From Micro-soft to Bio-soft: Computing with DNA

Lila Kari*
Department of Computer Science, University of Western Ontario
London, Ontario, N6A 5B7 Canada
website: http://www.csd.uwo.ca/~lila
email: lila@csd.uwo.ca

"Jump at the sun and you might at least catch hold of the moon"
(Jamaican proverb)

1 From Digits and Beads to Bits and Biomolecules

The notion of *computing* seems nowadays to be so synonymous with *computers*, that we often seem to forget that electronic computers are relatively new players on the world stage, [31].

Indeed, 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 constructions. The means used for performing calculations were whatever was available, and thus gradually progressed from manual to mechanical, and from there on to electrical devices.

Indeed, man started off by counting on his digits, a fact attested by the use of the word digit to mean both "any of the ten numbers from 0 to 9" and "a finger, thumb or toe" (*Oxford Advanced Learner's Dictionary*). The need for counting and tracking occurrences in the physical world is witnessed by primitive calendars like the one at Stonehenge, 2,800 B.C., or by primitive calculators like the abacus. The abacus, the most common of which comes from China, was man's first attempt at automating the counting process, and it involved the idea of positional representation: the value assigned to each bead (pebble, shell) was determined not by its shape but by its position.

The transition to a qualitatively superior way of doing computation had to wait until the 17th century when Pascal built the first *mechanical adding machine* (1642), based on a gear system. In his machine, based on the design of Hero of Alexandria (2 A.D.), a wheel engaged its single tooth with a ten-teeth

*This paper was written during my visit in Japan supported by the "Research for the Future" Program, JSPS-RFTF 96I00101 of the Japan Society for the Promotion of Science.

wheel once every time it revolved, the result being that it had to make ten revolutions in order to completely turn the 10-teeth wheel once.

Later on, Charles Babbage (1812) theorized, but never built, a fully-automatic steam driven calculating machine. In 1823 he envisioned a better idea, of an "analytical engine" that would have conditional control and where the instructions were to be stored on punched cards.

The introduction in 1890 of the punched cards by Herman Hollerith, whose firm later became International Business Machine (IBM), marked the transition from the decimal system to the binary system. The binary system had two states, 0 and 1, corresponding to the place in the card having or not a hole punched in to it.

The binary representation could take advantage of many two-state devices like card readers, electric circuits (on/off states) and vacuum tubes, and it led to the next stage in the evolution of computing: the *electronic* one.

The theoretical foundation of modern electronic computers was laid by Alan Turing who described in 1936, [58], a hypothetical device that became known as a *Turing machine*. A Turing machine can read, write and erase symbols written on squares of an infinite tape. Its endless tape can be interpreted as a general purpose internal memory, on which operations of read, write and erase are performed as in any modern day RAM.

The first high-speed electronic digital computer was the ENIAC (Electrical Numerical Integrator and Calculator), which used almost 18,000 vacuum tubes stored on 167 square meters, had input/output on punch cards, and could perform a few arithmetical operations. However, the instructions of a program had to be "wired-in" separately for each problem. John von Neumann addressed this issue and laid the principles of how a practical computer should be organized and built. His main ideas were the introduction of a conditional go-to instruction and the storing of the data and the program together in the same memory unit, which meant that the machine itself could alter either its data or program. This lead to the 1st generation of modern *programmable electronic computers* (1947), which used random access memory (RAM), had punch-card or punch-tape input and output, and were the size of a great piano. They still required considerable maintenance, attained only 70 % to 80 % reliable operation and could be used for 8 to 12 years.

It was two technological advances, the discovery of the transistor in 1947 and building of the first integrated circuits in 1958, that truly released the potential of electronic computers. In 1971 Intel released the first microprocessor. In 1975 Bill Gates and Paul Allen wrote a BASIC (Beginners All purpose Symbolic Instruction Code) compiler for ALTAIR 8800, hailed as the world's first minicomputer kit to rival commercial models. In the same year, the first software company, Microsoft, was born. The PC explosion that followed and the increasing popularity of the World Wide Web are witnesses of the computer revolution, the fastest growing technology in man's history.

The above mini-incursion into the history of computing is meant to point out that electronic computers are only the latest in a long chain of man's attempts to use the best technology available for doing computations. While it is true that their appearance, some 50 years ago, has revolutionized computing, computing does not start with electronic computers, and there is no reason why it should end with them. Indeed, even electronic computers have their limitations: there is only so much data they can store and their speed thresholds determined by physical laws will soon be reached. The latest attempt to break down these barriers is to replace, once more, the tools for doing computations: instead of electrical use biological ones.

Research in this area was started by Leonard Adleman in 1994, [1], when he surprised the scientific community by using the tools of molecular biology to solve a hard computational problem. Adleman's experiment, solving an instance of the Directed Hamiltonian Path Problem solely by manipulating DNA strands, marked the first instance of a mathematical problem being solved by biological means. The experiment provoked an avalanche of computer science/molecular biology/biochemistry/physics research, while generating at the same time a multitude of open problems.

The excitement DNA computing incited was mainly caused by its capability of massively parallel searches. This, in turn, showed its potential to yield tremendous advantages from the point of view of *speed, energy consumption* and *density of stored information.* For example, in Adleman's model, [2], the number of operations per second was up to 1.2×10^{18}. This is approximately 1,200,000 times faster than the fastest supercomputer. While existing supercomputers execute 10^9 operations per Joule, the energy efficiency of a DNA computer could be 2×10^{19} operations per Joule, that is, a DNA computer could be about 10^{10} times more energy efficient (see [1]). Finally, according to [1], storing information in molecules of DNA could allow for an information density of approximately 1 bit per cubic nanometer, while existing storage media store information at a density of approximately 1 bit per 10^{12} nm^3. As estimated in [8], a single DNA memory could hold more words than all the computer memories ever made.

