23

Computing with DNA

Lila Kari and Laura F. Landweber

1. A New Player in the History of Computation

A brief look at the history of humanity shows that since the earliest days people needed to count and compute, either for measuring the months and the seasons or for commerce and construction. The means used for performing calculations were whatever was available, and thus progressed gradually from manual (digits) to mechanical (abacus, mechanical adding engine), and from there on to electronic devices. Electronic computers are only the latest in a long chain of human efforts to use the best technology available for performing computations. Although it is true that their appearance, some 50 years ago, has revolutionized computing, electronic computers mark neither the beginning nor the end of the history of computation. Indeed, even electronic computers have their limitations: There is a limit to the amount of data they can store, and physical laws dictate the speed thresholds they will soon reach. The most recent attempt to break down these barriers is to replace, once more, the tools for performing computations with biological ones instead of electrical ones.

DNA computing (also sometimes referred to as biomolecular computing or molecular computing) is a new computational paradigm that employs (bio)molecule manipulation to solve computational problems, at the same time exploring natural processes as computational models. Research in this area began with an experiment by Leonard Adleman, who surprised the scientific community in 1994 ***(1)*** by using the tools of molecular biology to solve a difficult computational problem. Adleman's experiment solved an instance of the Directed Hamiltonian Path Problem solely by manipulating DNA strands. This marked the first solution of a mathematical problem by use of biology.

From: *Methods in Molecular Biology, vol. 132: Bioinformatics Methods and Protocols*
Edited by: S. Misener and S. A. Krawetz © Humana Press Inc., Totowa, NJ

Computing with biomolecules (mainly DNA) generated a tremendous amount of excitement by offering a brand new paradigm for performing and viewing computations. The main idea was the encoding of data in DNA strands and the use of tools from molecular biology to execute computational operations ***(1a)***. Besides the novelty of this approach, molecular computing has the potential to outperform electronic computers. For example, DNA computations may use a billion times less energy than an electronic computer, while storing data in a trillion times less space ***(2)***. Moreover, computing with DNA is highly parallel: In principle there could be billions upon trillions of DNA molecules undergoing chemical reactions, that is, performing computations, simultaneously ***(3)***.

Despite the complexity of this technology, the idea behind DNA computing follows from a simple analogy between the following two processes, one biological and one mathematical:

a. The complex structure of a living organism ultimately derives from applying a set of simple operations (copying, splicing, inserting, deleting, and so on) to initial information encoded in a DNA sequence.
b. Any computation, no matter how complex, is the result of combining very simple basic arithmetical and logical operations.

Adleman realized that the two processes are not only similar but that advances in molecular biology allow one to use biology to do mathematics. More precisely, DNA strings can encode information while various molecular biology laboratory techniques perform simple operations. (The reader is referred to **ref. *4*** for further molecular biology notions.) These practical possibilities of encoding information in a DNA sequence and performing simple DNA strand manipulations led Adleman ***(1)*** to solve a seven node instance of the Directed Hamiltonian Path Problem.

A directed graph G with designated nodes v_{in} and v_{out} is said to have a Hamiltonian path if and only if there exists a sequence of compatible one-way edges $e_1, e_2, ..., e_z$ (that is, a path) that begins at v_{in}, ends at v_{out} and enters every other node exactly once. A simplified version of this problem, known as the traveling salesman problem, poses the following question: given an arbitrary collection of cities through which a salesman must travel, such as the graph in **Fig. 1**, what is the shortest route linking those cities? Adleman's version limited the number of connecting routes between the cities by specifying the origin and final destination cities of his journey. Because not all cities are connected, the challenge was to discover a continous path to link them all, if one exists.

The following (nondeterministic) algorithm solves the problem:

1. Generate random paths through the graph.
2. Keep only those paths that begin with v_{in} and end with v_{out}.

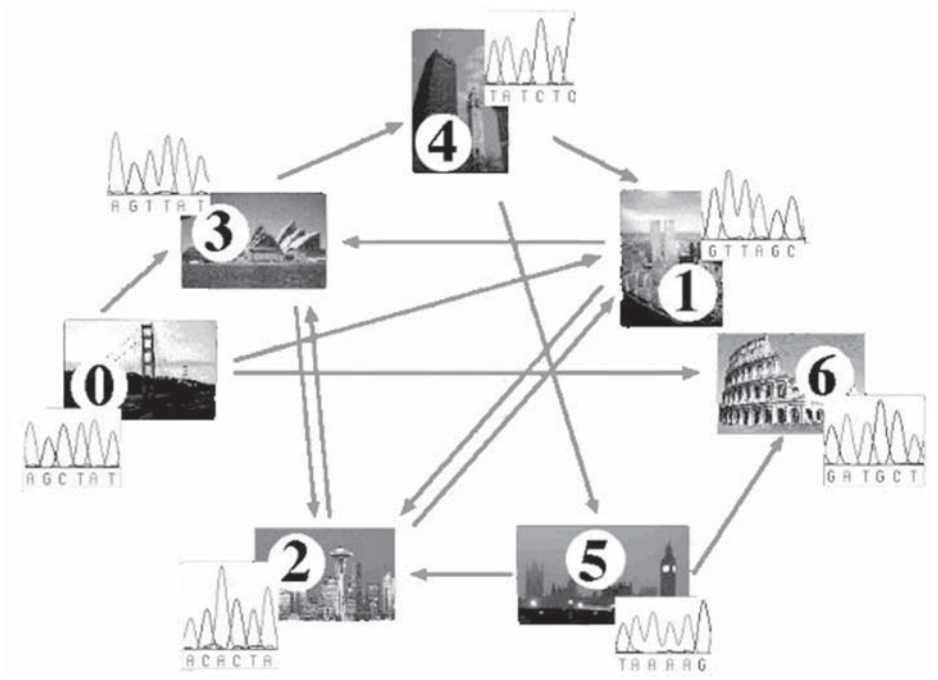

Fig. 1. An example of the graph used in Adleman's experiment ***(1)***. Cities, or nodes, are represented as arbitrary DNA sequences. The traveling salesman must find the simplest path which takes him through all seven cities shown, in this case departing from San Francisco (city 0) and arriving in Rome (city 6) as the final destination.

