, or simply* a genetic code *when Q and Σ are understood. The* natural *genetic code is the genetic code used by the dominant*

[5] It seems possible that this part of our model is too ambitious, and it may be impractical to use anticodons of arbitrary length, because long anticodon strings ultimately entail low rates of CNA synthesis. A somewhat different objection might be made to polynucleotide contributions. If a tRNA variant attempted to contribute a polypeptide (as opposed to a single amino acid), the end of the contributed polypeptide might extend beyond the active region of the ribosome, making it impossible for the polypeptide to be bonded to the protein being synthesized.

Until the limits on the maximum lengths of codons and contributions are established, it is simpler mathematically to assume that both are unconstrained.

forms of biological life on earth[6]*. Other genetic codes are* synthetic.

If G is a genetic code, then CNA *molecule β can be G-*synthesized *from a* CNA *molecule α if there is a finite sequence $\left\langle \beta_i\text{-tCNA}^{\alpha_i}_{a_i,b_i} \right\rangle_{i=0\ldots k}$ of elements of G such that*

1. $\alpha = \alpha_1\alpha_2\ldots\alpha_k$,
2. $\beta = \beta_1\beta_2\ldots\beta_k$, *and*
3. $a_{i+1} = b_i$ *for all* $i = 0\ldots k$.

If β is G-synthesized from α, we write $\alpha \vdash_G \beta$.

Notice that the letters of Σ and the states of Q are autocomplementary, that is, we pair a letter $\sigma \in \Sigma$ on the CNA with σ on the anticodon of the corresponding tCNA molecule, rather than with a hypothetical $\overline{\sigma}$. Although this is convenient mathematically, it does not make for particularly plausible chemistry. Therefore, this should be thought of as a notational convention in which a compound is denoted by its complement when it is in an "anti-" role. For example, since cytosine is the complement of guanine, this convention denotes cytosine on the anticodon of tRNA by G rather than C. A similar consideration applies to the exit states ($\mathsf{T}\psi\mathsf{CG}$ loop) of tCNA (tRNA).

Also notice that we have dispensed with the usual control mechanism: there are no dedicated start and stop codons, rather, the interpretation of a CNA molecule begins at its beginning, and ends when it ends.

Much of the power of our model will come from the use of "designer" genetic codes. That is, we imagine that the machine architect (or perhaps even programmer) for a biological computer built along these lines will be able to specify a particular collection of tCNA molecules to be used in transcription. Indeed, it might be useful to imagine sequences of transcriptions, each with its own characteristic assemblage of tCNA molecules.

4.3 The synthesis of tCNA

The number of possible tCNA varieties is infinite, even if the state space Q is fixed. If we impose an additional fixed bound of k on the length of the contribution and anticodon, the number of potential varieties of tCNA is still immense

$$\left(\frac{|\Sigma|^{k+1}-1}{|\Sigma|-1}\right)^2 \cdot |Q|^2 .$$

[6] There is some minor variation in the interpretation of mRNA among different species of cellular life. For the sake of definiteness, the dominant form of biological life should be understood to mean *E. Coli.* Of course, the natural genetic code does not really fit within our model of CNA, but prepending the phrase "synthetic" to the genetic codes we construct seems likely to reduce confusion.

Even with fairly modest parameters—$k = 3$, $|\Sigma| = 4$, and $|Q| = 4$—there would be $115,600$ possible different varieties of tCNA.

To help make our model plausible, we show how relatively simple chemical system can be used to synthesize each of these varieties in a systematic way.

We will suppose that there is a simple *tCNA-core* molecule with four distinct binding sites, corresponding to the contribution, anticodon, entry state, and exit state. A *tCNA-arm* molecule is a double-ended molecule: one end consists of a complement to a binding site of the tCNA-core, the other consists of a contribution acceptor appropriate. If we limit the size of the contribution and anticodon to k, we can construct any tCNA molecule from a library of

$$2 \cdot \left(\frac{|\Sigma|^{k+1} - 1}{|\Sigma| - 1} + |Q| \right)$$

tCNA-arms.

Continuing our previous example, with $k = 3$, $|\Sigma| = 4$, and $|Q| = 4$, we'd need 1 tCNA-core, 85 acceptor arms ($85 = 64 + 16 + 4 + 1$), 85 anticodon arms, 4 entry arms, and 4 exit arms. From this modest library of 175 compounds, we could synthesize any of the $115,600$ tCNA molecules.

5 The Microarchitecture of CNA Computers

At this point, we know of three very different approaches to using biomolecular systems to perform computations.

