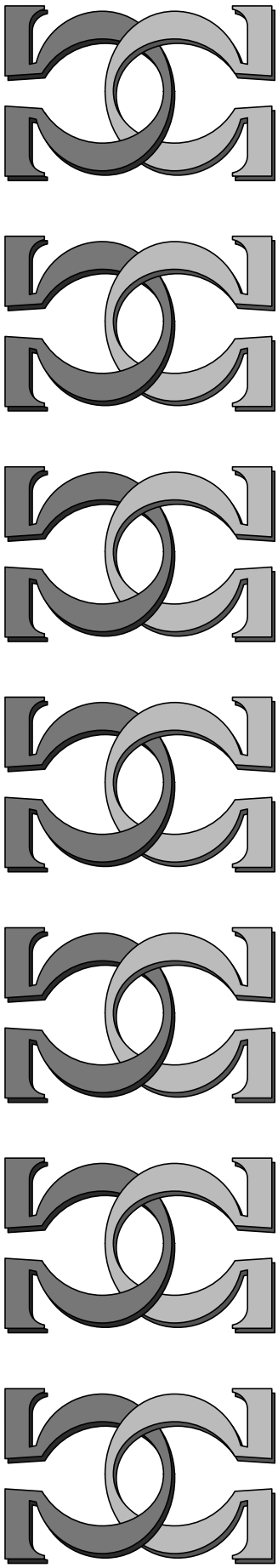

CDMTCS
Research
Report
Series

A Glimpse into Natural Computing

Cristian S. Calude
Auckland University, New Zealand

Gheorghe Păun
Institute of Mathematics, Romanian Academy

Monica Tătărâm
Bucharest University, Romania

CDMTCS-117
December 1999

Centre for Discrete Mathematics and
Theoretical Computer Science

A Glimpse into Natural Computing

Cristian S. Calude[(a)], **Gheorghe Păun**[(b)], **Monica Tătărâm**[(c)]

[(a)]Department of Computer Science
The University of Auckland
Private Bag 92019, Auckland, New Zealand
E-mail: cristian@cs.auckland.ac.nz

[(b)]Institute of Mathematics of the Romanian Academy
PO Box 1 – 764, R-70700 Bucureşti, Romania
E-mail: gpaun@imar.ro

[(c)]Faculty of Mathematics, University of Bucharest
Str. Academiei 14, R-70109 Bucureşti, Romania
E-mail: tataram@math.unibuc.ro

Abstract. We consider as pertaining to Natural Computing (in some sense, characterizing it) the following five domains: Neural Networks, Genetic Algorithms, DNA Computing, Membrane Computing, and Quantum Computing. The first two domains are well established, the last three are just now looking for a place in the toolkit of practitioners. Here, we briefly introduce the last three domains to the reader. The main point is that in all these areas one aims at solving intractable (NP-complete) problems in polynomial (in many cases, even linear) time. Taking into account that most significant practical problems (optimization, scheduling, programming, combinatorial, etc.) are intractable, it follows that the promises of Natural Computing should be taken seriously.

1 Natural Computing = Five Revolutionary Domains

A somewhat traditional view interprets the syntagm "natural computing" in the sense of "the ways the *alive* nature computes". The premise is that life computes[1] since billions of years with rather good (at the qualitative level) results.

The main such area is that of processes which take place at the genetic level and which is responsible of the incredible diversity of life. At this level, "good results" refer to the continuously improved offsprings[2], to the stability and efficiency of "genetic computations". The suggestion is immediate: try to imitate the life, in the hope that procedures tested for billions of years should work well also in "non-natural" circumstances.

Actually, Natural Computing included traditionally two domains: Neural Networks and Genetic Algorithms. The first one tries to make use of (and to imitate, in the extent this is possible) the way the human brain computes. Actually, one stops at the level of small depth networks of neuron-like elementary processors; the power of the machinery lies in the parallelism and cooperation of components. The second domain makes use of genetic transformations (mainly crossing-over and point mutations) in search of good solutions of optimization problems, in the hope that by combining good solutions we also get good solutions, possibly better.

From a theoretical/mathematical point of view, it is somewhat surprising that both Neural Networks and Genetic Algorithms are very successful in very many circumstances. Engineers are enthusiastic about both these domains. Design problems which cannot be solved analytically and which are intractable as optimization problems can be quite satisfactorily solved by genetic algorithms (the optimality is not guaranteed, but this is not important as long as the solution at hand is good enough in comparison to existing solutions).

The reader is referred to [26] and [33] for more information; we particularly recommend the latter reference, for a huge list of practical problems which were at least approached – if not satisfactorily solved – by means of Genetic Algorithms.

Because the previous two areas are already well established and many good bibliographical sources are available for them, we do not enter here in any further details.

Since a few years one new branch of Natural Computing (inspired from the genetic area) has emerged: DNA Computing. This time, the ambition is much higher. DNA handling is not only the source of ideas for a new computing paradigm, but also the media of implementing the computation. One aims not only to a new class of computing models, but to a new class of computers, called in various places in various ways: bio-computers, wet computers, computers based on bioware, protein computers, etc. The idea is not new, speculations about the possibility of using DNA and specific operations as a support for computations were made since fifties and sixties (Ch. Bennett, R. Feynman, M. Conrad). An important theoretical step was made by T. Head, in 1987, when he has introduced a theoretical model of the recombination operation of DNA molecules, [27], but the domain birth certificate is considered Adleman's experiment reported in [1]: a small instance of the Hamiltonian Path Problem in a graph was solved by purely biochemical means. The

[1] Actually, we do not accept the fact that nature computes. We, humans as rational beings, see computations in the many places where the nature just evolves. Accepting that nature computes is like accepting that the bees have any geometrical representation of the fact the cells of their nests are hexagonal. Still, we speak here, metaphorically, about ways/tools/frameworks used by nature for computing.

[2] At least, this is what we believe about evolution...

problem is known to be NP-complete, hence intractable, while Adleman's algorithm was linear (as the number of biochemical operations) in the number of nodes of the graph. We shall return in the next section to this topic.

Let us now state an easy, but fundamental, question, which has led to one more area of Natural Computing: how does life compute? This can be reformulated as: how do people believe that life computes? Answers referring to the (human) brain and the genetic area were already mentioned above, but there is one more area where it was several times asserted that we meet computations – the cellular level. In many places one claims that the processes which take places in a cell, the reactions which develop in cell regions, the processing of substances, energy and information in these regions and through the membranes which delimit them, is a computation process. See, e.g., [8] and its references.

Taking this assertion seriously leads to a genuinely new computing area: Computing with Membranes. The domain is about one year old – the first paper was [36] – but its bibliography counts already over 20 papers. In short, a distributed device is considered, based on a membrane structure in whose regions certain objects evolve according to given evolution rules, in parallel; many variants are possible, some of them being able to solve in linear (parallel, biochemical) time NP-complete problems. Section 3 is devoted to this subject.

However, we want to stress the similarities and the differences between the four domains already mentioned. Figure 1 is illustrative in this respect.