The research in the field has, from the beginning, had both experimental and theoretical aspects; for an overview of the research on DNA computing see [37].

The experiments that have actually been carried out are not numerous so far. P. Kaplan, [34], replicated Adleman's experiment; a Wisconsin team of computer scientists and biochemists made partial progress in solving a 5-variable instance of the SAT problem by using a surface-based approach, [46]; F.Guarnieri, M.Fliss and C.Bancroft have used a horizontal chain-reaction for DNA-based addition, [26]. At the same time, various aspects of the implementability of DNA computing have been experimentally investigated: in [19] the effect of good encodings on the solutions of Adleman's problem were addressed; the complications raised by using the Polymerase Chain Reaction were studied in [35]; the usability of self-assembly of DNA was studied in [64]; the experimental gap between the design and assembly of unusual DNA structures was pointed out in [56]; joining and rotating data with molecules was reported in [6]; concatenation with PCR was studied in [6], [7]; evaluating simple Boolean formulas was started in [27]; ligation experiments in computing with DNA were conducted in

[33] .

The theoretical work on DNA computing consists, on one side, of designing potential experiments for solving various problems by means of DNA manipulation. Descriptions of such experiments include the Satisfiability Problem [44], breaking the Data Encryption Standard [15], expansions of symbolic determinants [43], matrix multiplication [49], graph connectivity and knapsack problem using dynamic programming [10], road coloring problem [32], exascale computer algebra problems [61], and simple Horn clause computation [42]. On the other side, the theoretical research on DNA computing comprises attempts to model the process in general, and to give it a mathematical foundation. To this aim, models of DNA computing have been proposed and studied from the point of view of their computational power and their in-vitro feasibility (see, for example, [1], [2], [12], [29], [38], [44], [63], [59]). There are pro's and con's for each of the proposed models. The ones based on a combination of Adleman's bio-operations have the advantage that they are already implementable, the downside being their involving extensive manual manipulation of test tubes and serious possibilities of errors. On the other hand, the models based on splicing, equality checking or on contextual insertion and deletion, [29], [30], [51], [50], [21], [18], [59], [60], [38], [39], have the advantage that they represent "one-pot" molecular computers based solely on hybridization and the actions of enzymes. Moreover, they are formal models with the results backed up by mathematical proofs. Their disadvantage is that they have yet to be implemented in the laboratory. Overall, the existence of different models with complementing features shows the versatility of DNA computing and increases the likelihood of practically constructing a DNA computing–based device.

A few more words, as to why we should prefer biomolecules to electricity for doing computation: the short answer is that it seems more natural to do so. We could look at the electronic technology as just a technology that was in the right place at the right time; indeed, electricity hardly seems a suitable and intuitive means for storing information and for computations. For these purposes, nature prefers instead a medium consisting of biomolecules: DNA has been used for millions of years as storage for genetic information, while the functioning of living organisms requires computations. Such considerations seem to indicate that using biomolecules for computations is a promising new avenue, and that DNA computers might well be the successors of electronic computers.

2 From Mathematical Operations to Bio-operations

Any kind of computer, be it mechanical, electrical or biological, needs two basic capabilities in order to function: to store information and to perform operations on it. In the following I address both issues: how can information be stored in DNA strands, and what molecular biology techniques could potentially be used for computations. In order to distinguish between ordinary mathematical operations and bio-molecular procedures performed on DNA strands, I will use the term bio-operations to refer to the latter. For further details of molecular biology terminology, the reader is referred to [40].

DNA (deoxyribonucleic acid) is found in every cellular organism as the storage medium for genetic information. It is composed of units called nucleotides, distinguished by the chemical group, or base, attached to them. The four bases, are *adenine, guanine, cytosine* and *thymine*, abbreviated as A, G, C, and T. (The names of the bases are also commonly used to refer to the nucleotides that contain them.) Single nucleotides are linked together end–to–end to form DNA strands. A short single-stranded polynucleotide chain, usually less than 30 nucleotides long, is called an *oligonucleotide*. The DNA sequence has a *polarity*: a sequence of DNA is distinct from its reverse. The two distinct ends of a DNA sequence are known under the name of the $5'$ end and the $3'$ end, respectively. Taken as pairs, the nucleotides A and T and the nucleotides C and G are said to be *complementary*. Two complementary single–stranded DNA sequences with opposite polarity will join together to form a double helix in a process called *base-pairing* or *annealing*. The reverse process – a double helix coming apart to yield its two constituent single strands – is called *melting*.

A single strand of DNA can be likened to a string consisting of a combination of four different symbols, A, G, C, T. Mathematically, this means we have at our disposal a 4 letter alphabet $\Sigma = \{A, G, C, T\}$ to encode information, which is more than enough, considering that an electronic computer needs only two digits, 0 and 1, for the same purpose.

As concerning the operations that can be performed on DNA strands, the proposed models of DNA computation are based on various combinations of the following primitive *bio-operations*:

– *Synthesizing* a desired polynomial–length strand, used in all models. In standard solid phase DNA synthesis, a desired DNA molecule is built up nucleotide by nucleotide on a support particle in sequential coupling steps. For example, the first nucleotide (monomer), say A, is bound to a glass support. A solution containing C is poured in, and the A reacts with the C to form a two-nucleotide (2-mer) chain AC. After washing the excess C solution away, one could have the C from the chain AC coupled with T to form a 3-mer chain (still attached to the surface) and so on.

– *Mixing*: pour the contents of two test tubes into a third one to achieve union, [1], [2], [5], [45], [54]. Mixing can be performed by rehydrating the tube contents (if not already in solution) and then combining the fluids together into a new tube, by pouring and pumping for example.