1. Adleman's approach, based on DNA, affinity binding, and ligation;

2. The approach based on RNA editing we briefly sketched in Section 3.2, under *Processing*; and

3. The approach based on CNA, inspired by RNA transcription, that we described in Section 3.2, under *Translation*, and in Section 4.

Abstracting away from the particular details of these models reveals some common themes. Computations take the form of a series of chemical reactions taking place within a single "soup." Molecules are constructed that correspond to paths through a nondeterministic computation tree. At equilibrium, the soup contains a collection of molecules corresponding to the leaves of the computation tree; and the enormous number of molecules involved ensures that with high probability, every leaf of the tree has one or more of molecules corresponding to it.

Because nonaccepting leaves may greatly outnumber accepting leaves, this means that molecules that correspond to accepting computations may necessarily be present at extremely low (even homeopathic) concentrations. The problem of purifying and concentrating those molecules that correspond to accepting computations is substantial. In Adleman's paper [Adl94], the "soup" generated by the ligation is subject to a time-consuming sequence of affinity-based filtrations and polymerase chain reactions in order to determine whether or not any accepting paths were present.

As we have sketched the three technologies, it may seem that none of them offers any particular advantage with respect to the purification and concentration of desirable output molecules. This is not the case, however.

The computing model based on CNA and inspired by RNA transcription has the property that computation occurs by copying with modification, rather than by mutating existing molecules. It is easy to envision a ribosome being embedded in a membran in such a way that the molecule of CNA being synthesized passes a hole in the ribosome and so ends up separated by the membrane from the molecule of CNA that encoded it. Indeed, this is a fair description of the structure and function of the rough endoplasmic reticulum, an organelle involved in the production of secretory proteins.

Expanding on this vision, we imagine biochemical reaction "cells" consisting of parallel layers of membranes with embedded RNA-ribosomes. In such a system, polymer inputs would enter one side of the cell, and polymer outputs—representing possibly thousands of iterated transcriptions—would be secreted at the other end. Also note that there is no obvious reason why we must assume that only one genetic code is used. Each layer could perform its own idiosyncratic translation. Finally, the secreted outputs would *a fortiori* be pure and highly concentrated, facilitating their detection and any further analysis.

These reaction cells form a part of the microarchitecture of hypothetical biological computer, and provide valuable lessons at several levels.

Looking at actual biological organisms provides some useful insight into statements of the form "Biological computers can't do X." We, as biological organisms, routinely convert data about the macroscopic world gathered by our senses into a form that influences our biology at the molecular level. If we blow past a police car at 20 MPH over the speed limit, the production of adreneline is immediate and obvious. At the same time, we are able to convert the results of molecular reactions into macroscopic actions. You think "q," and your left pinky presses the appropriate key.

Moving beyond the feasibility argument, it seems like a good idea to look to real biological systems for solutions to particular problems. In this particular case, the rough endoplasmic reticulum seems to exist precisely to solve the sort of problem we envisioned.

At an altogether different level, a computational microarchitecture based on membrane-limited volumes justifies for the first time the name "biological," as opposed to merely "molecular" computing. In our reaction cells, the significance of a computational molecule does not depend merely on whether it is present or absent, as is the case with "soup" based models. The location of a molecule may play a critical role in determining its computational significance.

6 A Brief Discussion of Adleman's Model vs. Our Model

We made a conscious decision for this paper *not* to survey Adleman's paper [Adl94], or the many interesting papers that build upon the technology he invented. For this section, we will be assuming a general familiarity with those ideas. Fortunately, Adleman's paper is concise and well written. It should be easily accessible to any readers who have penetrated our discussion of molecular biology. Readers interested in subsequent developments within Adleman's framework should consult Reif's paper [Rei95], and the papers he cites.

The computational models we have developed are strikingly different from Adleman's use of biotechnology to solve the Hamiltonian Path problem.

The Hamiltonian Path problem is NP-complete. Like all NP-complete problems it has an exponentially large space of potential solutions and a polynomial time test to decide whether or not a given potential solution is an actual solution. Adleman's idea was to encode potential solutions by molecules of DNA. By performing a quadratically many chemical reactions, he is able to synthesize a volume of DNA which, with overwhelmingly high probability, contains for every potential solution at least one molecule that encoded it. By performing linearly many affinity-based filtration steps, he is able to winnow this solution down to just those *witness* molecules that encoded actual solutions. Finally, by using the polymerase chain reaction, he is able to increase the concentration of these witness molecules to detectable levels.

The algorithm he implemented was completely brute-force: generate and test all possible solutions. Indeed, it is difficult to reconcile Adleman's technique with the notion of *computation*. To us, it looks more like *evolution*: a huge *variety* of potential solutions are constructed by a chemical process that involves considerable randomness, and then from this huge variety, molecules are *selected* according to some *fitness* criterion.