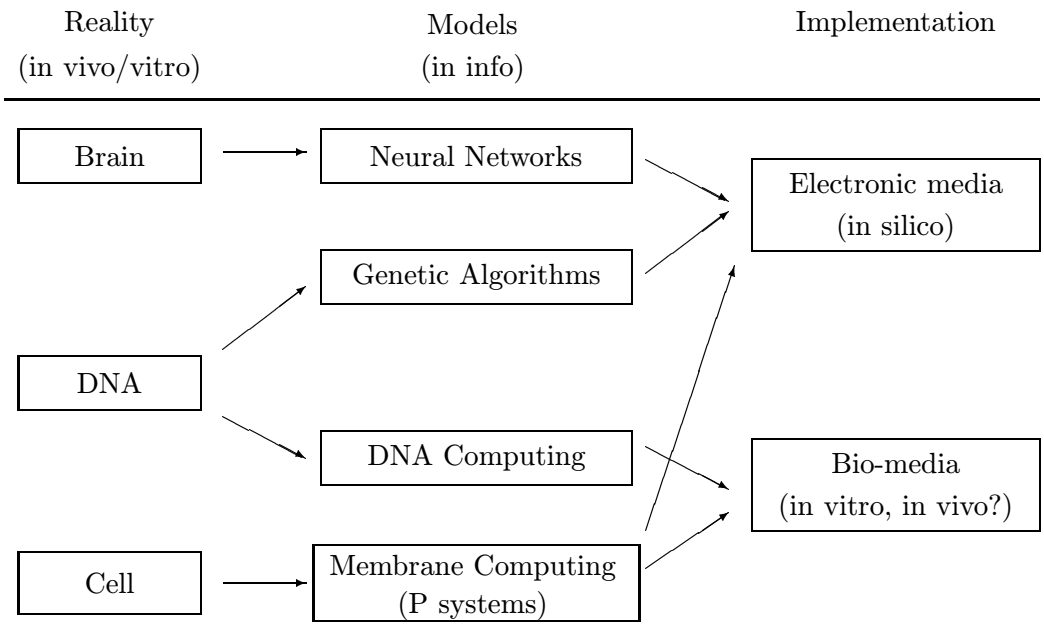

Figure 1: Four domains of Naural Computing

All domains are inspired from biology-biochemistry. While Neural Networks and Genetic Algorithms (Evolutionary Computation) are already domains in current use, DNA Computing and Membrane Computing are still at the level of attempts and trials. However, while the first two domains are satisfied with good solutions, the last two aim

at computing in parallel optimal solutions. Crucial is the difference in what concerns the implementation: Neural Networks and Genetic Algorithms are implemented on usual computers, they are just new classes of computation models; DNA Computing explicitly starts from the goal of making use of the huge parallelism made possible by DNA and aims at computing in a test tube. Several experiments were done – all of them dealing with toy problems. The situation of Membrane Computing is not clear. No lab experiment was carried out up to now, while the attempts to simulate P systems on the electronic computer (see, for instance, [42]) are not yet convincing from a practical point of view. However, the domains are so young and progresses so fast that any forecast is premature (and, definitely, no negative prediction is acceptable).

Recently, one further domain is considered as belonging to Natural Computing, although it does not deal with biology but with "hard science": Quantum Computing. For instance, a column has been started in *Bulletin of European Association for Theoretical Computer Science* with the title "Natural Computing" and aiming to cover all the five areas: Neural Networks, Genetic Algorithms, DNA Computing, Membrane Computing, Quantum Computing. A series of monographs will be soon published by Springer-Verlag, Berlin, with the same title and the same scope. We adhere to this interpretation of Natural Computing, that is, we include Quantum Computing in this field.

Again, the ambitions and promises are very high: parallel computations based, among other, on the quantum superposition phenomenon, which makes possible linear solutions of exponential problems, on a hardware of a completely new type, based on quantum physics. Section 4 will be devoted to this topic.

We conclude this introductory section by stressing the fact that DNA Computing, Membrane Computing, and Quantum Computing search and promise linear solutions for exponential problems. Taking into account the success of Neural Networks and Genetic Algorithms, we should be optimistic in this respect.

2 DNA Computing

As we have said before, speculations about the possibility of using DNA molecules as a support for computations were made since several decades. The DNA features which encourage these ideas are multiple: DNA has a well-known structure, of a clear syntactic type; it can be handled in many ways already tested by nature and known by genetic engineers; DNA is one of the most efficient data storage, with the possibility of encoding a bit at the level of a molecule; related to this and much more important, DNA makes possible a huge parallelism, in a small test tube one can put billions of molecules; the biochemical reactions are very efficient from an energetic point of view and they are reversible; the biochemical reactions have a high degree of nondeterminism, which, suitably exploited (together with the parallelism), can lead to very efficient computations. Of course, the nondeterminism of biochemical reactions is, at least in this moment, also a bad feature, because the result we get is reliable only with a certain probability[3].

When looking for a new computing model inspired from biochemistry (this is the case both for DNA Computing and for Membrane Computing), we look for *data structures* and *operations* with these data structures. In the case of DNA, the main data structure is the double stranded sequence, composed of "symbols" (A, C, G, T, the four nucleotides

[3]Other drawbacks of DNA Computing are the fact that we have to carefully handle the errors and the many exceptions customary in biochemistry, but we do not persist here in the negative direction...

which compose a DNA molecule) which are paired according to a well-specified *complementarity relation* (the Watson-Crick complementarity says that always A is paired with T and C with G). Actually, we can also work with usual strings: by heating a solution which contains DNA molecules, the two strands are separated (one says that DNA is *denaturated*). By cooling the solution, the single strands will again glue together, observing the complementarity of nucleotides and forming double stranded molecules (one says that DNA is *renaturated*; the operation is also called *annealing*).

We have thus already obtained two operations: denaturation and annealing. Much more can be found. A very important one is *recombination* (*crossing-over*), the basic one used in Genetic Algorithms. We consider it in the form formalized by T. Head, in 1987, under the mane of *splicing*, [27]. We do not give it mathematically, but we illustrate it by an example.

Consider the following two DNA molecules

$$5' - \text{CCCCCTCGACCCCC} - 3'$$
$$3' - \text{GGGGGAGCTGGGGG} - 5'$$

$$5' - \text{AAAAAGCGCAAAAA} - 3'$$
$$3' - \text{TTTTTCGCGTTTTT} - 5'$$

as well as the restriction enzymes *Taq*I and *Sci*NI, which recognize, respectively, the following patterns

$$\begin{array}{ll} \text{T}\vert\overline{\text{C G}}\,\text{A} & \text{G}\vert\overline{\text{C G}}\,\text{C} \\ \text{A G C}\vert\text{T} & \text{C G C}\vert\text{G} \end{array}$$

We have also indicated the way of cutting the DNA molecules. Thus, when acting on the two molecules mentioned above, these enzymes will produce the following four fragments:

$$\begin{array}{ll} 5' - \text{CCCCCT} & \text{CGACCCCC} - 3' \\ 3' - \text{GGGGGAGC} & \text{TGGGGG} - 5' \end{array}$$

$$\begin{array}{ll} 5' - \text{AAAAAG} & \text{CGCAAAAA} - 3' \\ 3' - \text{TTTTTCGC} & \text{GTTTTT} - 5' \end{array}$$

We have obtained molecules with identical sticky ends, therefore the four fragments can be bound together, either restoring the initial molecules, or producing new molecules by recombination. The recombination gives the following new molecules

$$5' - \text{CCCCCTCGCAAAAA} - 3'$$
$$3' - \text{GGGGGAGCGTTTTT} - 5'$$

$$5' - \text{AAAAAGCGACCCCC} - 3'$$
$$3' - \text{TTTTTCGCTGGGGG} - 5'$$

This operation (cut at certain places and recombine the fragments which have matching sticky ends) is now used as the basic ingredient of a large class of computing mechanisms, known under the name of *H systems*. Several chapters in the monograph [39]

are devoted to the study of these systems. Although the theory of H systems is much developed and in spite of the fact that H systems of various types are capable to perform universal computations (they can simulate any Turing Machine), we do not enter into details because of a "simple detail": nothing is known in this moment about the *complexity* of computations performed by H systems (whether or not hard problems can be solved in this framework in an efficient way), and we return to the history-making Adleman experiment [1].