3. If the graph has n nodes, then keep only those paths that enter exactly n nodes.
4. Keep only those paths that enter all of the nodes of the graph at least once.
5. If any paths remain, say "yes"; otherwise say "no".

To implement **step 1**, each node of the graph was encoded as a random 20-base strand of DNA, or oligonucleotide. Then, for each (oriented) edge of the graph, a different 20-base oligonucleotide was generated that contains sequences complementary to the second half of the source node plus the first half of the target node. By using these complementary DNA oligonucleotides as splints, all DNA sequences corresponding to compatible edges would self-assemble and be ligated, or linked together, by the enzyme T4 DNA ligase. Hence, annealing and ligation reactions generated DNA molecules encoding random paths through the graph (**Fig. 2**).

To implement **step 2**, the product of **step 1** was amplified by polymerase chain reaction (PCR) using oligonucleotide primers representing v_{in} and v_{out}.

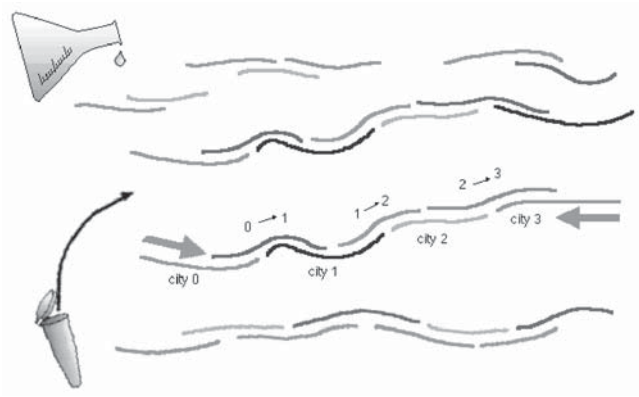

Fig. 2. Self assembly of DNA molecules representing paths through a graph. PCR primers marking origin and final destination oligonucleotides (cities 0 and 3 here) are shown as arrows. Complementary overlap exists between the second half of the sequence representing city i and the first half of a sequence representing edge $i \rightarrow j$, and also between the second half of the sequence representing $i \rightarrow j$ and the first half of the sequence representing city j.

This amplified and thus retained only those molecules encoding paths that begin with v_{in} and end with v_{out}.

For implementing **step 3**, agarose gel electrophoresis allowed separation and recovery of DNA strands of the correct length. The desired path, if it exists, would pass through all seven nodes, each of which was assigned a length of 20 bases. Thus PCR products encoding the desired path would have to be $7 \times 20 = 140$ bp.

Step 4 was accomplished by successive use of affinity purification for each node other than the start and end nodes. This process permits the separation and recovery of single strands encoding a given node from a heterogeneous pool of strands. DNA strands complementary to the node sequence were attached to magnetic beads. The heterogeneous solution containing single-stranded DNA was then passed over the beads and those strands containing the node sequence were selectively retained. Strands that lack one of the required node sequences generally do not survive **step 4**, because they pass through at least one of the columns without being retained.

To implement **step 5**, the presence of a molecule encoding a Hamiltonian path was checked by PCR. The first PCR amplified the results of **step 4** and checked for the presence of a product, as in **step 2**. If a product was present, then a second PCR confirmed the presence of each internal node by using the DNA oligonucleotides complementary to each node as PCR primers. This step

also elegantly allowed mapping and readout of connected nodes in the graph, without need for DNA sequencing.

A remarkable observation about Adleman's experimental result is that it not only solved a mathematical problem, but that it was also a difficult computational problem in the sense explained below (see **refs. *5*** and ***6***).

Problems can be ranked in difficulty according to the length of time the best algorithm will require to solve the problem on a single computer. Algorithms whose time complexity function is bounded by a polynomial function, in terms of the size of the input describing the problem, are in the polynomial time class *P*. Such algorithms are generally considered efficient. Any algorithm whose time complexity function cannot be so bounded belongs to the inefficient exponential class EXP. A problem is called *intractable* if it is so hard that no polynomial time algorithm can possibly solve it.

A special class of problems, apparently intractable, including *P* and included in EXP is the "nondeterministic polynomial time" class, or *NP*. The following chain of inclusions between problem classes holds:

$$P \subseteq NP \subseteq \text{EXP} \subseteq \text{Universal}$$

NP contains the problems for which no polynomial time algorithm to solve them is known, but that can be solved in polynomial time on a nondeterministic computer (a computer that has the ability to pursue an unbounded number of independent computational searches in parallel). The Directed Hamiltonian Path problem is a special kind of problem in *NP* known as "*NP*-complete." An *NP*-complete problem has the property that every other problem in *NP* can be reduced to it in polynomial time. Thus, in a sense, *NP*-complete problems are the "hardest" problems in *NP*.

The question of whether or not the *NP*-complete problems are intractable, mathematically formulated as "Does *P* equal *NP*?", is now considered one of the foremost open problems of contemporary mathematics and computer science. Because the Directed Hamiltonian Path problem has been shown to be *NP*-complete, it seems likely that no efficient (that is, polynomial time) algorithm exists for solving it with an electronic computer.