– *Annealing*: bond together two single-stranded complementary DNA sequences by cooling the solution. (See [11], [16], [54], [57], [63].) Annealing in vitro is also known as *hybridization*.

– *Melting*: break apart a double-stranded DNA into its single-stranded complementary components by heating the solution. (See [11], [16], [54], [57], [63].) Melting in vitro is also known under the name of *denaturation*.

– *Amplifying (copying)*: make copies of DNA strands by using the Polymerase Chain Reaction (PCR). (See [1], [2], [5], [11], [12], [13], [16], [43], [45], [57].) PCR is an in vitro method that relies on DNA polymerase to quickly amplify specific DNA sequences in a solution. Indeed, the *DNA polymerase* enzymes perform several functions including replication of DNA. The replication reaction requires a guiding DNA single-strand called *template*, and a shorter oligonucleotide called a *primer*, that is annealed to it. Under these conditions, DNA polymerase catalyzes DNA synthesis by successively adding nucleotides to one end of the primer. The primer is thus extended in one direction until the desired strand that starts with the primer and is complementary to the template is obtained.

PCR involves a repetitive series of temperature cycles, with each cycle comprising three stages: denaturation of the guiding template DNA to separate its strands, then cooling to allow annealing to the template of the *primer* oligonucleotides, which are specifically designed to flank the region of DNA of interest, and, finally, extension of the primers by DNA polymerase. Each cycle of the reaction doubles the number of target DNA molecules, the reaction giving thus an exponential growth of their number.

– *Separating* the strands by length using a technique called gel electrophoresis that makes possible the separation of strands by length. The molecules are placed at the top of a wet gel, to which an electric field is applied, drawing them to the bottom. Larger molecules travel more slowly through the gel. After a period, the molecules spread out into distinct bands according to size. (See [1], [2], [5], [11], [12], [13], [16].)

– *Extracting* those strands that contain a given pattern as a substring by using affinity purification. This process permits single strands containing a given subsequence v to be filtered out from a heterogeneous pool of other strands. After synthesizing strands complementary to v and attaching them to magnetic beads, the heterogeneous solution is passed over the beads. Those strands containing v anneal to the complementary sequence and are retained. Strands not containing v pass through without being retained. (See [1], [2], [11], [13], [16], [45].)

– *Cutting* DNA double-strands at specific sites by using commercially available restriction enzymes. One class of enzymes, called *restriction endonucleases*, will recognize a specific short sequence of DNA, known as a *restriction site*. Any double-stranded DNA that contains the restriction site within its sequence is cut by the enzyme at that location. (See [11], [12], [13], [16], [29], [53], [57].)

– *Ligating*: paste DNA strands with compatible sticky ends by using DNA ligases. Indeed, another enzyme called *DNA ligase*, will bond together, or "ligate", the end of a DNA strand to another strand. (See [11], [12], [13], [29], [53], [57], [63].)

– *Substituting*: substitute, insert or delete DNA sequences by using PCR site-specific oligonucleotide mutagenesis (see [13], [38]). The process is a variation of PCR in which a change in the template can be induced by the process of primer modification. Namely, one can use a primer that is only partially complementary to a template fragment. (The modified primer should contain enough bases complementary to the template to make it anneal despite the mismatch.) After the primer is extended by the polymerase, the newly obtained strand will consist of the complement of the template in which a few oligonucleotides have been "substituted" by other, desired ones.

– *Marking* single strands by hybridization: complementary sequences are at-

tached to the strands, making them double-stranded. The reverse operation is *unmarking* of the double-strands by denaturing, that is, by detaching the complementary strands. The marked sequences will be double-stranded while the unmarked ones will be single-stranded, [5], [46], [54].
– *Destroying* the marked strands by using exonucleases, (see [46]), or by cutting all the marked strands with a restriction enzyme and removing all the intact strands by gel electrophoresis, [5]. (By using enzymes called *exonucleases*, either double-stranded or single-stranded DNA molecules may be selectively destroyed. The exonucleases chew up DNA molecules from the end inward, and exist with specificity to either single-stranded or double-stranded form.)
– *Detecting* and *Reading*: given the contents of a tube, say "yes" if it contains at least one DNA strand, and "no" otherwise, [1], [2], [5], [11], [16], [45]. PCR may be used to amplify the result and then a process called *sequencing* is used to actually read the solution. The basic idea of the most widely used sequencing method is to use PCR and gel electrophoresis. Assume we have a homogeneous solution, that is, a solution containing mainly copies of the strand we wish to sequence, and very few contaminants (other strands). For detection of the positions of A's in the target strand, a blocking agent is used that prevents the templates from being extended beyond A's during PCR. As a result of this modified PCR, a population of subsequences is obtained, each corresponding to a different occurrence of A in the original strand. Separating them by length using gel electrophoresis will give away the positions where A occurs in the strand. The process can then be repeated for each of C, G and T, to yield the sequence of the strand. Recent methods use four different fluorescent dyes, one for each base, which allows all four bases to be processed simultaneously. As the fluorescent molecules pass a detector near the bottom of the gel, data are output directly to an electronic computer.

The bio-operations listed above, and possibly others, will then be used to write "programs". A "program" will receive a tube containing DNA strands encoding information as input, and return as output either "yes" or "no" or a set of tubes. A bio-computation will consists of a sequence of bio-operations performed on tubes containing DNA strands.

3 Who is afraid of lab errors?

Despite the progress achieved, the main obstacles to creating a practical DNA computer still remain to be overcome. These obstacles are roughly of two types, [2]:

– *practical*, arising primarily from difficulties in dealing with large scale systems and in coping with errors;

– *theoretical*, concerning the versatility of DNA computers and their capacity to efficiently accomodate a wide variety of computational problems.

