11-2019

Computing Without Computing: DNA Version

Vladik Kreinovich

Julio C. Urenda

Comments:
Technical Report: UTEP-CS-19-107

Computing Without Computing: DNA Version

Vladik Kreinovich[1] and Julio C. Urenda[1,2]
[1]Department of Computer Science
[2]Department of Mathematical Sciences
University of Texas at El Paso
El Paso, TX 79928, USA
vladik@utep.edu. jcurenda@utep.edu

Abstract

The traditional DNA computing schemes are based on using or simulating DNA-related activity. This is similar to how quantum computers use quantum activities to perform computations. Interestingly, in quantum computing, there is another phenomenon known as *computing without computing*, when, somewhat surprisingly, the result of the computation appears without invoking the actual quantum processes. In this chapter, we show that similar phenomenon is possible for DNA computing: in addition to the more traditional way of using or simulating DNA *activity*, we can also use DNA *inactivity* to solve complex problems. We also show that while DNA computing without computing is *almost as powerful* as traditional DNA computing, it is actually *somewhat less powerful*. As a side effect of this result, we also show that, in general, security is somewhat more difficult to maintain than privacy, and data storage is more difficult than data transmission.

1 Introduction

In his famous 1994 paper [1], Leonard Adleman showed that, in principle, we can drastically speed up computations if we use the fact that DNA fragments combine together – in a process known as *ligation* – if we corresponding nucleotides match, i.e., if:

- A is matched with T,
- T is matched with A,
- C is matched with G, and
- G is matched with C.

For example, two fragments ACTTG and TGAAC match perfectly.

Specifically, this paper showed that we can speed up the solution to the following *Hamiltonian path problem*:

- given a graph,
- find a path in this graph that visits every vertex exactly once.

This seminal paper started the field of *DNA computing*, which now includes both:

- using actual DNA fragments (as Adleman did) and
- using computer simulation of the corresponding processes.

One of the main advantages of computing via molecular interactions, when each molecule serves as a processor, is that in each mole, we have 10^{23} molecules – and thus, 10^{23} processor working in parallel. Such unbelievable parallelism – many orders of magnitude higher than the usual thousands of processors in a supercomputer – is a clear indication that this approach has a great potential.

Later, similar DNA-based algorithms were proposed for solving other complex problems, such as propositional satisfiability (this problem is explained, in detail, later in this chapter). For reasonably recent overviews, see, e.g., [3, 9, 23, 28, 29].

All these algorithms are based on actually using (or simulating) the ligation process. This is similar to how quantum computers use quantum activities to perform computations. Interestingly, in quantum computing, there is another phenomenon known as *computing without computing*, when, somewhat surprisingly, the result of the computation appears without actually invoking quantum processes. In this chapter, we show that similar phenomenon is possible for DNA computing:

- in addition to the more traditional way of using or simulating DNA *activity*,
- we can also use DNA *inactivity* to solve complex problems.

2 Computing Without Computing – Quantum Version: A Brief Reminder

DNA computing is one of several directions in the general quest for using novel physical phenomena in computing. Another – probably even more well known – direction is *quantum computing*, the use of quantum effects to speed up computations; see, e.g., [25, 33].

Most quantum algorithms actually use quantum effects to perform the corresponding computations, but there is an interesting version called *counterfactual quantum computing*, or, alternatively, *computing without computing*. The idea is that:

- we *set up* the corresponding quantum computations, but
- we *do not* actually *run* them,

and still, because of the quantum effects, we get the desired result with some probability.

This idea was first proposed in [11]. The main motivation behind this idea was not so much about *computing* but rather about *testing*: the same idea can be, in principle, used to test the complex equipment without actually running it. For example, in principle, we can test whether the atomic bomb (that has been in storage for a long time) will actually explode when triggered – without actually having to explode it to find this out.

At this moment, this quantum computing-without-computing phenomenon is far from practical use – just like most quantum computing algorithms and most DNA computing algorithms are still far from practical use. However, there has been a lot of progress in this direction. For example:

- Initially, there was a fear that the probability of getting the correct result in the computing-without-computing setting may be too low to be practically useful.
- However, in 2006, a seminal paper [8] showed that this probability can be increased to almost 1.

The fact that in quantum computing, it is possible to perform some computations without actually running these computations encouraged us to check whether a similar phenomenon is possible for DNA computing as well. We were even further encouraged by the fact that computing without computing is also theoretically possible in yet another direction of using novel physical phenomena in computing – namely, in the use of acausal effects. Let us briefly recall this idea.

3 Computing Without Computing – Version Involving Acausal Processes: A Reminder

How can we speed up computations? A natural science-fiction idea is to use a *time machine* (also known as an *acaual* – i.e., causality violating – process):

- we let the computer spend as much time as needed, even it means several thousand years, and then
- we use the time machine to bring these results back to us.

For a long time, acausal processes remained mostly the subject of science fiction. Serious physicists mostly believed that time machines are not possible – due to well known paradoxes. These paradoxes can be summarized by stating the probably well known paradox of time travel – the grandfather paradox: what if a time traveler goes into the past and kills his own grandfather before the traveler's parents are conceived?