Other experiments have followed Adleman's to tackle mathematical problems using DNA manipulation. Kaplan et al. ***(7)*** repeated Adleman's experiment; Guarnieri, Fliss and Bancroft used a horizontal chain reaction for DNA-based addition ***(8)***; a Wisconsin team of computer scientists and biochemists made partial progress towards solving a five-variable instance of the Satisfiability (SAT) problem using surface chemistry ***(9)***; Quyang et al. ***(10)*** solved a six-variable *NP*-complete problem (the Maximal Clique Problem) using restriction enzymes; and most recently one of our laboratories ***(11)*** has solved a nine-variable SAT problem using RNA.

At the same time, numerous experiments have investigated a variety of aspects of the feasibility of DNA computing: They have addressed the effect of good encodings on solutions to Adleman's problem ***(12)***; studied the complications raised by PCR ***(13)***; investigated the use of self-assembly of DNA ***(14)***; pointed out the experimental gap between design and assembly of unusual DNA structures ***(15)***; reported joining and rotating data with molecules ***(16)***; studied concatenation with PCR ***(16,17)***; made progress towards evaluating simple Boolean formulas ***(18)***; conducted ligation experiments in computing with DNA ***(19)***; implemented an expert "Inference Engine" based on molecular computing ***(20)***; and obtained a partial solution to the Shortest Common Superstring Problem ***(21)***.

Theoretical studies have supplemented experimental research of DNA algorithms by suggesting potential strategies for solving various problems by means of DNA manipulation. Descriptions of such proposed experiments include the SAT Problem ***(22)***, breaking the Data Encryption Standard ***(23,24)***, expansions of symbolic determinants ***(25)***, matrix multiplication ***(26)***, graph connectivity and the knapsack problem using dynamic programming ***(27)***, the road coloring problem ***(28)***, exascale computer algebra problems ***(29)***, the Bounded Post Correspondence Problem ***(30)***, and simple Horn clause computation ***(31)***.

2. Towards a DNA Computer

The experiments mentioned so far are singular experiments that construct algorithms to solve particular problems. This immediately leads to two fundamental problems, posed already in **refs. *1*** and ***6***: What classes of problems can be efficiently solved by DNA algorithms? and Is it possible, at least in principle, to design a programmable DNA computer? Even though the models of DNA computation that have been proposed to answer these questions all differ from each other, they have a number of common features.

Indeed, any kind of computer, whether mechanical, electronic, or biological, needs two basic capacities to function: storage of information and the ability to perform operations on stored data. In the following we address both issues: how can information be stored in DNA strands, and what molecular biology techniques are potentially useful for computation. To distinguish between ordinary mathematical operations and biomolecular procedures performed on DNA strands, we use the term bio-operations to refer to the latter.

A single strand of DNA can be described as a string composed of a combination of four different symbols, A, G, C, T. Mathematically, this means we have at our disposal a four-letter alphabet $\Sigma = \{A, G, C, T\}$ to encode information. Incidentally, this is more than enough, considering that an electronic computer needs only two digits, 0 and 1, for the same purpose.

Concerning the operations performed on DNA strands, the proposed models of DNA computation generally use various combinations of the following "primitive" bio-operations:

— *Synthesizing* a desired polynomial-length strand, used in all models.
— *Mixing*: Combine the contents of two test tubes to achieve union ***(1,32–36)***.
— *Melting*: Dissociate a double-stranded DNA into its single-stranded complementary components by heating the solution (***35–39)***.
— *Annealing*: Bond together two single-stranded complementary DNA sequences upon cooling the solution (***35–39)***.
— *Amplifying* (copying): Make copies of DNA strands by using the polymerase chain reaction (PCR) (***1,25,32–38,40)***.
— *Separating* the strands by length using gel electrophoresis or other size fractionating methods (***1,32,33,36,37,40)***.
— *Extracting*: Capture strands that contain a given pattern as a substring by affinity purification (***1,32,34,40)***.
— *Cutting* DNA double strands at specific sites by using commercially available restriction enzymes. (***37,38,40–42)***.
— *Ligating*: Join DNA strands with compatible sticky ends by using DNA ligases (***37–42)***.
— *Substituting*: Substitute, insert, or delete DNA sequences by using PCR site-specific oligonucleotide mutagenesis (*see* **refs.** ***40,43)***.
— *Marking* single strands by hybridization: Complementary sequences are attached to the strands, making them double stranded. The reverse operation is unmarking of the double-strands by denaturing ***(9,33,35)***.
— *Destroying* the marked strands by using a variety of nucleases, (***9,11)***. or by cutting marked strands with a restriction enzyme and purifying intact strands by gel electrophoresis ***(10,33)***.
— *Detecting and reading*: Given the contents of a tube, say "yes" if it contains at least one DNA strand that meets the requirements of the applied operations, and then interpret the sequence; say "no" otherwise, ***(1,32–34,36)***.

A biocomputation consists of a sequence of bio-operations performed on tubes containing DNA strands. The bio-operations listed above, and possibly others, may then be used to write "programs." A program receives a tube containing DNA strands encoding information as input, and returns as output either statements "yes" or "no" or a new collection of tubes.

Various models of DNA computing, based on combinations of the above bio-operations, have been proposed and studied from the point of view of their computational power plus feasibility (*see*, for example, ***1,32,37,39,43–51)***. There are advantages and disadvantages for each of the proposed models but, overall, the existence of different models with complementary features shows the versatility of DNA computing and increases the likelihood of practical construction of a DNA computing device.