In the following I will point out possible pitfalls and complications that might arise in the process of implementing each of the bio-operations enumerated in the preceding section. None of the problems encountered seems clearly insurmountable: being aware of their existence is the first step towards overcoming them.

To start from the beginning, *synthesis* of a DNA strand can sometimes result in the strand annealing to itself and creating a hairpin structure. Even the seemingly straightforward *mixing* operation can sometimes pose problems: if DNA is not handled gently, the sheer forces from pouring and mixing will fragment it. Also of concern for this operation is the amount of DNA which remains stuck to the walls of the tubes, pumps, pipette tips, etc., and is thus "lost" from the computation.

Hybridization has also to be carefully monitored because the thermodynamic parameters necessary for annealing to occur are sequence dependent. This is important because, depending on the conditions under which the DNA reactions occur, two oligonucleotides can hybridize without exact matching between their base pairs. *Hybridization stringency* refers to the number of complementary base pairs that have to match for DNA oligonucleotides to bond. It depends on reaction conditions (salt concentration, temperature, relative percentage of A's and T's to G's and C's, duration of the reaction) and it increases with temperature. One solution for increasing hybridization stringency is the choice of good encodings for the input information of the problem, [19]. Another solution proposed in [9] to avoid self annealing and mismatches is encoding using specially chosen sequences as spacers that separate the information bits.

Amplification by PCR is used with the assumption that by maintaining a surplus of primer to template one can avoid undesired template-template interactions. As pointed out in [35], this assumption is not necessarily valid. Indeed, experimental evidence points to the possibility of the creation of complex structures like folded DNA, complexes between several strands and incorrect ligation products. This might further affect the accuracy of using the gel electrophoresis technique for *separation of strands by length*. Indeed, in the presence of complex structures, the mobility of the strands will not depend only on their length, as desired, but also on the DNA conformation (shape). As a possible solution, the use of denaturing or single-stranded gels for analysis is recommended in [35]. Moreover, by keeping concentrations low, heteroduplex (double-strands with mismatches) formation and template-template interactions can be minimized.

Separation of strands by length and *extraction* of strands containing a given pattern can also be inefficient, and this might pose problems with scale-up of the test-tube approach. An alternative methodology has been proposed in [46]: the set of oligonucleotides is initially attached to a surface (glass, silicon, gold, or beads). They are then subjected to bio-operations such as marking, unmarking and destruction, in order to obtain the desired solution. This method greatly reduces losses of DNA molecules that occur during extraction by affinity purification. Its drawbacks are that it relies on *marking* and *unmarking* which, in turn, assume specificity and discrimination of single-base mismatches. While these processes have proved reliable when using 15-mer sequences, they become more difficult for shorter or longer polynucleotide strands. Another problem is that the scale of the computation is restricted by the two-dimensional nature of the surface-based approach: one cannot reach too high an information storing density.

Extraction of those strands that contain some given pattern is not 100% efficient, and may at times inadvertently retain strands that *do not* contain the specified sequence. While the error rate is reasonable in case only a few extractions are needed, if the number of extractions is in the hundreds or thousands, problems arise even if 95% efficiency of extraction is assumed. Indeed, the probability of obtaining a strand encoding the solution, while at the same time obtaining no strands encoding illegal solutions is quite low. As another possible solution, in [5] the operation *remove* was proposed as a replacement for *extract*. The compound operation *remove* removes from a set of strands all strings that contain at least one occurrence of a given sequence. The operation is achieved by first marking all the strands that contain the given sequence as a substring and then destroying the marked strands. The advantage of the method is that the restriction enzymes used for the remove operation have a far lower error rate than extraction. One of the drawbacks is that, although the initial tube might contain multiple copies of each strand, after many *remove* operations the volume of material may be depleted below an acceptable empirical level. This difficulty can be avoided by periodic amplification by PCR.

Cutting (cleavage) of a DNA strand by a restriction endonuclease is also referred to as digestion of the DNA by that enzyme. The process may sometimes produce partial digestion products. One must test all protocols for the effectiveness of the restriction enzyme used, and it is often necessary to find means to remove undigested material. Similarly, the accuracy of *ligation* is high, but not perfect. A ligase may ligate the wrong molecule to a sticky end, if it bears a close resemblance to the target molecule.

Detection and *sequencing* conventionally require enzymatic reactions and gel electrophoresis that are expensive and laborious processes. A possible solution to these drawbacks is using a technique that achieves sequencing by hybridization, offering a one-step automated process for reading the output, [48]. In this method, target strands are hybridized to a complete set of oligonucleotides synthesized on a flat solid surface (for example an array containing all the possible 8-mers) and then the target is reconstructed from the hybridization data obtained. However, to avoid errors arising from self-annealing, a restricted genetic alphabet is recommended with this method, using only two of the four bases. In this way, the test tube contents would be resistant to intramolecular reactions but not to intermolecular reactions.

Besides the accuracy of bio-operations, another peril of the implementation of DNA computations is the fact that the size of the problem influences the concentration of reactants, and this, in turn, has an effect on the rate of production and quality of final reaction products. In [41], an analysis of Adleman's experiment showed that an exponential loss of concentration occurs even on sparse digraphs, and that this loss of concentration can become a significant consideration for problems not much larger than those solved by Adleman. For volume decreasing DNA algorithms, an error-resistant solution seems to be the repeated application of PCR to the intermediate products, [17]. However, this cannot always be the solution, as not all algorithms are volume decreasing. Indeed, as pointed out in [28], one barrier to scalable DNA computation is the weight of DNA. In some cases, [3], to achieve the desirable error rate, approximately 23 Earth masses of DNA were needed. Clearly, this is not an acceptable situation, and a combination of *algorithm transformations* (changing algorithms by using intelligent space and time trade-offs) might be required to reduce the amount of DNA, [2], [3], [54].