In spite of the paradoxes, acausal processes continued to naturally emerge in many areas of physics. This emergence is mostly related to the fact that:

- in contrast to pre-quantum physics, where everything is deterministic,
- in quantum physics, we can only make probabilistic predictions.

In other words, there are always fluctuations, deviations of the actual values from the expected values of the corresponding physical quantities.

In pre-quantum physics, at each moment of time, a particle is in a certain spatial location, with a certain velocity – and, in principle, we can measure both location and velocity with any desired accuracy. In quantum physics, such exact measurements are no longer possible. A particle's location and velocity are always probabilistic: e.g., even if we prepare several particles in the identical states and measure their velocities, we will get slightly different results. And the smaller the region we consider, the larger these fluctuations.

Similarly, the space-time tensor – that describes the geometry of space-time and the direction of causality – fluctuates. The smaller the region we consider, the larger these fluctuations. As a result, the maximal possible speed fluctuates from the usual macroscopic speed-of-light value c:

- in some microscopic locations, the maximal speed is slightly larger than c, while
- in some other microscopic locations, the maximal speed is slightly smaller than c.

If a micro-particle follows the locations when the local maximal speed is larger than c, then, from the macroscopic viewpoint, this perfectly physical particle goes faster than the speed of light – and, according to special relativity, this implies the possible of going back in time.

Many other schemes naturally appeared in physics, thus leading to acausal effects. As a result, in the late 1980s, a group of physicists led by a future Nobelist Kip Thorne decided to overcome the previous taboo and to seriously analyze possible acausal processes; see, e.g., [21, 22, 30, 31].

But what about the paradoxes? Here, the probabilistic nature of quantum physics also helps. As we have mentioned, in quantum world, nothing is guaranteed. If the time traveler attempted to kill his grandfather; then:

- since the grandfather was alive enough after that attempt to sire a son,
- this means that this attempt failed.

In other words, some event happened which prevented the killing:

- maybe a policeman walked by and prevented the murder,
- maybe the gun got stuck,
- maybe a meteorite fell on the gun at that exact moment.

We can try to prevent all such events, but no matter how much we try, no matter how many possibilities we take into account, there is always a possibility of some rare, low-probability event that would disrupt the process. So, the only real consequence of trying to implement a time-travel paradox is that some very low-probability event will happen.

And, interestingly, this can be used to computations – i.e., we can use the *possibility* of acausal effects to perform computations without actually invoking these effects. In other words, we have another case of computing without computing. Indeed, suppose, e.g., that:

- we are given a graph, and
- we need to find a Hamiltonian path in this graph.

What we can do is:

- use a random number generators to generate some (random) path through this graph, and then
- check if the resulting path is Hamiltonian.

If the path is not Hamiltonian, we launch a time machine – which is set up in such a way that its launch leads to some low-probability event, with probability $p_0 \ll 1$.

On the other hand, e.g., in a binary graph, the probability that a random selection of a direction at each of n nodes will lead to a selected path is 2^{-n}. So, nature has a choice:

- it can set up random processes so as to select a Hamiltonian path, or
- it will have to implement a low-probability event, with probability $p_0 \ll 1$.

According to the general idea of statistical physics, in most cases, nature selects the event with higher probability. So, if $p_0 \ll 2^{-n}$, nature will select a Hamiltonian path – and thus, we will find this path fast without actually having to use the time machine.

Comments.

- This idea is described, e.g., in [5, 14, 15, 16, 17, 19, 20, 26].
- Now that we have learned how computing without computing is possible in quantum and acausal computing, let us show how (and why) this idea is possible in DNA computing as well.

4 Computing Without Computing – DNA Version

Main idea. Let us show that with DNA computing, it is also possible to solve complex problems by using or simulating DNA inactivity.

The possibility of inactivity makes perfect biological sense:

- when resources are plentiful, it makes sense for the living creatures to be active and to actively multiply, but
- in situations when resources become scarce, such an activity would exhaust these resources really fast.

In such situations, it is important to slow down all the biological processes as much as possible.

In nature, we observe such slowing down all the time:

- from hibernating bears
- to plants that stop practically all activities in winter
- to bacteria and viruses that can slow down to such an extent that they can survive in this slowed-down condition for hundreds and even thousands of years.

The slow-down occurs on all the levels:

- from the macro level – when an animal (e.g., a hibernating bear) stops moving almost completely,
- to the cell level, where all the usual biochemical processes grind practically to a halt.

On the DNA level, this means that instead of enhancing the possible ligations, in such situations, the cell tries to prevent ligation as much as possible, so as to keep all the processes inactive. This phenomenon has indeed been traced on the gene expression level; see, e.g., [10]. The possibility for such prevention comes from the fact that:

- contrary to a somewhat simplified version of DNA processes used in the traditional DNA computing,
- the actual DNA-related biochemical processes do not simply involve matching of different parts of the RNA and DNA.

There is also a *control* that switches some genes (i.e., some parts of the RNA and DNA) on and off. This control is determined:

- partly by other genes, and
- partly by the signals that the cell gets from the environment.