It uses as basic operations the denaturation and annealing (plus filtering DNA molecules according to various criteria, amplification with selective primers, gel electrophoresis – for more information we refer to [1], [39], [11]) and solves in linear time an NP-complete problem – whether or not a given directed graph contains any Hamiltonian path starting in a given node and ending in a given node.

Following the terminology of [24], this was a convincing *demo*, which has proved that genetic engineering materials and techniques constitute a possible new framework for computability. Adleman's experiment is important not only because it was the first of this type, but also by the way it was conducted.

The graph considered by Adleman was that in Figure 2. We have seven vertices and fourteen arcs. The question is whether or not there is a path from vertex 0 to vertex 6 which passes exactly once through each of the other vertices.

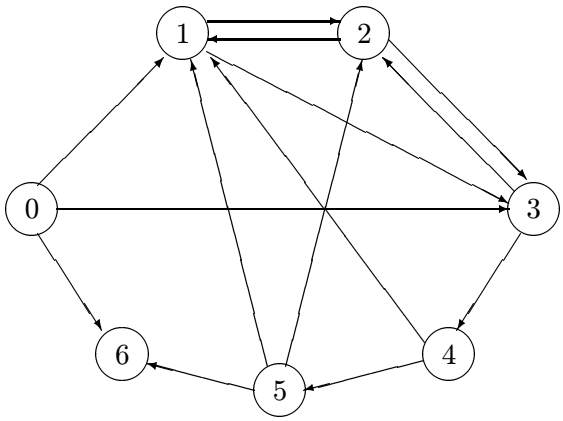

Figure 2: The graph in Adleman's experiment.

By a simple examination of the graph, we see that there is such a path, namely that following the numbering of the vertices: 0, 1, 2, 3, 4, 5, 6. However, we have mentioned that the problem is a hard one, among the hardest which can be solved in polynomial time by non-deterministic algorithms: it is an NP-complete problem. Otherwise stated, all known deterministic algorithms for solving this problem are essentially equally complex as the exhaustive search. However, we can trade time for space, and this is exactly what Adleman has done, making use of the massive parallelism of DNA (in some sense, massive parallelism can simulate non-determinism, and in this way non-deterministic algorithms can be implemented).

The algorithm used by Adleman was the following:

Input: A directed graph G with n vertices, among which there are two designated vertices v_{in} and v_{out}.

Step 1: Generate paths in G randomly in large quantities.

Step 2: Remove all paths that do not begin with v_{in} or do not end in v_{out}.

Step 3: Remove all paths that do not involve exactly n vertices.

Step 4: For each of the n vertices v, remove all paths that do not involve v.

Output: "Yes" if any path remains, "No" otherwise.

It is easy to see that, if we assume that each of the operations in the algorithm takes exactly one time unit, then the solution is obtained in a number of time units which is linear in n, the number of vertices in the graph: steps 1, 2, 3 need a constant number of time units (say, 4: generate all paths, select the paths starting with v_{in}, select the paths ending with v_{out}, select the paths of length n), while step 4 takes n time units (check for each vertex its presence in the currently non-rejected paths).

The main difficulty lies in step 1, where we have to generate a large number of paths in the graph, as large as possible, in order to reach with a high enough probability the Hamiltonian paths, if any. Of course, when the graph also contains cycles, the set of paths is infinite. In a graph without cycles, the set of paths is finite, but it can be of an exponential cardinality with respect to the number of vertices. This is the point where the massive parallelism and the non-determinism of chemical reactions were cleverly used by Adleman in such a way that this step was performed in a time practically independent of the size of the graph.

The biochemical implementation of the above described algorithm was the following.

Each vertex of the graph was encoded by a single stranded sequence of nucleotides, namely of length 20. These codes were constructed at random; the length 20 is enough in order to ensure that the codes are "sufficiently different". A huge number of these oligonucleotides (amplified by PCR) were placed in a test tube. In the same test tube are then added (a huge number of) codes of the graph edges, of the following form: if there is an edge from vertex i to vertex j and the codes of these vertices are $s_i = u_i v_i, s_j = u_j v_j$, where u_i, v_i, u_j, v_j are sequences of length 10, then the edge $i \to j$ is encoded by the Watson-Crick complement of the sequence $v_i u_j$.

For instance, for the codes of vertices 2, 3, 4 specified below

$$s_2 = 5' - \text{TATCGGATCGGTATATCCGA} - 3',$$
$$s_3 = 5' - \text{GCTATTCGAGCTTAAAGCTA} - 3',$$
$$s_4 = 5' - \text{GGCTAGGTACCAGCATGCTT} - 3',$$

the edges $2 \to 3, 3 \to 2$ and $3 \to 4$ were encoded by

$$e_{2\to 3} = 3' - \text{CATATAGGCTCGATAAGCTC} - 5',$$
$$e_{3\to 2} = 3' - \text{GAATTTCGATATAGCCTAGC} - 5',$$
$$e_{3\to 4} = 3' - \text{GAATTTCGATCCGATCCATG} - 5'.$$

By annealing, the codes of the vertices act as splints with respect to codes of edges and longer molecules are obtained, encoding paths in the graph. The reader can easily see

how the molecules specified above will lead to sequences encoding the paths $2 \to 3 \to 4$, $3 \to 2 \to 3 \to 4$, etc.

Adleman has let the process to proceed four hours, in order to be sure that all ligation operations take place. What we obtain is a solution containing a lot of non-Hamiltonian paths (short paths, cycles, paths passing twice through the same vertex). The rest of the procedure consists of checking whether or not at least a molecule exists which encodes a Hamiltonian path which starts in 0 and ends in 6.

There is here a very important point, which will be referred to later. The difficult step of the computation was carried out "automatically" by the DNA molecules, making use of the parallelism and the Watson–Crick complementarity. In this way, we get a large set of *candidate solutions*, (hopefully) both molecules which encode paths which are looked for, but also many molecules which should be filtered out, a sort of "garbage" to be rejected.