Of course, the well known reductions from arbitrary NP-problems to particular NP-complete problems hint that greater generality lurks beneath the particulars of Hamiltonian Path, albeit with polynomial blowup. Later authors, notably Rooß-Wagner [RW95] and Beaver [Bea95] have taken up this hint, and have shown how general computations (limited by time and material) can be carried out within Adleman's model.

Unfortunately, it is not completely clear whether or not the particular techniques used by Rooß-Wagner and Beaver will work in the real world. A common problem is that both rely on affinity-based reactions where the "sticky ends" are polynomially long. It is not clear how reliable or fast these long affinity reactions can be. This is not a minor objection: Adleman's computation required a week even though only relatively short affinity reactions were involved; and there are good chemical reasons to expect that the reaction rate will decrease exponentially in the length of the affinity sequence.

In our CNA model, we can estimate the instruction rate based on the rate of protein synthesis, which is to say between 10 and 100 nucleotides per second. While this is not quick compared to modern desktop computers, it is several orders of magnitude faster than the reactions Adleman used. This reaction rate does not decrease as the length of the molecule being translated increases, and so our model seems much more robust under changes of scale.

7 Conclusions

Why hasn't research in artificial intelligence succeeded in producing a computer program that can pass the Turing test? Is it because our colleagues in AI aren't as sharp as they think they are (a favorite hypothesis of complexity theorists)? Or is it, perhaps, that the current computing technologies do not offer adequate performance?

A fairly simple plausibility argument suggests that computers have overtaken our brains in raw processing power. There are about 10^{12} neurons in the brain, and each can fire 250 times per second[7]. On this basis, the brain is capable of $2.5 \cdot 10^{14}$ one-bit instructions per second. A current-generation desktop computer is capable of about 10^8 32-bit instructions per second, which is to say about $4 \cdot 10^{17}$ one-bit instructions per second.

If we accept this computation, the human brain becomes extremely useful for making plausibility arguments about the capabilities of our present machines: if our brains can do it, they can do it. Thus, we might naturally hope that they will some day soon perform a few burdensome tasks that seem to require intelligence, e.g., arguing about budgets, organizing conferences, writing proposals, refereeing papers, and grading student programs. The computation suggests that only development of appropriate software for existing machines stands between us and our emancipation—or unemployment.

This same computation generally supports the theory that the perceived failures of artificial intelligence are due to the unworthiness of its practi-

[7] There is a fair amount of variation in published estimates of these numbers. The particular values we chose come from Lodish, el. al., [LBB+95].

tioners, rather than to the difficulty of the problem they face. But what if the computation is wrong? What if there is more to human intelligence than the electrical activity of our neurons?

Digital information can be stored, modified, and interpreted at the molecular level by biological systems. We looked at the biochemistry of the cell, and saw computations in its activities. Although we would like to take credit for these ideas as being our own invention—we didn't get there first. The cell did. The systems that look like computers, *are* computers; and they don't need our imprimatur to make them so. By viewing a neuron as a simple switch, we have radically underestimated its computational power, and have correspondingly underestimated the computation power of the brain.

By better understanding the true computational capabilities of biological systems, we open the path to computers that are much more powerful than those of the present generation. Implementing such machines may make the many goals of artificial intelligence acheivable for the first time.

8 Acknowledgements

The third author (Royer) would like to acknowledge the support of the National Science Foundation, through grant CCR-952287.

All four authors would like to gratefully acknowledge the heroic efforts of Mitsu Ogihara in producing a careful and thorough review of a late draft of this paper under extreme time constraints. We would also like to acknowledge the effort and consideration of the editors.

9 References

[Adl94] L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

[Bea95] D. Beaver. A universal molecular computer. Technical report, Pennsylvania State University, 1995.

[ESB95] E. H. Ekland, J. W. Szostak, and D. P. Bartel. Structurally complex and highly active RNA ligases derived from random RNA sequences. *Science*, 269:364–370, July 21 1995.

[Har95] J. Hartmanis. On the weight of computations. *Bulletin of the European Association for Theoretical Computer Science*, 55:136–138, 1995.

[LBB+95] H. Lodish, D. Baltimore, A. Berk, S. L. Zipursky, P. Matsudaira, and J. Darnell. *Molecular Cell Biology*. Scientific American Books, third edition, 1995.

[Rei95] J. Reif. Parallel molecular computation. In *Proceedings of the 7th ACM Symposium on Parallel Algorithms and Architectures*, pages 213–223, 1995.

[RW95] D. Rooß and K. Wagner. On the power of bio-computers. Technical report, Universität Würzburg, 1995.

[Tur36] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematics Society*, 42:230–265, 1936.

Index