From this viewpoint, in the case of scarce resources, the corresponding control processes are organized in a way to maximally prevent ligations.

We will describe this control process in precise terms, and let us show that the corresponding problem is NP-hard – which means that it can be used to solve complex computational problems. But before we do that, let us explain why we believe that such control can be used for computations.

It is not easy to stop biological processes. The great potential of DNA computing comes from the fact that the corresponding biological processes are very complicated. In spite of the original optimism, even though the genomes of many living creatures – including humans – have been decoded, we are almost as far from the full understanding the corresponding processes as before – and even farther from artificially synthesizing even the simplest living creatures. The problems are complex, but within each of numerous cells of numerous living creatures, nature solves the corresponding complex problems all the time. Thus, it is natural to try to use these naturally occurring solutions to solve our complex problems.

DNA processes are complex, but nature knows how to solve them – and thus, they occur all the time. Stopping these processes is much more difficult, even for nature – indeed, very few living creatures can do it, and we are still far from understanding how this is done.

- A grain left outside eventually spoils and rots, but some grains got preserved for thousands of years – and when planted, turned into plants.
- Freezing kills most living creatures, but some mysteriously survive – and get revised when thawed out.
- Viruses and bacteria can survive for years in the cosmic cold – there is even a *panspermia* hypothesis that this is how life spreads between the planets, this is how originated on Earth.
- The possibility to stop biological processes in a human being – known as *anabiosis* – is a common feature in science fiction, but in real life, it remains a far-from-possible dream.

Since stopping of biological processes is to difficult, even more difficult that running them, it is even more reasonable to use this stopping – in addition to the DNA processes themselves – to solve other complex problems.

Towards describing ligation prevention in precise terms. In general, we have several fragments that, in principle, have matching parts. Each fragment consists of several sub-fragments, and we can decide which of these sub-fragments is switched on to be active. We want to select the sub-fragments in such a way that no two active sub-fragments are matched.

Here is a precise formulation of the problem.

What is given. We have several (N) nucleotide sequences ("fragments") s_1, $\ldots$, s_N, i.e., sequences consisting of symbols C, G, A, and T. Each fragment s_i is a concatenation of several subsequences ("sub-fragments") $s_i = s_{i1} \ldots s_{ik_i}$.

The sub-fragments s and s' *match* (or are *complementary*) if s' can be obtained from s by replacing A with T, T with A, C with G, and G with C.

What we want to find. The problem is to find the integers $j_1, \ldots, j_N$ such that $1 \le j_i \le k_i$ and for every two fragments i and i', the corresponding sub-fragments s_{ij_i} and $s_{i'j_{i'}}$ do not match.

Let us prove that the ligation prevention problem is NP-hard. In practice, we are usually interested in the problems in which, once someone provides us with a candidate for a solution, we can feasibly tell whether this is a solution or not. The class of all such problems is known as the class NP; see, e.g., [16, 27].

Some computational problems are NP-hard, meaning that every problem from the class NP can be reduced to this problem. In other words, if we have an efficient algorithm for solving an NP-hard problem, this means that by reducing to this problem, we can solve *any* practical problem in feasible time [16, 27].

If a problem is NP-hard *and* itself belongs to the class NP, then this general problem is known as *NP-complete*.

Let us show that the ligation prevention problem is NP-hard. Since it is easy to check that no two sub-fragments are complementary to each other, this means that this problem is also in the class NP and is, thus, actually NP-complete.

This would mean that, if – as we believe – nature has a way to solve the ligation prevention problem (at least many instances of this problem), then by reducing to this problem, we will be able to solve many practical problems in reasonable time.

How NP-hardness is usually proved. To show that a given problem P_{given} is NP-hard, it is sufficient to show that a known NP-hard problem P_{known} can be reduced to this problem. Indeed, by definition of NP-hardness, every problem P from the class NP can be reduced to P_{known}, and since the problem P_{known} can be, in its turn, reduced to P_{given}, this would mean that a two-stage reduction $P \to P_{\text{known}} \to P_{\text{given}}$ reduces P to P_{given}. Since this is true for every problem P from the class NP, this means that the given problem P_{given} is indeed NP-hard.

How we will prove NP-hardness. As the known problem P_{known}, we select the propositional satisfiability problem for 3-CNF formulas, historically the first problem proven to be NP-hard. In this general problem, we deal with *Boolean* (= *propositional*) variables, i.e., variables $x_1, \ldots, x_v$ that can take two possible values: 1 (meaning "true") and 0 (meaning "false"). A *literal* a is either a variable x_k or its negation $\neg x_k$.

A *clause* is an expression of the type $a \vee b$ or $a \vee b \vee c$ where a, b, and c are literals. Examples are $x_1 \vee \neg x_2$ or $\neg x_1 \vee \neg x_5 \vee x_9$.

Finally, a *formula* F is an expression of the type $C_1 \,\&\, C_2 \,\&\, \ldots \,\&\, C_m$, where C_i are clauses. An example of a formula is the expression

$$(x_1 \vee \neg x_2) \,\&\, (\neg x_1 \vee \neg x_5 \vee x_9).$$

The general propositional satisfiability problem is:

- given a formula,
- find the values of the variables that make it true (or, to be more precise, to check whether there exist values x_i that make it true).