This second part of the procedure, of filtering the result of step 1 in order to see whether a solution to our problem exists, was carried out by Adleman in about seven days of laboratory work, by performing the following operations: By a PCR amplification with primers representing the input and the output vertices (0 and 6), only paths starting in 0 and ending in 6 were preserved. After that, by gel electrophoresis there have been extracted the molecules of the proper length: 140, because we have 7 vertices encoded by 20 nucleotides each. Thus, at the end of step 3 we have a set of molecules which encode paths in the graph which start in 0, end in 6, and pass through 7 vertices. (It is worth noting that this does not ensure that such a path is Hamiltonian in our graph: 0, 3, 2, 3, 4, 5, 6 is a path which visits seven vertices, but it passes twice through 3 and never through 1.) Roughly speaking, step 4 is performed by repeating for each vertex i the following operations: melt the result of step 3, add the complement of the code s_i of vertex i and let to anneal; remove all molecules which do not anneal.

If any molecule survives step 4, then it encodes a Hamiltonian path in our graph, namely one which starts in vertex 0 and ends in vertex 6.

The practical details of this procedure are not very important for the present discussion. They depend on the present day laboratory possibilities and can be performed also by other techniques; furthermore, the algorithm itself can be changed, improved or completely replaced by another one. What is important here is the *proof that such a computation is possible*. Purely biochemical means were used in order to solve a hard problem, actually an intractable one, in a linear time as the number of lab operations. These operations, in an abstract formulation, are another main output of this experiment and of the thought about it, leading to a sort of programming language based on test tubes and DNA molecules manipulation.

Such a "test tube programming language" was proposed in [31] (where the well-known SAT Problem, probably the most used NP-complete problem, was solved in linear time by a procedure which extends Adleman's algorithm), developed in [2] and then discussed in many places. We do not describe it here, but we point out one of the main weaknesses of Adleman's procedure, from a practical point of view: the number of necessary single strands, codes of vertices or of edges, is of the order of $n!$, where n is the number of vertices in the graph. This imposes drastic limitations on the size of the problems which can be solved in this manner. (J. Hartmanis [25] has proved that in order to handle in this manner graphs with 200 nodes, graphs with a size of practical importance, easily handled by conventional computers, we need $3 \cdot 10^{25}$ Kg of DNA, which is more than the

weight of the Earth!) In short, Adleman's procedure is elegant, copes in a nice way with the errors, but it cannot be scaled-up.

Several improvements of Adleman's procedure were proposed. The most promising approach is to use an evolutionary computing strategy: instead of constructing a "complete data pool" from which the solutions are then filtered out, only "good" candidates are produced, which saves a lot of the needed DNA. Still, no problem of a practical size was reported to be solved.

The way of proceeding in Adleman's experiment and in most of the DNA Computing experiments reported up to now suggests a general (somewhat unusual) strategy of computing, proposed in [34] under the name of *computing by carving*: generate a set of candidate solutions and then "carve" it, removing sets of non-solutions, iteratively, until a solution is obtained. This mode of "working on the complement" proves to be very powerful (even sets which are Turing non-computable can be "computed" in this way) and it is expected to be also useful from the complexity point of view.

This type of computation reminds both the celebrated Erathostene's sieve and Khachian polynomial algorithm for linear programming. Note, however, that in Erathostene's sieve we do not first "generate" the set of natural numbers, they are given for free; in the case of computing by carving the set of candidate solutions is explicitly built in the computation, in some sense this is the main part of it, the place where the massive parallelism made possible by DNA is used. The comparison with Khachian algorithm (which also proceeds by shrinking the space where the solution is looked for, by cutting the current ellipsoid where lies the solution) points to an important warning: sometimes, the theoretical efficient solutions are not also practically efficient. In most practical cases, the old simplex algorithm, which is an exponential one, proves to be much more efficient than the polynomial ellipsoid algorithm; this is just a consequence of the fact that worst-case complexity is rather different from average complexity (while reality is closer to average, if not still better placed).

3 Membrane Computing – P Systems

One starts from the observation that life "computes" not only at the genetic level, but also at the cellular level. More generally, any non-trivial biological system is a hierarchical construct, composed of several "organs" which are well defined and delimited from the neighboring organs, which evolve internally and also cooperate with the other organs in order to keep alive the system as a whole; an intricate flow of materials, energy, and information underlies the functioning of such a system.

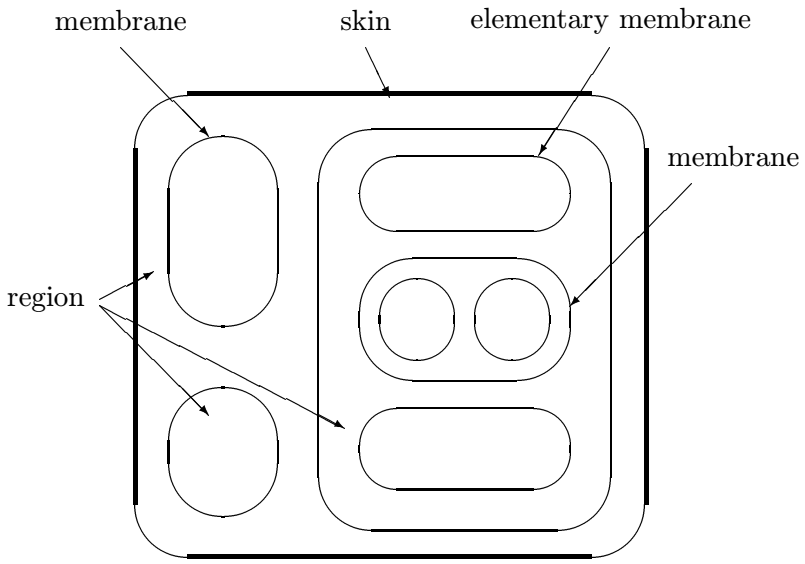

Figure 3: A membrane structure.

At a more specific level with respect to the models we are going to discuss, it is important to note that the parts of a biological system are well delimited by various types of *membranes*, in the broad sense of the term. These membranes can be arranged in a *membrane structure* of a complex shape. Such a structure can be represented in a natural way as a Venn diagram. This makes clear the fact that what matters is the topological structure, the relationships between membranes, not their ordering.

The Venn representation of a membrane structure μ also makes clear the notion of a *region* in μ: any closed space delimited by membranes is called a region of μ. It is clear that a membrane structure of degree n contains n internal regions, one associated with each membrane. We also use to speak about the *outer* region, the whole space outside the skin membrane.

Figure 3 illustrates some of the notions mentioned above.

If in the regions of a membrane structure we place *objects* from a given set, with well specified multiplicities (we work with *multisets* of objects, that is, the multiset is the data structure we use), as well as *evolution rules* for these objects, then we obtain a computing device (called a *P system*): we start from the initial configuration of the system and let the system evolve; at each time unit, all objects which can be the subject of an evolution rule should evolve (that is, the rules are used in a maximally parallel manner). In this way, we pass to another configuration of the system. Such a sequence of transitions among configurations is a *computation*. A computation is *complete* if it halts: no rule can be used in the last configuration. An *output* can be associated with a complete configuration in several ways. For instance, we can consider an elementary membrane as the *output* membrane of the system and the contents of this membrane at the end of the computation is the result of the computation. Another possibility is to "read" the result outside the system, just considering the objects which leave the system during the computation.