Many substantial engineering challenges to constructing a DNA computer remain at almost every stage. These arise primarily from difficulties in dealing with large-scale systems and in coping with ensuing errors ***(52)***. However, we remark that the issues such as active monitoring and adjusting the concentrations of biological molecules, as well as fault tolerance, are all addressed in biological systems by nature: Cells must adjust the concentrations of various compounds, to promote reactions of rare molecules, and they also cope with undesirable byproducts of their own activity. Because cells can successfully manage these problems *in vivo*, this may ultimately suggest strategies we can mimic *in vitro*. Taking a theoretical step in this direction, ***(53)*** suggests the use of membranes to separate volumes (vesicles) and active transport systems to shuttle selected chemicals across these borders ***(53)***. Moreover, familiar computer design principles for electronic computers could be exploited to build biomolecular computers ***(3,54,55)***.

3. A Formal Model for DNA Computing and its Computational Power

One aspect of theoretical research on DNA computing is the search for a suitable formal model to describe molecular computations. This approach often compares the computational power of such a model to the power of a Turing machine, which is the formal model of today's electronic computers.

We illustrate this type of research by *contextual insertion/deletion systems* ***(43,51)*** a formal language model of DNA computing. We show that this model of DNA computation, besides being feasible in the laboratory, has the full power of a Turing machine.

Before formally stating the model, we summarize its terminology ***(56)***. For a set Σ, card(Σ) denotes its cardinality, that is, the number of elements in Σ. An *alphabet* is a finite nonempty set. Its elements are called *letters* or *symbols*. The letters will usually be denoted by the first letters of the alphabet, with or without indices, i.e., a, b, C, D, a_i, b_j, and so on. (In the case of DNA computing, the alphabet at our disposal is $\Sigma = \{A, C, G, T\}$.) If $\Sigma = \{a_1, a_2, \ldots, a_n\}$ is an alphabet, then any sequence $w = a_{i1}a_{i2} \ldots a_{ik}$, $k \geq 0$, $a_{ij} \in \Sigma$, $1 \leq j \leq k$ is called a *string* (*word*) over Σ. The length of the word w is denoted by $|w|$ and, by definition, equals k. The words over Σ will usually be denoted by the last

letters of the alphabet, with or without indices, for example x, y, w_j, u_i, and so on. The set of all words consisting of letters from Σ will be denoted by Σ^*.

As a formal language operation, the *contextual insertion* is a generalization of catenation and insertion of strings and languages, ***(57)***: Words can be inserted into a string only if certain *contexts* are present. More precisely, given a set of contexts we add the condition that insertion of a word can be performed only between a pair of words in the context set. Analogously, contextual deletion allows erasing of a word only if the word is situated between a pair of words in the context set.

Besides being theoretically interesting, one of the motivations for studying insertions and deletions is their relevance to laboratory manipulation. Indeed, by using synthetic oligonucleotides and the technique of PCR site-directed mutagenesis ***(58)***, one can insert and delete oligonucleotide sequences in a variety of given contexts.

Kari et al. ***(43,51)*** investigated the mathematical properties of contextual insertions and deletions (below we refer to them as simply insertions and deletions): One of their results is that the actions of every Turing machine can be simulated entirely by insertion and deletion rules. Beaver ***(40)*** proposed that a similar operation, base substitution, simulates a universal Turing machine.

Using insertion-deletion systems, we briefly present several characterizations of recursively enumerable (RE) languages (the equivalents of the Turing machine model of computation). Such a system generates the elements of a language by inserting and deleting words, according to their contexts. Grammars based on insertion rules were already considered ***(59)*** with linguistic motivation. Insertion/deletion operations are also basic to DNA and RNA processing, particular RNA splicing and editing reactions ***(60)***. Our results show that these operations, even with strong restrictions on the length of the contexts and/or on the length of the inserted/deleted words, are computationally complete, that is, they can simulate the work of any Turing machine.

An insertion-deletion (in/del) system, ***(43)***, is a construct

$$\gamma = (V, T, A, I, D)$$

where V is an alphabet, $T \subseteq V$, A is a finite subset of V^*, and I, D are finite subsets of $V^* \times V^* \times V^*$.

The alphabet T is the terminal alphabet of γ, A is the set of axioms, I is the set of insertion rules, and D is the set of deletion rules. An insertion (deletion) rule is written as a triple (u, z, v), which means that z can be inserted in (deleted from) the context (u, v), where u represents the left context and v represents the right context.

For $x, y \in V^*$ we say that x derives y and we write $x \Rightarrow y$ if one of the following two cases holds:

1. $x = x_1uvx_2$, $y = x_1uzvx_2$, for some $x_1, x_2 \in V^*$ and $(u, z, v) \in I$ (insertion)
2. $x = x_1uzvx_2$, $y = x_1uvx_2$, for some $x_1, x_2 \in V^*$ and $(u, z, v) \in D$ (deletion).

Denoting by $\Rightarrow^*$ the reflexive and transitive closure of the relation $\Rightarrow$, the language generated by γ is defined by

$$L(\gamma) = \{w \in T^* \mid x \Rightarrow^* w, \text{ for some } x \in A\}.$$

Informally, $L(\gamma)$ is the set of strings obtained from the initial axiom set A by repeated application of insertion and deletion rules.

An in/del system $\gamma = (V, T, A, I, D)$ is said to be of weight (n, m, p, q) if

$$\max\{|z| \mid (u, z, v) \in I\} = n,$$

$$\max\{|u| \mid (u, z, v) \in I \text{ or } (v, z, u) \in I\} = m,$$

$$\max\{|z| \mid (u, z, v) \in D\} = p,$$

$$\max\{|u| \mid (u, z, v) \in D \text{ or } (v, z, u) \in D\} = q.$$

Thus n (respectively p) represents the maximum length of the inserted (deleted) sequences, whereas m (respectively q) represent the maximum length of the right/left contexts of an insertion (respectively deletion).