The actual proof by reduction. Let us assume that we are given an instance F of the propositional satisfiability problem, i.e., that we are given a propositional formula F of the type $C_1 \,\&\, \ldots \,\&\, C_m$ with v boolean variables $x_1, \ldots, x_v$.

To reduce this instance to an appropriate instance of the ligation prevention problem, first, we assign, to each boolean variable x_j, a fragment $f(x_j)$ consisting of letters C, G, A, and T, in such a way that fragments assigned to two different variables are not complementary.

There are many ways to do it. For example, we can assign v different fragments to v variables, and then add a letter A in front of each of these fragments. This way, no two fragments will fully match, since for them to match, their first symbols must match as well, but A does not match with A – it only matches with T.

To each negation $\neg x_j$, we assign a fragment – which we will denote by $f(\neg x_j)$ – which is complementary to $f(x_j)$, i.e., which is obtained from $f(x_j)$ by replacing A with T, T with A, C with G, and G with C.

Finally, to each clause C_i, we assign a fragment s_i in the following way:

- if the clause has the form $a \vee b$, then we take a fragment $s_i = f(a)f(b)$ consisting of two sub-fragments $f(a)$ and $f(b)$;
- if the clause has the form $a \vee b \vee c$, then we take a fragment $s_i = f(a)f(b)f(c)$ consisting of three sub-fragments $f(a)$, $f(b)$, and $f(c)$.

Let us show that the original formula F is satisfiable if and only if it is possible to select a sub-fragment in each fragment s_i so that none of the selected sub-fragments are complementary to each other.

Indeed, if the formula F is satisfiable, this means that there exists an assignment of truth values to all the boolean variables $x_1, \ldots, x_v$ that makes the formula F true – which means that each of the clauses C_i is true. The fact that a clause C_i is true means that one of its literals is true. We thus select a sub-fragment corresponding to one of the true literals.

No two selected sub-fragments are complementary to each other – indeed, complementary would mean that they represent a variable x_j and its negation $\neg x_j$, and the variable and its negation cannot be both true.

Vice versa, let us assume that we for each fragment s_i corresponding to a clause $C_i = a \vee \ldots$, we selected a sub-fragment – let us denote it by $f(a_i)$ – so that no two sub-fragments are complementary to each other. The fact that they are not complementary means that no two corresponding literals a_i and $a_{j'}$ are negations of each other. Thus, we can assign the truth value to each of the boolean variables x_j as follows:

- if one of the selected sub-fragments has the form $f(x_j)$, then we make the boolean variable x_j true;
- if one of the selected sub-fragments has the form $f(\neg x_j)$, then we make the boolean variable x_j false;

- if none of the selected sub-fragments is of the form $f(x_j)$ or $f(\neg x_j)$, then we assign any truth value to x_j.

Since no two selected sub-fragments have the form $f(x_j)$ and $f(\neg x_j)$, this means that this assignment is consistent. In this assignment, for each clause C_i, the literal a_i corresponding to the selected sub-fragment $f(a_i)$ is true. Thus, under this assignment, each clause C_i is true and hence, the whole formula $F = C_1 \,\&\, \ldots \,\&\, C_m$ is true.

The reduction is proven.

Comment. While we reduced propositional satisfiability to our problem, in fact, this proof can be viewed as reducing another NP-complete problem to our problem – namely, the problem of finding a *clique* of given size k in a given graph. A clique is defined as a subset of the graph's vertices in which every two vertices are connected to each other by an edge. Our proof is actually a modification of the standard proof that the clique problem is NP-complete; see, e.g., [27].

In this proof, we reduce the propositional satisfiability problem to the clique problem in the following way. Let an instance F of the propositional satisfiability problem be given. This instance has the form $C_1 \,\&\, \ldots \,\&\, C_m$, where C_i are clauses. For each literals a from each clause C_i, we add a vertex $V_i(a)$ to the resulting graph.

For example, for the formula $(x_1 \vee \neg x_2) \,\&\, (x_1 \vee x_2 \vee x_3)$, we have a graph with five vertices:

- two vertices $V_1(x_1)$ and $V_1(\neg x_2)$ corresponding to the first clause, and
- three vertices $V_2(x_1)$, $V_2(x_2)$, and $V_2(x_3)$ corresponding to the second clause.

We then connect, by edges, vertices corresponding to different literals provided that they do not correspond to opposite literals x_i and $\neg x_i$. In the above example,

- the vertex $V_1(x_1)$ is connected to $V_2(x_1)$, $V_2(x_2)$, and $V_2(x_3)$; and
- the vertex $V_1(\neg x_2)$ is connected to $V_2(x_1)$ and $V_2(x_3)$ (but *not* to $V_2(x_2)$).

The fact that this is indeed a reduction can be easily proven.