Many variants can be considered, by combining various ingredients used in the def-

inition sketched above. For instance, the evolution rules can be *non-cooperating* (when single objects evolve; e.g, $a \to v$, where a is an object and v is a multiset of objects) or *cooperating* (when objects evolve together with other objects); a particular case of cooperating systems is that of systems with *catalysts* (a catalyst is an object c which is involved only in rules of the form $ca \to cv$: the object a evolves with the help of the catalyst c, but the latter one is reproduced after the evolution step). Then, the rules can be freely applied, or a *priority* relation can be considered (at each step, one applies only rules for which there is no rule of a higher priority which can also be applied at that step). The objects can be *communicated* from a region to another one, through membranes. This is done by associating them indications of the form *out* (the object must exit the current membrane; if this is the system skin, then the object leaves the system), *here* (the object remains in the same region), or *in* (the object must go into a lower level membrane, nondeterministically chosen). A stronger variant is to associate with objects and membranes *electrical charges*, $+, -$ and 0: an object with "polarization" $+$ will go into a membrane marked with $-$, one with "polarization" $-$ will go into a membrane marked with $+$; neutral objects are not moved. A still stronger version is to precisely indicate the target membrane by specifying the label of it. A powerful feature of P systems is the possibility of controlling the *thickness* of membranes: by *dissolving* a membrane, all objects of the former membrane are left free in the region containing it (this is a very powerful communication tool, because all objects are communicated in only one step); by *thickening* a membrane, we inhibit any communication through it. Finally, we can consider systems which are *synchronized* (an universal clock is used at the level of all membranes) or *non-synchronized*.

Many of these variants are computationally universal, they can simulate Turing Machines as mapping computing devices (the argument of a function is introduced in the initial configuration of the system and the value is read in an elementary membrane at the end of a computation). We refer to [36], [37], [11] for details.

Much more important is the fact that there are classes of P systems which can solve intractable problems in linear time. This is the case with systems where we can also *divide* membranes. Because of the interest of this result, we enter into some details.

More specifically, we will consider P systems where the central role in the computation is played by the membranes: evolution rules are associated both with objects and membranes, while the communication through membranes is performed with the direct participation of the membranes; moreover, the membranes can not only be dissolved, but they also can be multiplied by *division*. An elementary membrane can be divided by means of an interaction with an object from that membrane. Each membrane is supposed to have an "electrical polarization", one of the three possible: *positive, negative*, or *neutral*. If in a membrane we have two immediately lower membranes of opposite polarizations, one *positive* and one *negative*, then that membrane can also divide in such a way that the two membranes of opposite charge are separated; all membranes of neutral charge and all objects are duplicated and a copy of each of them is introduced in each of the two new membranes. The skin is never divided.

In this way, the number of membranes can grow, even exponentially. As expected, by making use of this increased parallelism we can compute faster: SAT (Satisfiability of Propositional Formulas in the Conjunctive Normal Form) can be solved in this framework in linear time (the time units are steps of a computation in a P system as sketched above, where we perform in parallel, in all membranes of the system, applications of

evolution rules or division of membranes). Moreover, the model is shown to be computationally universal: any computably enumerable set of (vectors of) natural numbers can be generated by our systems.

A P system with active membranes is a construct

$$\Pi = (V, T, H, \mu, w_1, \ldots, w_m, R),$$

where:

(i) $m \geq 1$ (the initial degree of the system);

(ii) V is an alphabet (the *total alphabet* of the system);

(iii) $T \subseteq V$ (the *terminal* alphabet);

(iv) H is a finite set of *labels* for membranes;

(v) μ is a *membrane structure*, consisting of m membranes, labeled (not necessarily in a one-to-one manner) with elements of H; all membranes in μ are supposed to be neutral;

(vi) $w_1, \ldots, w_m$ are strings over V, describing the *multisets of objects* placed in the m regions of μ;

(vii) R is a finite set of *developmental rules*, of the following forms:

(a) $[_h a \to v]_h^\alpha$,
for $h \in H, \alpha \in \{+, -, 0\}, a \in V, v \in V^*$
(object evolution rules, associated with membranes and depending on the label and the charge of the membranes, but not directly implying the membranes, in the sense that the membranes are neither taking part to the application of these rules nor are they modified by them);

(b) $a[_h\,]_h^{\alpha_1} \to [_h b]_h^{\alpha_2}$,
for $h \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in V$
(communication rules; an object is introduced in the membrane, possibly modified during this process; also the polarization of the membrane can be modified, but not its label);

(c) $[_h a\,]_h^{\alpha_1} \to [_h\,]_h^{\alpha_2} b$,
for $h \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in V$
(communication rules; an object is sent out of the membrane, possibly modified during this process; also the polarization of the membrane can be modified, but not its label);

(d) $[_h a\,]_h^\alpha \to b$,
for $h \in H, \alpha \in \{+, -, 0\}, a, b \in V$
(dissolving rules; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);

(e) $[_h a\,]_h^{\alpha_1} \to [_h b\,]_h^{\alpha_2} [_h c\,]_h^{\alpha_3}$,
for $h \in H, \alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}, a, b, c \in V$
(division rules for elementary membranes; in reaction with an object, the

membrane is divided into two membranes with the same label, possibly of different polarizations; the object specified in the rule is replaced in the two new membranes by possibly new objects; all other objects are reproduced in both the two new membranes);

(f) $\left[_{h_0}\left[_{h_1}\right]^{\alpha_1}_{h_1} \cdots \left[_{h_k}\right]^{\alpha_1}_{h_k} \left[_{h_{k+1}}\right]^{\alpha_2}_{h_{k+1}} \cdots \left[_{h_n}\right]^{\alpha_2}_{h_n}\right]^{\alpha_0}_{h_0}$
$\rightarrow \left[_{h_0}\left[_{h_1}\right]^{\alpha_3}_{h_1} \cdots \left[_{h_k}\right]^{\alpha_3}_{h_k}\right]^{\alpha_5}_{h_0} \left[_{h_0}\left[_{h_{k+1}}\right]^{\alpha_4}_{h_{k+1}} \cdots \left[_{h_n}\right]^{\alpha_4}_{h_n}\right]^{\alpha_6}_{h_0}$,
for $k \geq 1, n > k, h_i \in H, 0 \leq i \leq n$, and $\alpha_0, \ldots, \alpha_6 \in \{+, -, 0\}$ with $\{\alpha_1, \alpha_2\} = \{+, -\}$; if this membrane with the label h_0 contains other membranes than those with the labels $h_1, \ldots, h_n$ specified above, then they should have neutral charge in order to allow the application of this rule
(division of non-elementary membranes; this is possible only if a membrane contains two immediately lower membranes of opposite polarization, + and −; the membranes of opposite polarizations are separated in the two new membranes, but their polarization can change; always, all membranes of opposite polarizations are separated by applying this rule; all objects and all other membranes from membrane h_0 are reproduced in both the two new membranes with label h_0).

Note that in all rules of types (a) – (e) only one object is specified (that is, objects do not directly interact) and, with the exception of rules of type (a), single objects are always transformed into single objects (the two objects produced by a division rule of type (e) are placed in two different regions).