We denote by $INS_n^m DEL_p^q$, $n, m, p, q \geq 0$, the family of languages $L(\gamma)$ generated by in/del systems of weight (n', m', p', q') such that $n' \leq n$, $m' \leq m$, $p' \leq p$, $q' \leq q$. When one of the parameters n, m, p, q is not bounded, we replace it by ∞. Thus, the family of all in/del languages is $INS_\infty^\infty DEL_\infty^\infty$.

The main results obtained regarding insertion and deletion systems are:

Theorem 1 ***(34)*** $RE = INS_3^6 del_2^7$.
Theorem 2 ***(35)*** $RE = INS_1^2 DEL_1^1$.
Theorem 3 ***(35)*** $RE = INS_2^1 DEL_2^0$.
Theorem 4 ***(35)*** $RE = INS_1^2 DEL_2^0$.

The interpretation of Theorem 1 is that the actions of every Turing machine can be simulated by an insertion/deletion system with finitely many rules, where the length of inserted strings is at most 3, and the length of the right and left contexts of insertion is at most 6, whereas the length of deleted strings is at most 2 and the length of the right and left contexts of deletion is bounded by 7. This suggests the possibility of using PCR site directed mutagenesis to simulate a Turing machine. Theorems 2–4 show that the same computational power can be obtained even with shorter contexts and inserted/deleted strings. These results point to yet another possible way of implementing biocomputations, namely by using RNA editing ***(60)*** which consists of insertions and deletions

```
DNA     G     G      GTTTTGG    AGA        G ATTTGG   A
RNA    uGuuuuGuuuuuGUUUUGGuuuuAGAuuuuuuuuGuAU**GGuuAuuu
```

Fig. 3. RNA editing by *u* insertion/deletion. Comparison of an edited RNA sequence encoding *H. mariadeanei* cytochrome oxidase subunit III (bottom) with its genomic DNA copy (top) ***(60)***. DNA sequences in upper case; uridines in mRNA that are added by RNA editing in lowercase (boldface); two encoded thymidines deleted from the mRNA indicated by asterisks.

of a single nucleotide. The most recent result, ***(51)***, proves that faithful restricted in/del systems have universal Turing machine power, where a faithful restricted in/del system has insertions and deletions of one letter only, but the length of contexts and inserted/deleted sequences is not bounded.

Overall, the general result that contextual insertions and deletions, by either site-directed mutagenesis or RNA editing, are sufficient to simulate the actions of a Turing machine suggests the existence of many platforms for biomolecular computing.

4. Nature's Solutions to Computational Problems

Research in molecular computing will undoubtedly have a great impact on many aspects of science and technology. In particular, molecular computing sheds new light on the very nature of computation, while it also introduces the prospect of designing computing devices that differ radically from today's computers. Probing the limits of biomolecular computation both *in vitro* and *in vivo* may provide new insights into the informational capacity of DNA in cellular organisms and the range of computational processes that exist in nature.

4.1. RNA Editing

Already, we have shown that computational processes exist in a variety of single- and multicellular organisms whose RNA molecules undergo RNA editing ***(61)***. Found in a wide variety of eukaryotes, from parasitic protozoa to humans, RNA editing by addition, deletion, or substitution of nucleotides alters the sequence of a messenger RNA before translation into protein. For example, **Fig. 3** shows a gene with an enormous number of uridine (*U*) insertions. (Sequence information in RNA is encoded in *A*, *C*, *G*, or *U*, with *U* replacing *T*.) In organisms such as trypanosomes RNA editing adds and deletes literally hundreds of uridine residues. These create initiation and termination codons, alter the structural features of transcripts, and construct over 90% of the coding capacity of this gene. On average, *U* insertions and deletions contribute to more than 60% of the nucleotides contained many genes. The other bases–A, C, and G–are completely conserved between the DNA and the RNA sequence ***(60)***.

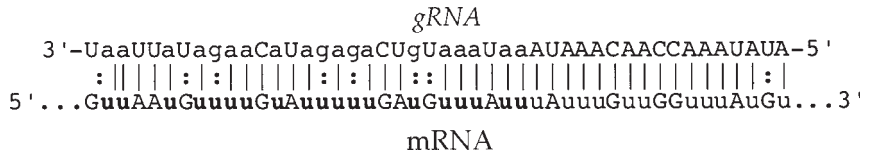

Fig. 4. Guide RNA–messenger RNA base-pairing interactions direct RNA editing. Lowercase *a*'s and *g*'s in the top gRNA sequence base pair with and guide the insertion of boldface lowercase **u**'s in this portion of the bottom messenger RNA sequence.

RNA editing restores coding messenger RNA (mRNA) sequences from encrypted pieces of the genome. Base-pairing interactions between small "guide RNA" (gRNA) molecules and the "pre-edited mRNA molecule" provide the context for determining these insertions and deletions (**Fig. 4**). Astonishingly, this process can create a single conserved protein coding sequence from over a dozen or so RNA molecules, each encoded in a unique circular DNA molecule (with the gene itself located on a *maxicircle* and the genes for each guide RNA usually found on one of the thousands of *minicircles*).

For every inserted U in the messenger RNA sequence, a corresponding A or G in the gRNA pairs with the fully edited product (**Fig. 4**). Complete editing proceeds 3′ to 5′ on the mRNA and requires a full set of overlapping gRNAs. Editing by each guide RNA creates an anchor sequence for binding the next guide RNA, leading to an ordered cascade of insertions and deletions—a genuinely RNA-based computer (***61***).