Indeed, if the original formula F is satisfiable, then in each clause, (at least) one of the literals is true. We can select one true literal in each clause. These literals cannot be opposite: since we cannot have x_i and $\neg x_i$ both true. Thus, every two corresponding vertices are connected – i.e., the resulting subgraph indeed forms a clique of size m.

Vice versa, if we have a clique of size m, then, since literals corresponding to the same clause are not connected, this means that vertices from this clique correspond to different clauses. And since we have exactly m clauses, this means that the clique contains exactly one vertex corresponding to each

clause. Now, we can select, for each variable x_i, the value "true" or "false" depending on whether the clique contains a vertex corresponding to x_i or a vertex corresponding to $\neg x_i$. The clique cannot contain both – since vertices corresponding to opposite literals are not connected. (For the variables not reflected in any of the vertices from the clique, we can select any truth value.)

Since each clause C_i contains at least one vertex $V_i(a)$ from the clique, the corresponding literal a is true in this assignment, and thus, the clause C_i is also true. So, under this assignment, all clauses are true – and hence, the original formula $F = C_1 \,\&\, \ldots \,\&\, C_m$ is also true.

The reduction is proven.

5 DNA Computing Without Computing Is Somewhat Less Powerful Than Traditional DNA Computing: A Proof

Which of the two DNA computing schemes is more powerful? In the previous section, we have shown that, in addition to the traditional DNA computing that utilizes the actual DNA-related chemical processes, we can also perform effective computations by using the ability of a body to stop these chemical processes. A natural question is: which of the two DNA computing schemes is more powerful: the active or the passive one?

Overall, they are both NP-complete, in this sense they are both equally powerful. However, we can still talk about which problems are more powerful and which problems are less powerful if we take into account a subtle subdivision of NP-complete problems.

W-hierarchy: a brief reminder. The subtle subdivision that we have in mind – called *W-hierarchy* – is based on the notions of *fixed parameter tractable* (fpt) problems and of *weft*. We will briefly explain these notions in this section; readers interested in detail can check, e.g., [4, 7, 24].

The main idea is that while a problem may be, in general, NP-hard – which means that unless it turns out that P = NP, we cannot have a feasible (polynomial-time) algorithm for solving this problem, there usually is a parameter k such that if we bound the value of this parameter, the problem can be solved in polynomial time, i.e., if, some computable functions $f(k)$ and $C(k)$, a problem with input x of size $n \stackrel{\text{def}}{=} \text{len}(x)$ can be solved in time $f(k) \cdot n^{C(k)}$. This way, if we fix some bound k_0 and only consider problems for which the value of k is bounded by k_0, then all thus limited problems can be solved in time $\le f_0 \cdot n^{C_0}$, where $f_0 \stackrel{\text{def}}{=} \max(f(1), f(2), \ldots, f(k_0))$ and $C_0 \stackrel{\text{def}}{=} \max(C(1), C(2), \ldots, C(k_0))$.

For some problems, the corresponding exponent $C(k)$ does not grow with k. Such problems are called *fixed parameter tractable* (fpt). In precise terms, a problem is fpt if, for some computable function $f(k)$ and for some constant C, a problem with input x of size $n = \text{len}(x)$ can be solved in time $f(k) \cdot n^C$. This way, if we fix some bound k_0 and only consider problems for which the

value of k is bounded by k_0, then all thus limited problems can be solved in time $\leq f_0 \cdot n^C$.

Similarly to the usual reduction, we can define *fpt-reduction* as an reduction that preserves both the size of the inputs (modulo a possible feasible – polynomial-size – increase) *and* preserves the bounds on the parameter, so that problems for which the value of the parameter is bounded by some value k_0 get transformed into problems for which the parameter is bounded by $g(k_0)$ for some feasible function $g(x)$.

The W-hierarchy is based on reduction to computational schemes of a certain *weft*. To describe the weft computation scheme, we first represent this scheme as a directed graph:

- whose vertices are elementary logical (bit) operations and
- where an edge from a vertex a to a vertex b means that the output of a is one of the inputs of the operation b.

For commutative and associative logical operations,

- in addition to the usual binary operations,
- we also allow operations with more than two inputs.

Such operations are "and", "or", and addition modulo 2 (which is the same as "exclusive or").

The weft is defined as the largest number of logical units from an input to the output. For each natural number $i = 0, 1, 2, \ldots$, the i-th class $W[i]$ is defined as the class of the problems that can be reduced to a computation scheme of weft $\leq i$ with several inputs $v_1, \ldots, v_m$ and one output v for which:

- the original problem x with parameter k has a solution if and only if
- there is a combination of inputs $v_1, \ldots, v_m$ that produces the result $v =$"true" and in which at most k inputs v_j are 1s – the rest are 0s.