These rules are applied according to the following *principles*:

1. All the rules are applied in parallel: in a step, the rules of type (a) are applied to all objects to which they can be applied, all other rules are applied to all membranes to which they can be applied; an object can be used by only one rule, non-deterministically chosen (there is no priority relation among rules), but any object which can evolve by a rule of any form, should evolve.

2. If a membrane is dissolved, then all the objects in its region are left free in the region immediately above it. Because all rules are associated with membranes, the rules of a dissolved membrane are no longer available at the next steps. The skin membrane is never dissolved.

3. All objects and membranes not specified in a rule and which do not evolve are passed unchanged to the next step. For instance, if a membrane with the label h is divided by a rule of type (e) which involves an object a, then all other objects in membrane h which do not evolve are introduced in each of the two resulting membranes h. Similarly, when dividing a membrane h by means of a rule of type (f), the neutral membranes are reproduced in each of the two new membranes with the label h, unchanged if no rule is applied to them (in particular, the contents of these neutral membranes are reproduced unchanged in these copies, providing that no rule is applied to their objects).

4. If at the same time a membrane h is divided by a rule of type (e) and there are objects in this membrane which evolve by means of rules of type (a), then in the new copies of the membrane we introduce the result of the evolution; that is, we may suppose that first the evolution rules of type (a) are used, changing the objects,

and then the division is produced, so that in the two new membranes with label h we introduce copies of the changed objects. Of course, this process takes only one step. The same assertions apply to the division by means of a rule of type (f): always we assume that the rules are applied "from bottom-up", in one step, but first the rules of the innermost region and then level by level until the region of the skin membrane.

5. The rules associated with a membrane h are used for all copies of this membrane, irrespective of the fact that the membrane is an initial one or it is obtained by division. At one step, a membrane h can be the subject of only one rule of types (b) – (f).

6. The skin membrane can never divide. As any other membrane, the skin membrane can be "electrically charged".

We can pass from a configuration to another one by using the rules from R according to the principles given above. We say that we have a (direct) *transition* among configurations.

As usual, a computation is *complete* if it cannot be continued: there is no rule which can be applied to objects and membranes in the last configuration.

Note that during a computation the number of membranes (hence the degree of the system) can increase and decrease but the labels of these membranes are always among the labels of membranes present in the initial configuration (by division we only produce membranes with the same label as the label of the divided membrane).

As we mentioned above, the SAT problem can be solved by a P system with active membranes in a time which is linear in the number of variables and the number of clauses.

This is obtained by trading space for time: an exponential number of membranes is created (of the order of 2^n, where n is the number of variables in the formula we want to decide), which work in parallel; a certain object leaves the system if and only if the considered formula is satisfiable. Details can be found in [38].

The idea from [38] has been extended in [28] to other two NP-complete problems, the Hamiltonian Path Problem (like in Adleman's experiment, but without specifying the initial and the final node) and the Node Covering Problem. Also these problems were solved in a linear parallel time.

In the first section we have said that no experiment was reported so far in the Membrane Computing area. On the other hand, it is not clear which is the right way to proceed towards implementation: to try to build a wet computer based on membranes or to use the silicon (building a purpose-designed computer or using the existing computers, in the same way as we use them for Genetic Algorithms). This is a fundamental question which we do not address here.

4 Quantum Computing

Quantum computing is the fifth new paradigm of Natural Computing. It has originated in the studies of thermodynamics of computation initiated by Landauer [29, 30], Feynman [18, 19, 20] Bennett [4, 5, 6]. "Real computers", in contrast with "paper machines"

like Turing machines, Chomsky grammars or Markov algorithms, are physical devices: whatever they can or cannot do is determined by the laws of physics.

In Deutsch's[4] words ([14, p. 101]; see also [16]):

> *The reason why we find it possible to construct, say, electronic calculators, and indeed why we can perform mental arithmetic, cannot be found in mathematics or logic.* The reason is that the laws of physics "happen" to permit the existence of physical models for the operations of arithmetic *such as addition, subtraction and multiplication. If they did not, these familiar operations would be non-computable functions. We might still know* of *them and invoke them in mathematical proofs (which would be presumably called "non-constructive") but we could not perform them.*

Quantum mechanical effects,[5] like the exponential state space, the entangled states, and the linearity of quantum state transformations, make the real power of these new machines, the exponential parallelism. Programming quantum machines requires innovative techniques; people have only recently begun to research such techniques. Among the most notable successes is Shor's [41] polynomial-time factorization algorithm, Grover's [22] database search algorithm.

To get just a glimpse of the Quantum Computing we will present two examples of problems which cannot be solved efficiently by conventional computing methods: the implementation of the square root of NOT gate and the solution to Deutsch problem. First, we need a bit of formalism.

A classical **bit** (e.g., the position of gear teeth in Babbage's differential engine, a memory element or wire carrying a binary signal, in contemporary machines) is a system comprising many atoms. Typically, the system is described by one or more continuous parameters, for example, voltage. Such a parameter is used to separate the space into two well-defined regions chosen to represent 0 and 1. Manufacturing imperfections, local perturbations may affect, so signals are periodically restored toward these regions to prevent them from drifting away. An n-bit register of memory can exist in any of 2^n logical states, from $00\ldots0$ (n zeros) to $11\ldots1$ (n ones).

A quantum event in which we have two possible mutually exclusive outcomes is the elementary act of observation: all knowledge of the physical world is based upon such acts. An elementary act of observation is simultaneously like a coin-toss and not like a coin-toss. The information derived from an elementary act of observation is no more than a single bit, but *there is more on it than that.* To mark this difference B. Schumaker [40] has coined the name qubit. A quantum bit, **qubit**, is typically a microscopic system, such as an atom or nuclear spin or polarized photon. For example, the state of a spin-$\frac{1}{2}$ particle, when measured, is always found to be in one of two possible states, represented as

$$|+\frac{1}{2}\rangle \text{ (spin-up) or } |-\frac{1}{2}\rangle \text{ (spin-down).}$$

There is nothing special about spin systems–any 2-state quantum system can be equally used to represent 0 and 1. What is really special here is the existence of a

[4]The author of the first "fully" quantum computer, [14]; see also [15]

[5]Not all quantum effects may be needed for computation. *Nonlinearity* (to support quantum logic and ensure universality) and *coherence* (for the manipulation of coherent quantum superpositions) are necessary and, in principle, sufficient conditions for computation.

continuum of intermediate states which are superpositions of 0s and 1s. Mathematically, they are just linear combinations of the basis states. Unlike the intermediate states of a classical bit (for example, any voltages between the "standard" representations of 0 and 1) which can be distinguished from 0 and 1, but do not exist from an informational point of view, quantum intermediate states cannot be reliably distinguished, even in principle, from the basis states, but do have an informational "existence".

Consider the two dimensional space Hilbert space $\mathbf{C}^2$, and fix a orthonormal basis, $\{|0\rangle, |1\rangle\}$. These vectors, $|0\rangle$ and $|1\rangle$, correspond to the classical bit values 0 and 1, respectively. A qubit is a unit vector in the space $\mathbf{C}^2$, so for each qubit $|x\rangle$, there are two (complex) numbers $a, b \in \mathbf{C}$ such that

$$|x\rangle = a|0\rangle + b|1\rangle = \begin{pmatrix} a \\ b \end{pmatrix}, \tag{1}$$

and $|a|^2 + |b|^2 = 1$.