4.2. Gene Unscrambling

Ciliated protozoa possess two types of nuclei: an active macronucleus (soma) and a functionally inert micronucleus (germline) that contributes only to sexual reproduction. The macronucleus develops from the micronucleus after sexual reproduction. The micronuclear copies of some protein-coding genes in hypotrichous ciliates are obscured by intervening nonprotein-coding DNA sequences (internally eliminated sequences, or IESs) which must be removed before the assembly of a functional macronuclear DNA copy. Furthermore, the protein-coding DNA segments (macronuclear destined sequences, or MDSs) in *Oxytricha* and *Stylonychia* are sometimes present in a permuted order relative to their final position in the macronuclear copy. For example, we have found that the gene encoding DNA polymerase α in *S. lemnae* is scrambled in several dozens of pieces in the micronucleus. Destined to unscramble its micronuclear genes by putting the pieces together again, *O. trifallax* impressively solves a potentially complicated computational

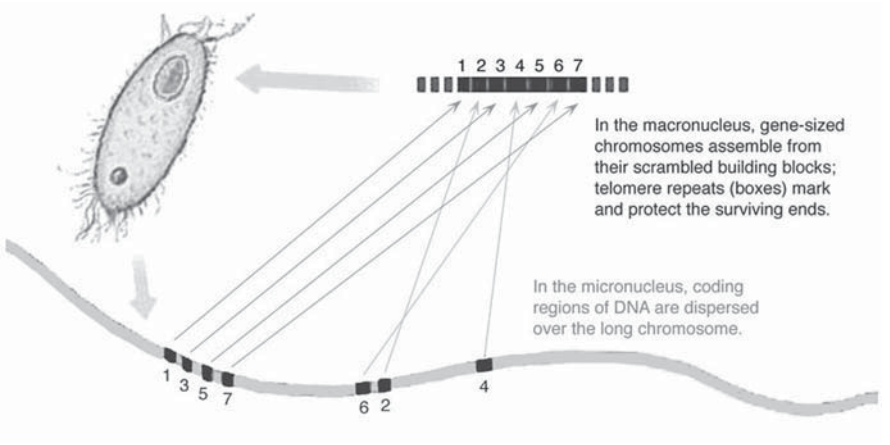

Fig. 5. Gene unscrambling as a computational problem. Dispersed coding MDSs, such as 1–3–5–7–6–2–4 (bottom), reassemble during macronuclear development to form the functional gene copy (top). Telomere addition marks and protects the ends of the gene, replacing the role of PCR primers in Step 2 of Adleman's experimental computation, because only those strands that have telomeres at both ends survive (***61***).

problem when assembling its functional sequences from their smaller constituents ***(61)***.

The process of unscrambling bears a striking resemblance to the DNA algorithm Adleman ***(1)*** used to solve a seven-city instance of the Directed Hamiltonian Path problem. The developing ciliate macronuclear 'computer' (**Fig. 5**) makes use of the information contained in short 2–14 nucleotide direct repeats. These act as guides in a series of homologous recombination events. For example, the DNA sequence present at the junction between MDS *n* and the downstream IES is generally the same as the sequence between MDS *n* +1 and its upstream IES, leading to correct ligation of MDS *n* to MDS *n* + 1. By providing the splints analogous to edges in Adleman's graph, this mechanism assembles protein-encoding segments (MDSs, or 'cities' or nodes in this graph) in the correct order in which they belong in the final protein coding sequence ("Hamiltonian Path"), though the details of this mechanism are still unknown. As such, the unscrambling of gene sequences accomplishes an astounding feat of cellular computation, especially as Hamiltonian Path Problems of this size (approx 50 nodes) present a challenge to any computer.

Together RNA editing and gene unscrambling provide a unique array of potentially usable paradigms for biological computation. Furthermore, these processes underscore the diversity of computational paradigms that exist in

biological systems and suggest a plethora of models for importing biology into mathematics.

References

1. Adleman, L. (1994) Molecular computation of solutions to combinatorial problems. *Science* **266,** 1021–1024.

1a. Kari, L. (1997) DNA computing: arrival of biological mathematics. *The Mathematical Intelligencer,* **19,** 2, 9–22.

2. Baum, E. (1995) Building an associative memory vastly larger than the brain. *Science* **268,** 583–585.
3. Reif, J. (1995) Parallel molecular computation: models and simulations, in *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, Santa Barbara, CA, pp. 213–223.
4. Kendrew, J. (eic) (1994) *The Encyclopedia of Molecular Biology*, Blackwell Science, Oxford.
5. Garey, M. and Johnson, D. (1979) *Computers and Intractability. A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, San Francisco.
6. Gifford, D. K. (1994) On the path to computation with DNA. *Science* **266,** 993–994.
7. Kaplan, P., Cecchi, G., and Libchaber, A. (1995) *Molecular computation: Adleman's experiment repeated.* NEC Technical Report.
8. Guarnieri, F., Fliss, M., and Bancroft, C. (1996) Making DNA add. *Science,* **273,** 220–223.
9. Liu, Q., Guo, Z., Condon, A., Corn, R., Lagally, M., and Smith, L. (1999) A surface-based approach to DNA computation, in *DNA Based Computers II* (L. F. Landweber and E. B. Baum, eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 123–132.
10. Ouyang, Q., Kaplan, P. D., Liu, S., and Libchaber, A. (1997) DNA solution of the maximal clique problem. *Science*, **278,** 446–9.
11. Cukras, A., Faulhammer, D., Lipton, R., and Landweber, L. F. (1998). Chess games: a model for RNA-based computation, in *Proceedings of the Fourth International Meeting on DNA Based Computers* (Kari, L., Rubin, H., and Wood, D. H., eds.), University of Pennsylvania, Philadelphia, PA, pp. 27–37.
12. Deaton, R., Murphy, R., Rose, J., Garzon, M., Franceschetti, D., and Stevens, S. (1997) A DNA based implementation of an evolutionary search for good encodings for DNA computation, in *Proceedings of the IEEE International Conference on Evolutionary Computation,* Indianapolis, IN, IEEE, Piscataway, NJ, pp. 267–271.
13. Kaplan, P., Cecchi, G., and Libchaber, A. (1999) DNA-based molecular computation: template-template interactions in PCR, in *DNA Based Computers II*

(Landweber, L. F. and Baum, E. B., eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 97–104.