It can be shown that $W[0]$ is exactly the class FPT of all fpt problems, and one can easily see that $W[0] \subseteq W[1] \subseteq W[2] \subseteq \ldots$

It is not proven that classes $W[i]$ corresponding to different i classes are indeed different, but most computer scientists believe that they *are* different, i.e., that the containment is strict: $W[0] \subset W[1] \subset W[2] \subset \ldots$ Within each class $W[i]$, there are problems which are the hardest in this class – in the sense that every other problem from this class can be fpt-reduced to this problem. Such problems are called $W[i]$*-complete.*

In particular:

- the Hamiltonian path problem – historically the first problem for which an DNA-based solution has been proposed – has been proven to be $W[2]$-complete for k being the graph width (see, e.g., [18]), while, e.g.,

- the clique problem – the problem of finding, in a given graph a clique of a given size k – is known to be $W[1]$-complete; see, e.g., [4, 7].

Since:

- the original DNA computing solves the Hamiltonian path problem while
- the DNA-based computing without computing is equivalent to the clique problem,

we thus arrive at the following conclusion.

Conclusion. The traditional DNA computing is more powerful that DNA computing without computing.

Specifically, while both traditional DNA computing and DNA-based computing without computing solve NP-complete problems:

- the traditional DNA computing is $W[2]$-complete, while
- the DNA-based computing without computing is only $W[1]$-complete, i.e., complete for the somewhat less-complex class of the W-hierarchy.

6 First Related Result: Security Is More Difficult to Achieve than Privacy

What we plan to do in this section. The result from the previous section can be applied to a topic which is not related to DNA computing, but which is very important: the need to maintain privacy and security when using computers.

The reason why such an application is possible is that the main problems of both privacy and security can be reformulated in graph terms.

How to describe privacy in graph terms. Privacy means that:

- while we *should* have access to our own records,
- we *should not* get unauthorized access to any other records.

This means, in particular, that:

- if we perform a simple modification of codewords and other means to get access to our own records,
- we should not be able to gain access to records of anybody else (unless that person gave us a special permission).

To describe this in graph terms, let us form a graph in which individuals are vertices, and two vertices a and b are connected if:

- it is not possible for the individual corresponding to vertex a to access b's record by a simple modification of a's access information; and,

- vice versa, it is not possible for the individual corresponding to vertex b to access a's record by a simple modification of b's access information.

Each abstract access scheme can be represented as such a general graph. The question is: can we use this abstract scheme to provide full privacy for a given number k of users? In terms of the above graph, this is equivalent to finding a subset of k vertices in which every two vertices are connected to each other – i.e., to finding a clique of the given size k.

Thus, in graph terms, maintaining privacy is equivalent to solving the clique problem. We already know that this problem is NP-complete and $W[1]$-complete.

How to describe security in graph terms. In general, computer security (and security in general) means that we have resources so that:

- if we have trouble at some location (physical or virtual),
- one of these resources is available to resolve the corresponding problem.

In the ideal world, we should have such resources at each location. However, realistically, this is usually not possible, so only some locations have resources. In terms of the police example, this means that:

- while we cannot place a police officer at every house, but
- we need to make sure that if a crime is reported, the police from the nearby police station should arrive on time to stop this crime.

Similarly, in computer security, if a suspicious message appears on a computer, the corresponding server should be able to block the corresponding virus from infecting other computers.

This situation can also be described as a graph. Namely, its vertices are possible locations. We connect the two locations a and b if these two locations are "close" in the following sense:

- a resource located at location a can reach location b in time to resolve any possible problem, and
- a resource located at location b can reach location a in time to resolve any possible problem.

Based on the geography and/or on communication ability of the corresponding network, we can form a graph of possible locations, in which edges correspond to the above "closeness". Our overall resources are limited. So, the question is:

- given that we only have k resources,
- is it possible to place them in such a way that every location is the graph is covered – i.e., that each vertex is close to one of the k selected locations?

In graph terms, the corresponding set of k locations is called a *dominating set*. In these terms, the question is: given a graph, is there a dominating set of size k in this graph? It is known that this problem is NP-complete and $W[2]$-complete; see, e.g., [6].

Conclusion: security is more difficult to maintain than privacy. Since:

- security corresponds to a $W[2]$-complete problem, and
- privacy corresponds to a $W[1]$-complete problem – which are, in general, somewhat less complex than $W[2]$-complete problems,

we can therefore conclude that security is somewhat more difficult to maintain than privacy.

7 Second Related Result: Data Storage Is More Difficult Than Data Transmission

Application to information science. A similar result is related to information science, the science of *storing* and *transmitting* information; see, e.g., [2]. This result is very relevant for DNA computing, since this is exactly the main objective of DNA: to store and transmit the biological information.

Data storage. The first type of problems relates to the first objection of information science: storing information. Let us consider situations in which we need to store information about different objects. Let X denote the set of the corresponding objects. In mathematical terms, these objects may be signals, 2D images, 3D bodies, etc.

In many practical cases, storing all possible information about each object requires too much memory space. For example:

- if we want to store the whole information about a human body cell-by-cell,
- we will need to store all the information about billions of cells, the relation between them, etc. – this is not easy to store.

In practice:

- we often do not need the exact information,
- it is usually sufficient to reconstruct it with some reasonable accuracy.

For example, if we want to store a photo, a minor change in intensity will not even be noticeable by a human eye.

To describe this in precise terms, we can form a graph in which:

- vertices are elements of the set X, and

- two objects x and y are connected by an edge if and only if they are practically indistinguishable, i.e., if, for practical purposes, it is OK to reconstruct x if the actual object is y and vice versa.