Indeed, until now we have mainly been concerned with the perils and pitfalls of the biotechnology used to implement the bio-operations. This might have given the skewed impression that the only way to advance the DNA computing research is to wait for progresses in biotechnology. In fact, a complementary approach to improving the accuracy of DNA computing should come from the computer science side. It should include development of novel programming techniques, the use of probabilistic algorithms, genetic algorithms, heuristic approaches, ([7]), and other modifications of the classical mathematical problem-solving tools.

This section, which discussed implementation techniques and the associated error rates, indicates that many substantial engineering challenges to constructing a DNA computer remain at almost all stages. However, we want to point out that the issues of actively monitoring and adjusting the concentration of reactants, as well as fault tolerance, are all addressed by biological systems in nature: cells need to control the concentrations of various compounds, to arrange for rare molecules to react, and they need to deal with undesirable byproducts of their own activity. There is no reason why, when these problems have successfully been dealt with in vivo, they should not have solutions in vitro. As a step in this direction, in [41] a mechanism is suggested, based on membranes that separate volumes (vesicles) and on active systems that transport selected chemicals across membranes. Moreover, [4], [61], [47], [52], suggest how familiar computer design principles for electronic computers can be exploited to build molecular computers.

4 Current Research: a Sample

As mentioned before, the current research in DNA computing has two aspects: practical and theoretical. This section will give a sample of both, briefly describing an experimental project and some of the theoretical work of the DNA computing research team at the University of Western Ontario, Canada, that includes myself, Greg Gloor, Sheng Yu, Gabriel Thierrin, Yong Kang, Helmut Jürgensen and our overseas colleagues Gheorghe Păun and Arto Salomaa.

4.1 Experimental research

The problems chosen for our experiments are *The Shortest Common Superstring Problem* and *The Bounded Post Correspondence Problem*. The reasons for our choices were the following:

• Both problems are NP-complete, that is they are *hard* computational problems. This means, in particular, that they cannot yet be solved in real-time by electronic computers. (In fact, the question whether a real-time, i.e., polynomial time algorithm exists for NP-complete problems is still open.) Finding efficient DNA algorithms for solving them would thus indicate that DNA computing could be quantitatively superior to electronic computing, [24].
• Both experiments proposed for solving the problems use readily available reagents and techniques. The first problem is a good testing ground for Adleman's bio-operations, while the second one tests other standard molecular procedures for potential use in DNA computing.
• The second problem is a famous computer science problem. If the condition "bounded" were dropped, the resulting problem would be *unsolvable* by classical means of computation. The search for DNA solutions of this problem could thus give insights into the limitations of DNA computing, and shed light into the conjecture that DNA computing is a qualitatively new model of computation.

In the following I will only describe our DNA solution of the first problem, [25], that is currently the object of our experimental work. Before formally stating the problem, I summarize the notation used, [55]. 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 be usually denoted by the first letters of the alphabet, with or without indices, i.e., a, b, C, D, a_i, b_j, etc. (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 $w = a_{i_1}a_{i_2}\ldots a_{i_k}$, $k \geq 0$, $a_{i_j} \in \Sigma$, $1 \leq j \leq k$ 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, etc. The set of all words consisting of letters from Σ will be denoted by Σ^*.

The Shortest Common Superstring Problem [23]
INPUT: Alphabet Σ, finite set R of strings from Σ^*, and a positive integer K.
QUESTION: Is there a string $w \in \Sigma^*$ with length $|w| \leq K$ such that each string $x \in R$ is a substring of w, i.e., $w = w_0xw_1$, where each w_i is a string in Σ^*?

Note that the problem remains NP-complete even if card(Σ) = 2 or if all $x \in R$ have lengths $|x| \leq 8$ and contain no repeated symbols. On the other hand, the problem is solvable in polynomial time if all $x \in R$ have $|x| \leq 2$.

Restatement of the Problem in Molecular Terms
Given n oligonucleotide strings $x_1, x_2, \ldots, x_n$ of arbitrary lengths, and a positive number K, is there a nucleotide sequence w of length K that contains all the oligonucleotides $x_1, x_2, \ldots, x_n$ as subsequences? (The solution to this problem also provides a method for finding the minimum length sequence containing all the given oligonucleotides. Note that such a sequence always exists: the catenation $x_1x_2\ldots x_n$ of all nucleotides strings is a nucleotide sequence containing all

the given oligonucleotides. Due to possible overlaps, this catenation is not necessarily the minimal (shortest) sequence that contains all given oligonucleotides. The minimal sequence is called the *shortest common superstring* of the given oligonucleotides.)

Proposed DNA Algorithm
Step 1. Encode all the strings $\{x_1, x_2, \ldots, x_n\}$ of the set R in DNA strands.
Step 2. Generate all the possible DNA strands w of length less than or equal to K.
Step 3. Let x_1 be a string of R. From the string population of candidates generated in *Step 2* select only those strands that contain x_1 as a subsequence. From the newly obtained string population, select only those strings that contain $x_2 \in R$ as a subsequence, etc. Repeat the step for each strand x_i in R, $1 \leq i \leq n$.
Step 4. If, after *Step 3*, there is any strand w remaining (which means that w contains all $x_i \in R$, $1 \leq i \leq n$, as subsequences), say YES, otherwise say NO.

I omit here the details of our experimental project, mentioning only that initial studies will be done using two chosen candidate oligonucleotide sequences, derived by inspection, one of them containing all the given sequences and another lacking at least one sequence. These test sequences will be flanked by marker sequences. These experiments will ensure that the procedure can reliably produce a positive and a negative result.