An n-qubit system can exist in any superposition of the form

$$\Psi = \sum_{x=00\ldots0}^{11\ldots1} c_x|x\rangle, \tag{2}$$

where c_x are (complex) numbers such that $\sum_x |c_x|^2 = 1$.[6] The exponential "explosion" represented by formula (2) distinguishes quantum systems from classical ones: in a classical system a state is described by a number of parameters growing only linearly with the size of the system,[7] but most quantum systems do not admit such a description because quantum states may be "entangled".

The quantum evolution of a qubit is described by a "unitary operator", that is an operator induced by a unitary matrix.[8] Here is a simple, but important example. Let θ be a real number in the interval $[0, 2\pi)$ and consider the rotation R_θ given by

$$R_\theta = \begin{pmatrix} cos\,\theta & -sin\,\theta \\ sin\,\theta & cos\,\theta \end{pmatrix}.$$

Hence, R_θ acts as follows:

$$|0\rangle \mapsto cos\,\theta|0\rangle + sin\,\theta|1\rangle,\ |1\rangle \mapsto -sin\,\theta|0\rangle + cos\,\theta|1\rangle.$$

One can easily verify that $R_\theta R_\theta^\dagger = R_\theta R_\theta^T = I$, hence R_θ is unitary. For $\theta = 0$ we get the identity transformation: $R_0 = I$.

We may think of logic gates as transformations. For example, the NOT transformation which interchanges the vectors $|0\rangle$ and $|1\rangle$, is given by R_π, that is the matrix

$$\text{NOT } = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

[6] Re-phrased, a quantum state of n qubits is just a direction in a Hilbert space of dimension equal to the number of classical states, i.e., 2^n.

[7] Reason: classical systems are completely described locally, that is, via each state in part.

[8] A quadratic matrix A of order n over $\mathbf{C}$ is *unitary* if $AA^\dagger = I$ (the identity $n \times n$ matrix); $A^\dagger$ is the transposed conjugate matrix of A.

It flips that state of its input,

$$\text{NOT } |0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle,$$

and

$$\text{NOT } |1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle.$$

The phase shift gate $Shift$ is defined as follows: $Shift\,|0\rangle = |0\rangle, Shift\,|1\rangle = -|1\rangle$, so

$$Shift = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Since NOT $\cdot$ NOT $^\dagger = I$ and $Shift \cdot Shift^\dagger = I$, the operators NOT and $Shift$ are also unitary. The operator NOT $\cdot\, Shift$ is also a unitary transformation and we have:

$$\text{NOT} \cdot Shift|0\rangle = \text{NOT } |1\rangle = -|1\rangle,$$

$$\text{NOT} \cdot Shift|1\rangle = \text{NOT } |0\rangle = |0\rangle.$$

Therefore, its associated matrix is

$$R_{3\pi/2} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

The square-root of NOT (introduced by Deutsch [15]) is the transformation

$$\sqrt{\text{NOT}}\ : \begin{array}{lcl} |0\rangle & \rightarrow & \frac{1}{2}(1+i)|0\rangle + \frac{1}{2}(1-i)|1\rangle, \\ |1\rangle & \rightarrow & \frac{1}{2}(1-i)|0\rangle + \frac{1}{2}(1+i)|1\rangle, \end{array}$$

$$\sqrt{\text{NOT}} = \frac{1}{2} \begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix}.$$

A routine check shows that

$$\sqrt{\text{NOT}} \cdot \sqrt{\text{NOT}} = \text{NOT}\ , \tag{3}$$

and

$$\sqrt{\text{NOT}} \cdot \sqrt{\text{NOT}}^\dagger = \frac{1}{4} \begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix} \begin{pmatrix} 1-i & 1+i \\ 1+i & 1-i \end{pmatrix} = I.$$

The square-root of NOT is a typical "quantum" gate in the sense that *it is impossible to have a single-input/single-output classical binary logic gate that satisfies* (3). Indeed, any classical binary

$$\sqrt{\text{NOT}}_{\,\text{classical}}$$

gate is going to output a 0 or a 1 for each possible input 0/1. Assume that we have such a classical square-root of NOT gate acting as a pair of transformations

$$\sqrt{\text{NOT}}_{\,\text{classical}}(0) = 1,\ \sqrt{\text{NOT}}_{\,\text{classical}}(1) = 0.$$

Then, two consecutive applications of it will *not* flip the input!

The simplest way to illustrate the power of quantum parallelism is to solve the so-called *Deutsch's problem*. Consider a Boolean function $f : \{0,1\} \to \{0,1\}$ and suppose that we have a black box to compute it. We would like to know whether f is constant (that is, $f(0) = f(1)$) or balanced ($f(0) = f(1)$). To make this test classically, we need two computations of f, $f(0)$ and $f(1)$ and one comparison. Is it possible to do it better? The answer is affirmative, and here is a possible solution.

Suppose that we have a quantum black box to compute f. Consider the transformation U_f which applies to two qubits, $|x\rangle$ and $|y\rangle$ and produces $|x\rangle|y \oplus f(x)\rangle$, see Figure 4; by $\oplus$ we denote the sum modulo 2. The transformation U_f flips the second qubit if f acting on the first qubit is 1, and does nothing if f acting on the first qubit is 0.

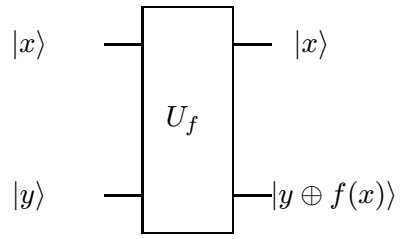

Figure 4: Quantum gate array U_f.

The black box is "quantum", so we can chose the input state to be a superposition of $|0\rangle$ and $|1\rangle$. Assume first that the second qubit is initially prepared in the state $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Then,

$$\begin{aligned} U_f\left(|x\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right) &= |x\rangle\frac{1}{\sqrt{2}}(|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle) \\ &= (-1)^{f(x)}|x\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \end{aligned}$$

Next take the first qubit to be $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. The black box will produce

$$\begin{aligned} U_f\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right) &= \frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\ &= (-1)^{f(0)}(|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle)(|0\rangle - |1\rangle). \end{aligned}$$

Next we will perform a measurement that projects the first qubit onto the basis $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$: we will obtain $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ if the function f is balanced and $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ in the opposite case. So, Deutsch's problem was solved with only one computation of f. The explanation consists in the ability of a quantum computer to be in a blend of states: we can compute $f(0)$ and $f(1)$, but also, and more importantly, we can extract some information about f which tells us whether $f(0)$ is equal or not to $f(1)$.