Usually, indistinguishability is described by a formula $d(x, y) \le \varepsilon$ for an appropriate metric $d(x, y)$ on the set X and an appropriate positive real number $\varepsilon > 0$.

So, instead of storing the actual elements $x \in X$, we only store, for each element x, its approximation s – which should be practically indistinguishable from x. The set S of all such approximation must be such that each element $x \in S$ is practically indistinguishable from some element $s \in S$ – i.e., in graph terms, the set S must be a dominating set in the corresponding graph.

For example, if we want to store a single real number, and we are OK with reconstructing it with accuracy 2^{-n}, then we can restrict ourselves to numbers 0, 2^{-n}, $2 \cdot 2^{-n}$, $3 \cdot 2^{-n}$, etc.

How many bits do we need to store such approximating elements? We need as many bits as are needed to distinguish between different elements of the set S.

- If we use 1 bit, which has 2 possible values 0 and 1 – which can represent 2 different elements.
- If we use 2 bits, with $2^2 = 4$ possible combinations, we can represent 4 different elements.
- With b bits, we can represent 2^b different elements.

So, to represent a set consisting of k elements, we need to have $2^b \ge k$. The smallest such number of bits is $\lceil \log_2(k) \rceil$.

Thus, to find out how many bits of memory we need to represent each element of the original set S, we need to know the binary logarithm of the smallest size of the dominating set. This binary logarithm is known as *ε-entropy*. This notion was first introduced by Kolmogorov and his research group [12, 13, 32]; they also provided asymptotic formulas for the ε-entropy of different function spaces X.

It is known that computing ε-entropy is NP-hard. The above result shows that this problem is $W[2]$-complete.

Data transmission. The data needs to be transmitted. Let us denote by n the overall number of signals that we want to send. We need to assign, to each of these signals s, a physical signal $x(s)$. Examples are:

- the sequence of instantaneous pulses – as when the information is transmitted in a brain; or
- a sequence of shorter and longer pulses, as in the Morse code.

Transmission usually comes with noise. We therefore need to make sure that, even when the transmitted signals are corrupted by noise, we can still distinguish between them. Let us describe this problem in precise terms. Let X denote the set of all physical signals. We can then form a graph in which:

- possible physical signals are vertices, and
- two signals are connected if and only if they can still be distinguishable after applying the noise.

For example, if we know the largest possible change δ caused by a noise – i.e., we know that the distance $d(x, \widetilde{x})$ between the original signal x and the noised signal $\widetilde{x}$ cannot be larger than δ – then the signals x and y can be separated if $d(x, y) > \varepsilon \stackrel{\text{def}}{=} 2\delta$. Indeed, in this case, from the triangle inequality, we can conclude that

$$d(\widetilde{x}, \widetilde{y}) \geq d(x, y) - d(x, \widetilde{x}) - d(y, \widetilde{y}) > 2\delta - \delta - \delta = 0,$$

so $d(\widetilde{x}, \widetilde{y}) > 0$ and thus, $\widetilde{x} \neq \widetilde{y}$. So, corrupted versions of two different signals are always different.

Once we know the set X of possible physical signals, we want to know whether we can use these signals to correctly transmit a given number k of different signals in the presence of noise – and, if yes, what physical signal $x(s)$ we should use to transmit each symbol s from the original messages. In terms of the above-described graph, this means that we need to find a clique of size k in the graph. As we have mentioned, this problem is $W[1]$-hard.

Conclusion: data storage is more difficult than data transmission. Since:

- data storage corresponds to a $W[2]$-complete problem, and
- data transmission corresponds to a $W[1]$-complete problem – which are, in general, somewhat less complex than $W[2]$-complete problems,

we can therefore conclude that data storage is a somewhat more difficult problem than data transmission.

Acknowledgments

This work was supported in part by the US National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science) and HRD-1242122 (Cyber-ShARE Center of Excellence).

The authors are greatly thankful to Evgeny Katz for his encouragement.

References

[1] L. M. Adleman, "Molecular computation of solutions to combinatorial problems", *Science*, 1994, Vol. 266, No. 5187, pp. 1021–1024.

[2] R. Ahlswede, *Storing and Transmitting Data*, Sringer Verlag, Cham, Switzerland, 2014.

[3] M. Amos, *Theoretical and Experimental DNA Computation*, Springer, Berlin, Heidelberg, 2005.

[4] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*, Springer, New York, 2015.

[5] V. Dimitrov, M. Koshelev, and V. Kreinovich, "Acausal processes and astrophysics: case when uncertainty is non-statistical (fuzzy?)", *BUlletin for Studies and Exchanges on Fuzziness and its AppLications (BUSEFAL)*, 1997, Vol. 69, pp. 183–191.

[6] R. G. Downey and M. R. Fellows, "Fixed-parameter tractability and completeness I: Basic results", *SIAM Journal on Computing*, 1995, Vol. 24, pp. 873–921.

[7] R. G. Downey and M. R. Fellows, *Fundamentals of Parameterized Complexity*, Springer, London, 2013.