4.2 Theoretical research

As mentioned in the first section, one of the aspects of the theoretical side of the DNA computing research comprises attempts to find a suitable model and to give it a mathematical foundation. This aspect is exemplified below by the *contextual insertion/deletion systems*, [38], [39], a formal language model of DNA computing.

As a formal language operation, the *contextual insertion* is a generalization of catenation and insertion of strings and languages, [36]: words can be inserted into a string only if certain *contexts* are present. More precisely, given a set of contexts we put 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 their being theoretically interesting, one of the motivations for studying these operations is that they are already implementable in the laboratory. Indeed, by using available reagents and a standard technique called *PCR site-specific oligonucleotide mutagenesis*, [20], one can perform insertions and deletions of nucleotide sequences in given contexts. (A similar operation, substitution, has been proposed in [13] as a bio-operation necessary to simulate a universal Turing machine.)

We have investigated mathematical properties of contextual insertions and deletions (in the sequel we refer to them simply as insertions and deletions);

one of the obtained results is that the actions of every Turing machine can be simulated entirely by using insertion and deletion rules.

In the following, several characterizations of recursively enumerable (RE) languages (the equivalents of the Turing machine model of computation) are briefly presented, using insertion-deletion systems. Such a system generates the elements of a language by inserting and deleting words, according to their contexts (the insertion-deletion rules are triples (u, z, v), with the meaning that z can be inserted or deleted in/from the context (u, v)). Grammars based on insertion rules were already considered in [22] with linguistic motivation. Insertion/deletion operations are also basic in DNA and RNA processing, [14]. 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* (shortly, *insdel*) *system*, [38], 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 given in the form (u, z, v).

For $x, y \in V^*$ we say that x *derives* y and we write $x \Longrightarrow y$ iff 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 $(u, z, v) \in I$ (an insertion step);
2. $x = x_1uzvx_2$, $y = x_1uvx_2$, for some $x_1, x_2 \in V^*$ and $(u, z, v) \in D$ (a deletion step).

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

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

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

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

$$\begin{aligned}
&\max\{|z| \mid (u, z, v) \in I\} = n, \\
&\max\{|u| \mid (u, z, v) \in I \text{ or } (v, z, u) \in I\} = m, \\
&\max\{|z| \mid (u, z, v) \in D\} = p, \\
&\max\{|u| \mid (u, z, v) \in D \text{ or } (v, z, u) \in D\} = q.
\end{aligned}$$

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

We denote by $INS_n^m DEL_p^q$, $n, m, p, q \geq 0$, the family of languages $L(\gamma)$ generated by insdel 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 insdel languages is $INS_\infty^\infty DEL_\infty^\infty$.

The main results obtained regarding insertion and deletion systems are:

Theorem 1 **(([38]))** $RE = INS_3^6 DEL_2^7$.

Theorem 2 **(([39]))** $RE = INS_1^2 DEL_1^1$.

Theorem 3 **(([39]))** $RE = INS_2^1 DEL_2^0$.

Theorem 4 **(([39]))** $RE = INS_1^2 DEL_2^0$.

The interpretation of, say, 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 both the right and the left context of insertion is at most 6, while the length of deleted strings is at most 2 and the length of the right and left contexts of deletion is bounded by 7.

The proof that contextual insertions and deletions are enough to simulate the actions of a Turing machine opens thus another possible way for designing a molecular computer that uses readily available reagents and techniques.

5 Quo Vadis, DNA Computing?

"Predicting is difficult, especially of the future." said someone famous whose name I can't recall. Indeed, the history of innovations abounds in erroneous predictions about their future impact. As an example of overoptimism, when helicopters were first invented it was predicted that, in the same way we had personal automobiles, we would soon have personal helicopters in our backyard. At the opposite pole, an instance of overpessimism is the forecast (made by one of the leading persons at IBM) that there will be indeed a big market for electronic computers: maybe four or five for the United States, at least one or two for Great Britain... As these examples indicate, one indeed never knows what future will bring and what sudden advance in technology will propel a certain invention at the expense of its competitors.

In the same way, it is difficult to predict what vistas DNA computing research will take in the future. The existing research projects seem to point to a few short term goals and long term goals, summarized below. By its very nature, such a list is both incomplete and subjective, and should be taken with a grain of salt.

Some of the **short term goals** of the DNA computing research are:

– Test the bio-operations used in Adleman's experiment (annealing, polymerase activity), for usability under other conditions and for other computational problems.

– Test other molecular procedures potentially useful for DNA computing (ligation, restriction enzyme digestion, and nuclease digestion on immobilized substrates) for applicability, implementability, scalability, error robusteness, cost and possibility of automatization.

– Search for other types of problems that can potentially be solved by DNA computing. Preference will be given to problems that have practical applications, for example scheduling problems, DNA memories, robotics or weather prediction.

Some of the **long(er) term goals** of DNA computing research are:

– Evaluate existing models of DNA computing from the point of view of their in-vitro implementability.

– Design a viable DNA computability model based on operations specific to DNA processing in vitro: recombination (splicing), matching, insertion, deletion, etc. Investigate the feasibility of the designed model: scalability, fault tolerance and cost.

– Study the computational aspects of the designed model: In particular, compare the model with existing models of computability (especially Turing machines and Chomsky grammars): power, descriptional complexity, dynamic complexity.

– Define and study computational complexity, biological complexity and implementability measures for DNA computing.

– Search for *primitive bio-operations* necessary for describing a *molecular high-level language*. A population of DNA sequences is a *set of strings*, therefore these operations should include basic set operations, string operations and some control operations. The operations should be easily implementable and easy to automatize, and it should be possible to combine them in order to form instructions for solving a large class of problems.