14. Winfree, E., Yang, X., and Seeman, N. (1999) Universal computation via self-assembly of DNA: some theory and experiments, in *DNA Based Computers II* (Landweber, L. F. and Baum, E. B., eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 191–213.
15. Seeman, N., Wang, H., Liu, B., Qi, J., Li, X., Yang, X., Liu, F., Sun, W., Shen, Z., Sha, R., Mao, C., Wang, Y., Zhang, S., Fu, T.-J., Du, S., Mueller, J. E., Zhang, Y., and Chen, J. (1999) The perils of polynucleotides: the experimental gap between the design and assembly of unusual DNA structures, in *DNA Based Computers II* (Landweber, L. F. and Baum, E. B., eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 215–233.
16. Arita, M., Hagiya, M., and Suyama A., (1997) Joining and rotating data with molecules, in *Proceedings of the IEEE International Conference on Evolutionary Computation*, Institute of Electrical and Electronics Engineers (IEEE), pp. 243–248.
17. Arita, M., Suyama, A., and Hagiya, M., (1997) A heuristic approach for Hamiltonian Path Problem with molecules, in *Genetic Programming 1997: Proceedings of the Second Annual Conference* (Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., and Riolo, R. L., eds.), Stanford University, Palo Alto, CA, Morgan Kaufmann, pp. 457–462.
18. Hagiya, M. and Arita M. (1999) Towards parallel evaluation and learning of Boolean μ-formulas with molecules, in *DNA Based Computers III* (D. H. Wood, ed.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence, RI, in press
19. Jonoska, N. and Karl, S. (1997) Ligation experiments in computing with DNA. *Proceedings of the IEEE International Conference on Evolutionary Computation*, Indianapolis, IN, IEEE, Piscataway, NJ, pp. 261–266.
20. Mulawka, J., Weglenski, P., and Borsuk, P. (1998). Implementation of the Inference Engine based on Molecular Computing Technique, in press.
21. Gloor, G., Kari, L., Gaasenbeek, M., and Yu, S. (1998) Towards a DNA solution to the Shortest Common Superstring Problem, in *Proceedings of the IEEE International Joint Symposia on Intelligence and Systems*, Rockville, MD, IEEE Computer Society Press, Los Alamitos, CA, pp. 140–145.
22. Lipton, R. (1995) DNA solution of hard computational problems. *Science,* **268,** 542–545.
23. Boneh, D., Dunworth, C., and Lipton, R. J. (1996). Breaking DES using a molecular computer, in *DNA Based Computers: Proceedings of a DIMACS Workshop* (Lipton, R. J. and Baum, E. B., eds.), American Mathematical Society, Providence, RI, 27, pp. 37–65.

24. Adleman, L., Rothemund, P., Roweis, S., and Winfree, E. (1999) On applying molecular computation to the Data Encryption Standard, in *DNA Based Computers II* (Landweber, L. F. and Baum, E. B., eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 31–44.
25. Leete, T., Schwartz, M., Williams, R., Wood, W., Salem, J., and Rubin, H. (1999) Massively parallel DNA computation: expansion of symbolic determinants, in *DNA Based Computers II* (Landweber, L. F. and Baum, E. B., eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 45–58.
26. Oliver, J. (1997) Matrix multiplication with DNA. *Journal of Molecular Evolution*, **45,** 161–7.
27. Baum, E. and Boneh, D. (1999) Running dynamic programming algorithms on a DNA computer, in *DNA Based Computers II* (Landweber. F. and Baum, E. B., eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 77–85.
28. Jonoska, N. and Karl, S. (1999) A molecular computation of the road coloring problem, in *DNA Based Computers II* (Landweber, L. F. and Baum, E. B., eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 87–96.
29. Williams, R. and Wood, D. (1999) Exascale computer algebra problems interconnect with molecular reactions and complexity theory, in *DNA Based Computers II* (Landweber, L. F. and Baum, E. B., eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 267–275.
30. Kari, L., Gloor, G., and Yu, S. (1999) Using DNA to solve the Bounded Post Correspondence Problem. *Theoretical Computer Science*, in press.
31. Kobayashi, S., Yokomori, T., Sampei, G., and Mizobuchi, K. (1997) DNA implementation of simple Horn clause computation, in *Proceedings of the IEEE International Conference on Evolutionary Computation*, Indianapolis, IN, IEEE, Piscataway, NJ, pp. 213–217.
32. Adleman, L. (1996) On constructing a molecular computer, in *DNA Based Computers: Proceedings of a DIMACS Workshop* (Lipton, R. J. and Baum, E. B., eds.), American Mathematical Society, Providence, RI, pp. 1–21.
33. Amos, M., Gibbons, A., and Hodgson, D. (1999) Error-resistant implementation of DNA computation, in *DNA Based Computers II* (Landweber, L. F. and Baum, E. B., eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 151–161.
34. Lipton, R. (1996) Speeding up computations via molecular biology, in *DNA Based Computers* (Lipton, R. J. and Baum, E. B., eds.), American Mathematical Society, Providence, RI, pp. 67–74.
35. Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N., Goodman, M., Rothemund, P., and Adleman, L. (1999) A sticker based architecture for DNA computation, in

