DNA Computing*– molecular computation

H.J. Hoogeboom, Leiden University

Introduction. Within Computer Science the field of *Natural Computation* studies computational techniques inspired by natural phenomena. A well-known example of such a technique is evolutionary computation (genetic algorithms) where properties of objects are coded using a sequence of bits (their 'genetic information'), and the most suitable object emerges from a pool of candidates by a process similar to natural selection. Note that evolutionary computation is usually performed on classical 'silicon' computers. With *DNA computing* one intends to turn the table by proposing strands of DNA as hardware, and not only as a programming paradigm. These strands are to be manipulated by biological and biotechnological methods. On the one hand nature has provided us with a large box of enzymes designed to perform specific operations on DNA, on the other hand biotechnological research leads to new and powerful lab techniques useful for genetic manipulations.

The potential of molecular computation is quite impressive. A simple test tube of DNA may easily contain 10^{15} strands of DNA, all of which can be operated in a massively parallel fashion, leading to "the ultimate computer: a trillion calculations in parallel" [8]. The scale of DNA is so small that it provides a very dense storage medium which, under favourable conditions, can be kept over a long time. Finally, biotechnological research is constantly improving on the tools that are available for synthesising, manipulating and analysing DNA.

In 1994 Leonard Adleman reported his initial and exciting experiments in Science [1]. He solved a very simple instance of the Hamiltonian path problem using DNA, providing the first laboratory sample of molecular computation.

The biochemical tool box. A single strand of DNA (deoxyribonucleic acid) consists of a sequence of nucleotides linked together by a strong backbone. These nucleotides may differ by their attached 'bases' only, four of

*Presentation for the IPA Introductiedagen, session 'trends in programming', September 1996, Noordwijk

which occur in ordinary DNA: Adenine, Thymine, Guanine, and Cytosine. Two single strands of DNA can zip together into a stable double stranded molecule, provided the base pairs in the two strands are complementary: Adenine only matches to Thymine, Guanine to Cytosine. This phenomenon is called Watson-Crick complementarity, after its discoverers. For our purposes DNA can be conveniently represented by (double) strings over the base alphabet {A,T,C,G}.

There are several basic techniques that are used to compute with DNA:

denaturing double stranded DNA; splitting it into single strands, for instance by increasing the temperature.

annealing 'sticky ends'; pieces of double stranded DNA with single stranded overhang can be recombined into a double strand if the extending pieces match through complementarity; an enzyme called *ligase* repairs the phosphor-ribose backbones.

```
TATA          GCACT       TATAGCACT
ATATCGT   +     GA    →   ATATCGTGA
```

separation of strands of DNA by their length; usually exploiting the fact that the travelling speed through a gel is dependent on the size of the molecule.

selecting single stranded sequences that match specific patterns; by binding them to a surface that carries the complement of the pattern.

multiplying a specific (double) strand of DNA, 'amplification'; perhaps most efficient amplification technique is by a process called PCR, *polymerase chain reaction*. In this reaction double strands are repeatedly split into single strands which are grown into full double strands by an enzyme called polymerase (with a sufficient supply of nucleotides).

Adleman's experiment. Adleman proposed the following 'abstract' nondeterministic algorithm for solving the directed Hamiltonian path problem (find a path with given initial and final nodes that passes through each vertex in the graph exactly once) on a graph with n nodes. He then implemented this algorithm on the new hardware: DNA.

1. generate random paths through the graph
2. keep only paths from the initial to the final node
3. keep only paths that enter exactly n vertices

4. keep only paths that enter all of the vertices (at least once)

5. if any paths remain, the graph contains a Hamiltonian path

The input graph is coded into DNA as follows. Start by synthesising different base sequences for each of the vertices in the graph. The edges in the graph are represented by base sequences that will anneal to each of their adjacent vertices. When these short sequences are mixed together in large quantities arbitrary large double stranded molecules will form, representing paths in the graph.

Consider for example the vertex sequences `GTAGACCT` and `GCGTTCAC`, that can be joined by the edge sequence `TGGACGCA`, to form
```
GTAGACCTGCGTTCAC
  TGGACGCA
```

Paths from the initial to the final vertex are selected by special tuned forms of PCR, amplifying only segments that start and end with specific patterns. As we know the number of bases used to code each vertex, the paths that enter precisely n vertices can be recovered by selection on molecular length.