– Optimize "molecular procedures" that accomplish the bio-operations above. For example, it is conceivable that one would need basic set operations, like *generate all the subsets with k elements of a given set* and language theoretic operations like *catenate two strings* or *generate all the strings of a given length over a given alphabet*.

– Automate, as far as practical, the procedures above.

– Develop new problem-solving tools and programming techniques suited for computation with bio-molecules.

– Determine classes of applications that *(a)* can be practically solved by DNA computing, *(b)* have robust DNA-algorithms from the point of view of error-detection and error-correction, and *(c)* have DNA solutions that show substantial advantages (speed, information storage, energy efficiency) compared with the electronic computer based solutions.

– Design DNA based modules for solving these classes of problems.

– Automate and construct DNA-based devices to solve the problems.

– Investigate the feasibility and advantages of a DNA computer.

– Design a DNA-based computer.

I venture to anticipate that the pioneer research in DNA computing, which has as its ultimate goal the design of a DNA computer, will have great significance in many aspects of science and technology. Indeed DNA computing sheds new light onto the very nature of computation, and opens vistas for computability models totally different from the classical ones. In an optimistic way, one may think of an analogy between the work of researchers in this area and the work on finding models of computation carried out in the 30's, which has laid the foundation for the design of the electronic computers.

Acknowledgements. I want to thank Tom Head, Satoshi Kobayashi and Takashi Yokomori for their helpful suggestions and comments, and Michelle Hoyle for clarifying discussions on the history of computing.

References

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

[2] L.Adleman. On constructing a molecular computer. *1st DIMACS workshop on DNA based computers*, Princeton, 1995. In *DIMACS series*, vol.27 (1996), 1–21.

[3] L.Adleman, P.Rothemund, S.Roweis, E.Winfree. On applying molecular computation to the Data Encryption Standard. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 28–48.

[4] J.Amenyo. Mesoscopic computer engineering: automating DNA-based molecular computing via traditional practices of parallel computer architecture design. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 217–235.

[5] M.Amos, A.Gibbons, D.Hodgson. Error-resistant implementation of DNA computation. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 87–101.

[6] M.Arita, M.Hagiya, A.Suyama. Joining and rotating data with molecules. Proceedings of *1997 IEEE International Conference on Evolutionary Computation*, Indianapolis, 243–248.

[7] M.Arita, A.Suyama, M.Hagiya. A heuristic approach for Hamiltonian Path Problem with molecules. *Genetic Programming 1997: Proceedings of the Second Annual Conference (GP-97)*, Morgan Kaufmann Publishers, in press.

[8] E.Baum. Building an associative memory vastly larger than the brain. *Science*, vol.268, April 1995, 583–585.

[9] E.Baum. DNA sequences useful for computation. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 122–127.

[10] E.Baum. D.Boneh. Running dynamic programming algorithms on a DNA computer. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 141–147.

[11] D.Beaver. The complexity of molecular computation. *http://www.transarc.com/~beaver/research/alternative/molecute/molec.html.*

[12] D. Beaver. Computing with DNA. *Journal of Computational Biology*, (2:1), Spring 1995.

[13] D.Beaver. A universal molecular computer. *1st DIMACS workshop on DNA based computers*, Princeton, 1995. In *DIMACS series*, vol.27 (1996), 29–36.

[14] R. Benne (ed.), *RNA-Editing: The Alteration of Protein Coding Sequences of RNA*, Ellis Horwood, 1993.

[15] D.Boneh, C.Dunworth, R.Lipton. Breaking DES using a molecular computer. *1st DIMACS workshop on DNA based computers*, Princeton, 1995. In *DIMACS series*, vol.27 (1996), 37–65.

[16] D.Boneh, R.Lipton, C.Dunworth, J.Sgall. On the computational power of DNA. *http://www.cs.princeton.edu/~dabo.*

[17] D.Boneh, C.Dunworth, J.Sgall, R.Lipton. Making DNA computers error resistant. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 102–110.

[18] E.Csuhaj–Varju, R.Freund, L.Kari, G.Paun. DNA computing based on splicing: universality results. *First Annual Pacific Symposium on Biocomputing*, Hawaii, 1996, also *http://www.csd.uwo.ca/~lila.*

[19] R.Deaton, R.Murphy, J.Rose, M.Garzon, D.Franceschetti, S.Stevens. A DNA based implementation of an evolutionary search for good encodings for DNA computation. Proceedings of *1997 IEEE International Conference on Evolutionary Computation*, Indianapolis, 267–271.

[20] C.W.Dieffenbach, G.S.Dveksler, Eds. *PCR primer: a laboratory manual*, Cold Spring Harbor Laboratory Press, 1995, 581-621.

[21] R.Freund, L.Kari, G.Paun. DNA computing based on splicing: the existence of universal computers. Submitted. Also at *http://www.csd.uwo.ca/~lila.*

[22] B.S.Galiukschov. Semicontextual grammars (in Russian). *Mat.logica i mat. ling.*, Kalinin Univ., 1981, 38-50.

[23] M.Garey, D.Johnson. *Computers and Intractability. A Guide to the Theory of NP–completeness.* W.H.Freeman and Company, San Francisco, 1979.

[24] D.K.Gifford. On the path to computation with DNA. *Science* 266(Nov.1994), 993–994.

[25] G.Gloor, L.Kari, S.Yu. A DNA solution to the Shortest Common Superstring Problem. Manuscript.

[26] F.Guarnieri, M.Fliss, C.Bancroft. Making DNA add. *Science*, vol.273, July 1996, 220-223.

[27] M.Hagiya, M.Arita. Towards parallel evaluation and learning of Boolean μ-formulas with molecules. Submitted.

[28] J.Hartmanis. On the weight of computations. *Bulletin of the European Association of Theoretical Computer Science*, 55(1995), 136–138.