[8] O. Hosten, M. T. Rakher, J. T. Barreiro, N. A. Peters, and P. G. Kwiat, "Counterfactual quantum computation through quantum interrogation", *Nature*, 2006, Vol. 439, No. 7079, pp. 949–952.

[9] Z. Ignatova, I. Martínez-Pérez, and K.-H. Zimmermann, *DNA Computing Models*, Springer, New York, 2008.

[10] H. T. Jansen, S. Trojahn, M. W. Saxton, C. R. Quackenbush, B. D. Evans Hutzenbiler, O. L. Nelson, O. E. Cornejo, C. T. Robbins, and J. L. Kelley, "Hibernation induces widespread transcriptional remodeling in metabolic tissues of the grizzly bear", *Communications Biology*, 2019, Vol. 2, Article 336.

[11] R. Jozsa, "Quantum effects in algorithms", In: C. P. Williams (ed.), *Quantum Computing and Quantum Communications, Selected Papers from the First NASA International Conference QCQC'98, Palm Springs, California, USA, February 17–20, 1998*, Springer Lecture Notes in Computer Science, Vol. 1509, 1999.

[12] A. N. Kolmogorov, "On certain asymptotic characteristics of completely bounded metric spaces", *Dokl. Akad. Nauk SSSR*, 1956, Vol. 108, No. 3, pp. 385–388 (In Russian).

[13] A. N. Kolmogorov and V. M. Tikhomirov, "ε-entropy and ε-capacity of sets in functional spaces". *Amer. Math. Soc. Transl. Ser. 2*, 1961, Vol. 17, pp. 277–364; Russian original pubished in *Uspekhi Mat. Nauk*, 1959, Vol. 14, No. 2, pp. 3–86.

[14] M. Koshelev and V. Kreinovich, "Towards computers of generation omega – non-equilibrium thermodynamics, granularity, and acausal processes: a brief survey", *Proceedings of the International Conference on Intelligent*

Systems and Semiotics ISAS'97, National Institute of Standards and Technology Publ., Gaithersburg, Maryland, 1997, pp. 383–388.

[15] O. M. Kosheleva and V. Kreinovich. "What can physics give to constructive mathematics," In: *Mathematical Logic and Mathematical Linguistics*, Kalinin, 1981, pp. 117–128 (in Russian).

[16] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1998.

[17] V. Kreinovich and R. Mignani, "Noncausal quantum processes and astrophysics", *Bolletino della Societá Italiana di Fisica*, 1977, Vol. 112, August 29, p. 88.

[18] M. Lampis, G. Kaouri, and V. Mitsou, "On the algorithmic effectiveness of digraph decompositions and complexity measures", *Discrete Optimization*, 2011, Vol. 8, pp. 129–138.

[19] S. Yu. Maslov, *Theory of Deductive Systems and Its Applications*, MIT Press, Cambridge, Massachusetts, 1987.

[20] H. Moravec, *Time travel and computing*, Carnegie-Mellon University, Computer Science Department, Preprint, 1991.

[21] M. S. Morris and K. S. Thorne, "Wormholes in spacetime and their use for interstellar travel: a tool for teaching general relativity", *American Journal of Physics*, 1988, Vol. 56, pp. 395–412.

[22] M. S. Morris, K. S. Thorne, and U. Yurtzever, "Wormholes, time machines, and the weak energy condition", *Physical Review Letters*, 1988, Vol. 61, pp. 1446–1149.

[23] S. Namasudra and G. C. Deka (eds.), *Advances of DNA Computing in Cryptography*, CRC Press, Boca Raton, Florida, 2019.

[24] R. Niedermeier (eds.), *Invitation to Fixed Parameter Algorithms*, Oxford University Press, Oxford, UK, 2006.

[25] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, U.K., 2000.

[26] I. D. Novikov, "Analysis of the operation of a time machine", *Soviet Physics JETP*, 1989, Vol. 68, pp. 439–443.

[27] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, Massachusetts, 1994.

[28] G. Paun, G. Rozenberg, and A. Salomaa, *DNA Computing: New Computing Paradigms*, Soringer Verlag, Berlin, Heidelberg, 2006.

[29] C. Thachuk and Y. Liu, *DNA Computing and Molecular Programming. Proceedings of the 25th International Conference DNA'25, Seattle, Washington, USA, August 5–9, 2019*, Springer Lecture Notes in Computer Science, 2019, Vol. 11648.

[30] K. S. Thorne, "Do the laws of physics permit closed timelike curves?", *Annals of the New York Academy of Sciences*, 1991, Vol. 631, pp. 182–193.

[31] K. S. Thorne, *From Black Holes to Time Warps: Einstein's Outrageous Legacy*, W. W. Norton & Company, New York, 1994.

[32] A. G. Vitushkin, *Theory of Transmission and Processing of Information*, Pergamon Press, Oxford, UK, 1961.

[33] C. P. Williams and S. H. Clearwater, *Ultimate Zero and One: Computing at the Quantum Frontier*, Springer Verlag, New York, 2000.