Selecting on the base sequences for each of the vertices successively, we keep only those paths that enter each of the vertices (exactly) once. As very small amounts of DNA can't be detected, the remaining residue is amplified by PCR to see whether any path has survived the selection.

Comments. Adleman is well aware that his experiment was only an initial step towards real applications. We summarise his own comments on the feasibility of his technique.

He states that his methods could be scaled up to accommodate larger graphs. Critics have computed that this inevitably will mean bath-tubs full of DNA to code problems of a more realistic size. Adleman argues that his algorithm may be a poor implementation, and that suitable codings must be searched for. Not only space, but also time requirements are problematic. The original experiment took approximately seven days of lab work. Automation and alternative molecular algorithms may help to overcome this slow performance.

There are several details of the implementation that may result in possible errors. The main biochemical techniques used, PCR and separation procedures, are extremely error prone. Moreover, DNA may form non existing pseudo-paths by accidental ligation between similarly coded nodes. Additionally, DNA may form into hairpin loops.

A large part of present research is dealing with these sources of errors. New, refined implementations have been proposed to minimise faulty results

[2].

A comment on the power of the method is in order here. Adleman has computed that there are about 10^{14} strands of DNA in his test tubes, and he believes that 10^{20} is a plausible number to deal with. Even if we take into account the large reaction times, this number of parallel operations outperforms the present supercomputers by several thousand-folds.

The potential of the method lies in massively parallel searches. It is however not clear at this moment whether the techniques can be used to solve real computational problems, like the multiplication of 100-digit numbers.

Further Models. Adleman's seminal experiment initiated a lot of new research, answering Adleman's question for alternative models and better tuned DNA implementations of algorithms. We will have now a short look at some of these (which were selected according to personal taste).

Lipton [5] attacked another problem known to be NP-complete, the satisfiability problem. He proposed a method to code truth assignments as strands of DNA, and to select one strand by manipulating test tubes of DNA. The type of problem is, like Adleman's, again a combinatorial search. The novelty of Lipton's approach is that the problem defines the order in which several tubes are manipulated, *not* the initial coding (as in Adleman's approach).

Lipton also stressed that the operations on DNA should be classified, in order to facilitate a more 'abstract' way of modelling the implementations.

In his original paper, Adleman posed the question whether DNA computations could be 'programmed', as a further step towards DNA computers. Recall that his own experiment was special purpose. Several implementations of a Turing machine were proposed, but perhaps the most detailed description is given by Rothemund [9], who lists all the enzymes needed to perform the specific operations on his DNA Turing tapes. His intricate model does not indicate the power of molecular computation. Making no use of the inherent parallellism of DNA, a single step of his Turing machine is estimated to take four hours, slower than Turing's own pencil and paper implementation I guess.

Note that the question whether molecular computation is Turing complete is also answered by work on the *splicing system* model, proposed by Head [3], as early as 1987 (!) see for instance [4], or the work of Păun [7].

The model of Roweis et al. [10] views a single strand of DNA as addressable memory, initially set to zero. A particular memory address can be set to one, by synthesising the strand of DNA complementary to the specific address, and annealing this to the original strand. Strands with specific bits

set can be selected by the usual selection on DNA patterns. A robotic machinery to handle tubes of memory strands is proposed. They conclude that some challenging problems (breaking the Data Encryption Standard) need only modest volumes of DNA.

The initial contents of Adleman's test tubes are self-assembling: after synthesising only a small number of relatively short segments of DNA (in large amounts) the paths form themselves. Winfree et al. [11] investigate how this principle can be used to self-assemble computations of a programmable computer. (The models for simulating Turing machines that have been proposed all involve several laboratory steps for one cycle of the Turing machine computation.) One can argue that in Adleman's case the self-assembly is of a regular (finite automaton) type. Winfree shows that, using two-dimensional properties of DNA, this can be improved to self-assembly of context-free type: DNA fragments literally grow into derivation trees of context-free grammars. A more speculative proposal is to assemble intricately twisted pieces of DNA into computations of cellular automata (having Turing power).