[29] T.Head. Formal language theory and DNA: an analysis of the generative capacity of recombinant behaviors. *Bulletin of Mathematical Biology*, 49(1987) 737–759.

[30] T.Head, G.Paun, D.Pixton. Language theory and genetics. Generative mechanisms suggested by DNA recombination. In *Handbook of Formal Languages* (G.Rozenberg, A.Salomaa eds.), Springer Verlag, 1996.

[31] M.Hoyle. Computers: From the Past to the Present. *http://www.interpac.net/~eingang/Lecture/*, 1997.

[32] N.Jonoska, S.Karl. A molecular computation of the road coloring problem. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 148-158.

[33] N.Jonoska, S.Karl. Ligation experiments in computing with DNA. Proceedings of *1997 IEEE International Conference on Evolutionary Computation*, Indianapolis, 261–266.

[34] P.Kaplan, G.Cecchi, A.Libchaber. Molecular computation: Adleman's experiment repeated. Technical report, NEC Research Institute, 1995.

[35] P.Kaplan, G.Cecchi, A.Libchaber. DNA-based molecular computation: template–template interactions in PCR. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 159–171.

[36] L.Kari. On insertions and deletions in formal languages. *Ph.D. thesis*, University of Turku, Finland, 1991.

[37] L.Kari. DNA computing: arrival of biological mathematics. *The Mathematical Intelligencer*, vol.19, nr.2, Spring 1997, 9–22.

[38] L.Kari, G.Thierrin. Contextual insertions/deletions and computability. *Information and Computation*, 131, no.1 (1996), 47–61.

[39] L.Kari, G.Paun, G.Thierrin, S.Yu. At the crossroads of DNA computing and formal languages: characterizing recursively enumerable languages using insertion/deletion systems. Submitted.

[40] J.Kendrew et al., eds. *The Encyclopedia of Molecular Biology*, Blackwell Science, Oxford, 1994.

[41] S.Kurtz, S.Mahaney, J.Royer, J.Simon. Active transport in biological computing. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 111–121.

[42] S.Kobayashi, T.Yokomori, G.Sampei, K.Mizobuchi. DNA implementation of simple Horn clause computation. Proceedings of *1997 IEEE International Conference on Evolutionary Computation*, Indianapolis, 213-217.

[43] T.Leete, M.Schwartz, R.Williams, D.Wood, J.Salem, H.Rubin. Massively parallel DNA computation: expansion of symbolic determinants. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 49-66.

[44] R.Lipton. DNA solution of hard computational problems. *Science*, vol.268, April 1995, 542–545.

[45] R.Lipton. Speeding up computations via molecular biology. *1st DIMACS workshop on DNA based computers*, Princeton, 1995. In *DIMACS series*, vol.27 (1996), 67–74.

[46] Q.Liu, Z.Guo, A.Condon, R.Corn, M.Lagally, L.Smith. A surface-based approach to DNA computation. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 206–216.

[47] V.Mihalache. Prolog approach to DNA computing. Proceedings of *1997 IEEE International Conference on Evolutionary Computation*, Indianapolis, 249–254.

[48] K.Mir. A restricted genetic alphabet for DNA computing. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 128–130.

[49] J.Oliver. Computation with DNA: matrix multiplication. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 236–248.

[50] G.Paun. On the power of the splicing operation. *International Journal of Computer Mathematics*, 59(1995), 27–35.

[51] G.Paun, A.Salomaa. DNA computing based on the splicing operation. *Mathematica Japonica*, vol.43, no.3, 1996, 607–632.

[52] J.Reif. Parallel molecular computation: models and simulations. *Proceedings: 7th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'95)* Santa Barbara, CA, July 1995, pp. 213-223.

[53] P.Rothemund. A DNA and restriction enzyme implementation of Turing machines. *1st DIMACS workshop on DNA based computers*, Princeton, 1995. In *DIMACS series*, vol.27 (1996), 75–119.

[54] S. Roweis, E. Winfree, R. Burgoyne, N. Chelyapov, M. Goodman, P. Rothemund, L. Adleman. A sticker based architecture for DNA computation. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 1–27.

[55] A.Salomaa. *Formal Languages.* Academic Press, New York, 1973.

[56] N.Seeman et al. The perils of polynucleotides: the experimental gap between the design and assembly of unusual DNA structures. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 191–205.

[57] W.Smith. DNA computers in vitro and in vivo, *1st DIMACS workshop on DNA based computers*, Princeton, 1995. In *DIMACS series*, vol.27 (1996), 121–185.

[58] A.M.Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc.London Math Soc.*, Ser.2, 42(1936), 230–265.

[59] T.Yokomori, S.Kobayashi. DNA-EC: a model of DNA computing based on equality checking. Submitted.

[60] T.Yokomori, S.Kobayashi, C.Ferretti. On the power of circular splicing systems and DNA computability. Proceedings of *1997 IEEE International Conference on Evolutionary Computation*, Indianapolis, 219–224.

[61] R.Williams, D.Wood. Exascale computer algebra problems interconnect with molecular reactions and complexity theory. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 260–268.

[62] E.Winfree. Complexity of restricted and unrestricted models of molecular computation. *1st DIMACS workshop on DNA based computers*, Princeton, 1995. In *DIMACS series*, vol.27 (1996), 187–198.

[63] E.Winfree. On the computational power of DNA annealing and ligation. *1st DIMACS workshop on DNA based computers*, Princeton, 1995. In *DIMACS series*, vol.27 (1996), 199-221.

[64] E.Winfree, X.Yang, N.Seeman. Universal computation via self-assembly of DNA: some theory and experiments. *2nd DIMACS workshop on DNA based computers*, Princeton, 1996, 172–190.