DNA Based Computers II (Landweber, L. F. and Baum, E. B., eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 1–29.
36. Ogihara, M. and Ray, A. (1998). The minimum DNA computation model and its computational power. University of Rochester, Technical report TR-672.
37. Beaver, D. (1995) Computing with DNA. *J. Comput. Biol.*, **2,** 1–7.
38. Smith, W. (1996) DNA computers *in vitro* and *in vivo*, in *DNA Based Computers: Proceedings of a DIMACS Workshop* (Lipton, R. J. and Baum, E. B., eds.), American Mathematical Society, Providence, RI, DIMACS series, 27, pp. 121–185.
39. Winfree, E. (1996) On the computational power of DNA annealing and ligation, in *DNA Based Computers: Proceedings of a DIMACS Workshop* (Lipton, R. J. and Baum, E. B., eds.), American Mathematical Society, Providence, RI, DIMACS series, vol. 27, pp. 199–221
40. Beaver, D. (1996) A universal molecular computer, in *DNA Based Computers: Proceedings of a DIMACS Workshop* (Lipton, R. J. and Baum, E. B., eds.), American Mathematical Society, Providence, RI, pp. 29–36.
41. Head, T. (1987) Formal language theory and DNA: an analysis of the generative capacity of recombinant behaviors. Bulletin of Mathematical Biology, **49,** 737–759.
42. Rothemund, P. (1996) A DNA and restriction enzyme implementation of Turing machines, in *DNA Based Computers: Proceedings of a DIMACS Workshop* (Lipton, R. J. and Baum, E. B., eds.), American Mathematical Society, Providence, RI, pp. 75–119.
43. Kari, L., and Thierrin, G. (1996) Contextual insertions/deletions and computability. *Information and Computation,* **131,** 47–61.
44. Yokomori, T. and Kobayashi, S. (1999) DNA-EC: a model of DNA computing based on equality checking, in *DNA Based Computers III* (in Wood, D. H., ed.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence, RI. , in press.
45. Head, T., Paun, G., and Pixton, D. (1996) Language theory and molecular genetics, in *Handbook of Formal Languages* (Rozenberg, G. and Salomaa, A., eds.), Springer Verlag, Berlin, **2,** 295–358.
46. Paun, G. and Salomaa, A. (1996) DNA computing based on the splicing operation. *Mathematica Japonica,* **43,** 3, 607–632.
47. Paun, G. (1995) On the power of the splicing operation. *International Journal of Computer Mathematics,* **59,** 27–35.
48. Freund, R., Kari, L., and Paun, G. (1999) DNA computing based on splicing: the existence of universal computers. *Theory of Computing Systems* **32,** 69–112.
49 Csuhaj-Varju, E., Freund, R., Kari, L., and Paun, G. (1996). DNA computing based on splicing: universality results, in *Proceedings of 1st Annual Pacific Symposium on Biocomputing*, Hawaii (Hunter, L. and Klein, T., eds.), World Scientific Publ., Singapore, pp. 179–190.
50. Yokomori, T., Kobayashi, S., and Ferretti, C. (1997) On the power of circular

splicing systems and DNA computability, in *Proceedings of the IEEE International Conference on Evolutionary Computation*, Indianapolis, IN, IEEE, pp. 219–224.
51. Kari, L., Paun, G., Thierrin, G., and Yu, S. (1999) At the crossroads of DNA computing and formal languages: characterizing recursively enumerable languages using insertion/deletion systems, in *DNA Based Computers III* (Wood, D. H., ed.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence, RI, in press.
52. Hartmanis, J. (1995) On the weight of computations. *Bulletin European Association of Theoretical Computer Science,* **55,** 136–138.
53. Kurtz, S., Mahaney, S., Royer, J., and Simon, J. (1999) Active transport in biological computing, in *DNA Based Computers II* (Landweber, L. F. and Baum, E. B., eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 171–179.
54. Amenyo, J. (1999) Mesoscopic computer engineering: automating DNA-based molecular computing via traditional practices of parallel computer architecture design, in *DNA Based Computers II* (Landweber, L. F. and Baum, E. B., eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, American Mathematical Society, Providence, RI, pp. 133–150.
55. Mihalache, V. (1997) Prolog approach to DNA computing, in *Proceedings of the IEEE International Conference on Evolutionary Computation*, Indianapolis, IN, IEEE, pp. 249–254.
56. Salomaa, A. (1973) *Formal Languages*. Academic Press, New York.
57. Kari, L. (1991) *On insertions and deletions in formal languages*. Ph.D. thesis, University of Turku, Finland.
58. Dieffenbach, C. W. and Dveksler, G. S., (eds.), (1995) *PCR primer: a laboratory manual,* Cold Spring Harbor, NY, Cold Spring Harbor Laboratory Press, pp. 581–621.
59. Galiukschov, B. S. (1981) Semicontextual grammars (in Russian). *Mat. logica i mat. ling.*, Kalinin Univ., 38–50.
60. Landweber, L. F. and Gilbert, W. (1993). RNA editing as a source of genetic variation. *Nature* **363,** 179–182.
61. Landweber, L. F. and Kari, L. (1998) The Evolution of DNA Computing: Nature's Solution to a Combinatorial Problem, in *Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22–25, 1998*, (Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., and Riolo, R. L., eds), University of Wisconsin, Madison, WI, San Francisco, CA, Morgan Kaufmann, pp. 700–708.