Concerns. During the second Workshop on DNA Based Computers [2] people active in the 'conceptual' field of DNA computing met with people familiar with biotechnological manipulation with DNA. They expressed great optimism towards the final goal, the construction of a desk-top DNA computer, and pictures of robots manipulating large racks of tubes were sketched. Of course some realistic concerns were expressed related to the development of such hardware.

We summarize the main points that came up in the discussions.

- Almost no lab work has been done on the computational models, so far most of them have been conceptual models only. It is a common feeling that these concepts need some initial test for feasibility.

- Real-life *DNA properties* must be taken into account. Although DNA is a very versatile and stable molecular structure, it is not the linear one-dimensional string we like to think it is. It may engage into non-complementary binding and form into hairpin loops.

- We need a more robust *complexity analysis*, involving reaction rates. Just claiming that "Adleman efficiently solved an NP-complete problem" by attacking a specific seven city instance does not increase a possible acceptance of DNA computing by other computer scientists.

- Ordinary parallel computers allow at least basic *communication* between processors. In the DNA model no realistic ways of achieving this have been proposed yet.

- A common current intuition says that molecular computation will not outperform silicon computers on every type of problem. So far, DNA models have proved to be strong in massively parallel searches, but even a two bit adder turns out to be a major technological problem. The field definitely needs a '*killer application*' to prove its strength.

Conclusion. The consensus of the second Workshop on DNA Based Computers was that one should try to build a DNA computer now, by attempting to use the tremendous resources available for biochemical research (as someone said: "if you need something, just ask the chemists"). The final question is of course how far we are from the desktop DNA computer. On the one hand, we have to be patient: one was using the vacuum tube first, before moving to the transistor. On the other hand, progress in biotechnology (driven by commercial motivations other than DNA computing) has been tremendous, surely contributing to great expectations for the feasibility of DNA computing.

Thank you. Papers in the field were studied in an informal Leiden seminar, together with Grzegorz Rozenberg, Nikè van Vugt, and Ray Dassen. Ray maintains an extensive annotated bibliography on molecular computing, available at `http://www.wi.leidenuniv.nl/home/jdassen/dna.html`.

References

[1] L. Adleman. Molecular Computation of Solutions to Combinatorial Problems. Science 266 (1994) 1021–1024.

[2] DNA Based Computers, Second Annual Meeting, preliminary proceedings, June 10–12, 1996. DIMACS Workshop, Princeton University, Princeton, NJ, USA.

[3] T. Head. Formal Language Theory and DNA: an analysis of the generative capacity of recombinant behaviors. Bulletin of Mathematical Biology 49 (1987) 737–759.

[4] T. Head, G. Păun, D. Pixton. Language Theory and Molecular Genetics – generative mechanisms suggested by DNA recombination. In: G. Rozenberg, A. Salomaa (eds.) Handbook of Formal Language Theory, vol. 2, Springer-verlag, 1997.

[5] R.J. Lipton. DNA Solution of Hard Computational Problems. Science 268 (1995) 542–545.

[6] R.J. Lipton, E.B. Baum (eds.) DNA Based Computers: Proceedings of a DIMACS Workshop. American Mathematical Society, DIMACS Series in Discr. Math. and Theor. Comp. Sci., Vol. 27, 1996.

[7] G. Păun. On the Power of the Splicing Operation. Int. Journal of Computer Mathematics 59 (1995) 27–35.

[8] R. Pool. The Ultimate Computer: A trillion calculations in parallel. New Scientist, 13 July 1996, pp. 26–31.

[9] P.W.K. Rothemund. A DNA and Restriction Enzyme Implementation of Turing Machines. In: DNA Based Computers 1996, [6], pp. 75–119.

[10] S. Roweis, E. Winfree, R. Burgoyne, N.V. Chelyapov, M.F. Goodman, P.W.K. Rothemund, and L.M. Adleman. A Sticker Based Model for DNA Computation. In: DNA Based Computers, Second Annual Meeting 1996, [2], pp. 1–27.

[11] E. Winfree, X. Yang, N.C. Seeman. Universal Computation via Self-assembly of DNA: some theory and experiments. In: DNA Based Computers, Second Annual Meeting 1996, [2], pp. 172–190.