The above idea can be easily generalised to Boolean functions of n variables. To compute a complete set of values for such a function we need to calculate f in all 2^n points, an infeasible task if n is big (say, $n = 100$). With a quantum computer we use the transformation $U_f(|x\rangle|0\rangle) = |x\rangle|f(x)\rangle$, chose the input register to be in the state

$$\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)^n = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle,$$

and with just a single computation of f generate the state

$$\frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle|f(x)\rangle. \tag{4}$$

Of course, if we want to compute *all* the values of f we need to come back to 2^n computations of f: the point is that in some cases we don't need to get all values of f, but some "global" information about f, and it is possible to "extract" this information from (4).

Conventional devices for constructing quantum computers include ion traps, high finesse cavities for manipulating light and atoms using quantum electrodynamics, and molecular systems designed to compute using nuclear magnetic resonance. These latter store quantum information on the states of quantum systems as photons, atoms, or nuclei, and realise quantum logic by semiclassical potentials such as microwave or laser fields. Unconventional ideas for quantum computation include Fermionic quantum computers, Bosonic computers (which use photons, phonons, or atoms in a Bose-Einstein condensate), and architectures relying on anyons (their nonlocal topological nature make them intrinsically error-correcting and virtually immune to noise and interference). An intriguing idea is to consider designs that are not based on experimentally confirmed physical phenomena, but rather are based on speculative, hypothetical and not yet verified phenomena. A particularly interesting variation on conventional physics is nonlinear quantum mechanics (a discrete model based on cellular automata might be such a model). See more in [10, 9, 23, 11].

5 Final Comments

Natural Computing has the potential to dramatically change the way we think and do Computing and Mathematics. The challenge for computer scientists and mathematicians is to provide models for the emerging computing architectures, to develop new programming techniques appropriate for Natural Computing and to apply them to various domains.

References

[1] L. M. Adleman: Molecular computation of solutions to combinatorial problems. *Science*, 226 (November 1994), 1021–1024.

[2] L. M. Adleman: On constructing a molecular computer. [32], 1–22.

[3] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, J. D. Watson: *Molecular Biology of the Cell*. 3rd ed., Garland Publishing, New York, 1994.

[4] C. Bennett: Logical reversibility of computation, *IBM J. Res. Dev.*, 17 (1973), 525–532.

[5] C. H. Bennett: The thermodynamics of computation, *International Journal of Theoretical Physics* 21 (1982), 905-940.

[6] C. H. Bennett, R. Landauer: The fundamental physical limits of computation, *Scientific American*, July (1985), 48–56.

[7] E. Baum, D. Boneh, P. Kaplan, R. Lipton, J. Reif, N. Seeman (eds.): *DNA Based Computers*. Proc. of the Second Annual Meeting, Princeton, 1996

[8] D. Bray: Protein molecules as computational elements in living cells. *Nature*, **376** (1995), 307–312.

[9] C. S. Calude, J. L. Casti: Parallel thinking, *Nature* 392, 9 April (1998), 549–551.

[10] C. S. Calude, J. Casti, M. J. Dinneen, eds.: *Unconventional Models of Computation*, Springer-Verlag, Singapore, 1998.

[11] C. S. Calude, Gh. Păun: *Computing with Cells and Atoms*. Taylor and Francis, London, to appear.

[12] M. Conrad: Information processing in molecular systems. *Currents in Modern Biology*, 5 (1972), 1–14.

[13] M. Conrad: On design principles for a molecular computer, *Comm. of the ACM*, 28 (1985), 464–480.

[14] D. Deutsch: Quantum theory, the Church-Turing principle and the universal quantum computer, *Proceedings of the Royal Society London*, A 400 (1985), 97–119.

[15] D. Deutsch: Quantum computation, *Physics World*, 5 (1992), 57–61.

[16] D. Deutsch: *The Fabric of Reality*, Allen Lane, Pengiun Press, 1997.

[17] R. P. Feynman, In D. H. Gilbert (ed.): *Miniaturization*. Reinhold, New York, 1961, 282–296.

[18] R. P. Feynman: Simulating physics with computers, *International Journal of Theoretical Physics* 11 (1985), 11–20.

[19] R. P. Feynman: Quantum mechanical computers, *Optics News* 21 (1982), 467–488.

[20] *Feynman Lectures on Computation*, J. G. Hey and R. W. Allen (eds.), Addison-Wesley, Reading, Massachusetts, 1996.

[21] M. R. Garey, D. S. Johnson: *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freman and Comp., San Francisco, 1979.

[22] L.K. Grover: A fast quantum mechanical algorithm for database search, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, 1996, pp. 212–219.

[23] J. Gruska: *Quantum Computing*, McGraw-Hill, London, 1999.

[24] J. Hartmanis: About the nature of computer science. *Bulletin of the EATCS*, 53 (June 1994), 170–190.

[25] J. Hartmanis: On the weight of computation. *Bulletin of the EATCS*, 55 (February 1995), 136–138.

[26] S. Haykin: *Neural Networks – a Comprehensive Foundation*. IEEE Press, 1994.

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

[28] S. N. Krishna, R. Rama: A variant of P systems with active membranes: Solving NP-complete problems, *Romanian J. of Information Science and Technology*, 2, 4 (1999), in press.

[29] R. Landauer: Irreversibility and heat generation in the computing process, *IBM J. Res. Develop.*, 5 (1961), 183–191.

[30] R. Landauer: Information is physical, *Physics Today* 44 (1991), 23–29.

[31] R. J. Lipton: Using DNA to solve NP-complete problems. *Science*, 268 (April 1995), 542–545.

[32] R. J. Lipton, E. B. Baum, eds.: *DNA Based Computers*. Proc. of a DIMACS Workshop, Princeton, 1995, Amer. Math. Soc., 1996.

[33] V. Nissen: Evolutionary algorithms in management applications and other classic optimization problems, section F 1.2 of *Handbook of Evolutionary Computation*, Oxford Univ. Press, 1997.

[34] Gh. Păun: (*DNA*) *Computing by carving*. Research Report CTS-97-17, Center for Theoretical Study of the Czech Academy of Sciences, Prague, 1997, and *Soft Computing*, 3, 1 (1999), 30–36.

[35] Gh. Păun, ed.: *Computing with Bio-Molecules. Theory and Experiments*. Springer-Verlag, Singapore, 1998.

[36] Gh. Păun, Computing with membranes: *Journal of Computer and System Sciences*, to appear, and *Turku Center for Computer Science-TUCS Report* No 208, 1998 (www.tucs.fi).

[37] Gh. Păun: Computing with membranes. An introduction: *Bulletin of the EATCS*, 67 (Febr. 1999), 139–152.

[38] Gh. Păun: P systems with active membranes: Attacking NP complete problems. Submitted 1999, and *Auckland University, CDMTCS Report* No 102, 1999 (www.cs.auckland.ac.nz/ CDMTCS).

[39] Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing. New Computing Paradigms*. Springer-Verlag, Berlin, 1998.

[40] B. Schumaker: Quantum coding, *Physical Review A*, 51, 4 (1995), 2738–2747.

[41] P.W. Shor: Algorithms for quantum computation: discrete log and factoring, *Proceedings of the 35th IEEE Annual Symposium on Foundations of Computer Science*, 1994, 124–134.

[42] Y. Suzuki, H. Tanaka: On a LISP implementation of a class of P systems, submitted, 1999.