Complexity issues in computational biology

Jacek Blazewicz Marta Kasprzak*

Institute of Computing Science, Poznan University of Technology,
Piotrowo 2, 60-965 Poznan, Poland
and Institute of Bioorganic Chemistry, Polish Academy of Sciences, Poznan, Poland

Abstract

The progress of research in the area of computational biology, visible in last decades, brought, among others, a new insight into the complexity issues. The latter, previously studied mainly on the ground of computer science or operational research, gained by a confrontation with problems from the new area. In the paper, several complexity issues inspired by computational biology are presented.

Keywords: computational complexity, computational biology, graph theory, nondeterministic Turing machines, DNA computing.

1 Introduction

The progress of research in the area of computational biology, visible in last decades, brought, among others, a new insight into the complexity issues. The latter, previously studied mainly on the ground of computer science or operational research, gained by a confrontation with problems from the new area. Theoretical models for years constructed to solve real-world biological problems appeared to be useful also for other problems and changed a view on their computational complexity. Specificity of data coming from some biological experiments had an influence on the way the computational hardness of related problems could be proved. The interesting concept of DNA computing is also strictly connected with the computational complexity theory as they were acclaimed by some scientists to be a real implementation of the nondeterministic Turing machine. In the following, these issues are discussed.

Before that, let us introduce basic notions of molecular biology used in the paper. Deoxyribonucleic acid, DNA for short, exists in the form of a double helix, i.e. two twisted chains/strands of *nucleotides* joined together. Each nucleotide is composed of a nitrogenous base, a saccharide (deoxyribose) and a phosphoric acid. Nucleotides differ only in their nitrogenous bases. There are four bases: adenine (A), cytosine (C), guanine (G), and thymine (T). Their order in a DNA chain codes a genetic information, symbolically written as a sequence of the letters A, C, G, T. Bases from the two DNA strands are joined by double or triple hydrogen bonds. Against adenine always thymine stands, against cytosine — guanine. This property was named the *complementarity.* The DNA strands are complementary, i.e. knowing a sequence of bases from a fragment of one strand, one can always determine a sequence of bases from the corresponding fragment of the other strand. A length of a DNA chain is expressed in nucleotides or in bases (in base pairs when we have double-stranded DNA). The length of a human genome is about $3 \cdot 10^9$ base pairs. A short fragment of a DNA chain (formerly up to 20 nucleotides, now understood as of the length less than 100) is called an *oligonucleotide.*

Determining a sequence of bases of a DNA fragment — *DNA sequencing* — is the first stage of discovering genetic information. Later, the DNA fragments are combined into the whole chromosome (genome) by assembling and mapping algorithms. The sequencing process is based on the property of complementarity of DNA duplexes. Novel next-generation sequencing approaches produce great volumes of recognized short fragments (reads) at relatively low cost. One of the former approaches, *sequencing by hybridization* (SBH), became the inspiration for defining a series of graph classes (characterized in

*The corresponding author: marta@cs.put.poznan.pl.

the next section), thus we pay here more attention to it. What is more, the algorithmic part of SBH is basically very similar to the current algorithmic approaches solving the DNA assembling problem.

The aim of the *SBH experiment in its standard version* is to detect all oligonucleotides of a given length l (usually 8-12 nucleotides) composing a part of a DNA chain of a known length n (a few hundreds of nucleotides). For this purpose, the oligonucleotide library is generated consisting of all possible single-stranded DNA fragments of length l (i.e. 4^l). Next, the library is exposed to the reaction of hybridization with many copies of the studied DNA. (In order to operate on that great number of molecules, the microarray technology is involved.) During the *hybridization*, complementary subchains of an oligonucleotide from the library and the DNA chain join each other (form a duplex). As the result, one can select oligonucleotides composing the studied DNA, and the oligonucleotides written as words of equal length over the alphabet {A,C,G,T} make a set called *spectrum* (see e.g. [5, 37, 26]). The computational phase of the sequencing process consists in the reconstruction of an original DNA sequence on the basis of the spectrum.

If the hybridization experiment were executed without errors, then the spectrum would be *ideal*, that is it would contain only all substrings of length l of the original sequence of the known length n (see Example 1). However, the experiment usually produces errors in the spectrum. There are two types of *errors*: *negative* ones, i.e. oligonucleotides missing in the spectrum (the words which are parts of the original sequence but are not present in the set), and *positive* ones, which are erroneous oligonucleotides (the words which are not parts of the original sequence but are present in the set). In the literature the errors are alternatively named false negatives and false positives, respectively.

Example 1. Suppose the original sequence to be found is ACTCTGG, $n = 7$. In the hybridization experiment one can use, for example, the complete library of oligonucleotides of length $l = 3$, composed of the following $4^3 = 64$ oligonucleotides: {AAA, AAC, AAG, AAT, ACA, ..., TTG, TTT}. As a result of the experiment performed without errors one obtains the ideal spectrum for this sequence, containing all three-letter substrings of the original sequence: {ACT, CTC, CTG, TCT, TGG}. The reconstruction of the sequence consists of finding such an order of the spectrum elements, where each pair of neighboring elements overlaps on $l - 1 = 2$ letters. $\square$

In the standard approach to SBH, the complete library contains oligonucleotides of the same length but of compositions differing in the ratio of C/G nucleotides to A/T ones. However, DNA duplexes of C/G rich oligonucleotides are more stable than A/T rich ones, and the hybridization conditions should differ for these two groups. The constant temperature kept during the reaction causes a number of hybridization errors in the resulting spectra. In the earliest studies a simple equation was used to calculate melting temperatures of oligonucleotide duplexes, assuming 4 degrees for C/G pairs and 2 degrees for A/T pairs. Although simple, this rule gives a useful estimation (for more advanced rules see e.g. [50]). Therefore, to equalize melting temperatures of all oligonucleotides in the library, the ones composed mainly of A/T nucleotides should be longer than the ones composed mainly of C/G. This fact justifies a new approach to DNA sequencing by hybridization using *isothermic oligonucleotide libraries* [14]. Such library is composed of all possible oligonucleotides of a constant (for the library) melting temperature of oligonucleotide duplexes, but of different lengths. The hybridization with isothermic libraries should result in spectra with lower number of experimental errors, and then the computational phase of the DNA sequencing would be much effective. We refer the interested reader to [49, 44, 51] for a broader treatment of these issues.

The paper is organized as follows. Section 2 describes several classes of graphs with the background in computational biology and their impact on computational complexity of combinatorial problems. In Section 3 it is shown how presence or absence of experimental errors affects computational complexity of biological problems, resulting even in a distinct way of proving their NP-hardness. Similarities and differences between DNA computers and nondeterministic Turing machines are discussed in Section 4. We conclude the paper in Section 5.

2 Biologically inspired graphs and computational complexity

The computational biology, being the research area where molecular biology and computer science meet, by its interdisciplinary nature gave a fresh view on problems of both sides. The biological side was supplied by models and methods from computer science, which allow to solve biological problems efficiently. The computer science side also gained a lot from this junction, among others new classes of graphs useful for modeling problems were defined and analyzed. Their usefulness can be measured on the ground of computational complexity, as they enabled polynomial-time exact algorithms solving several problems being NP-hard in their primary formulations.

Graph theory appeared to have wide application in modeling biological problems, according to combinatorial nature of nucleic acids. Especially the recognition of one-dimensional DNA and RNA structures well fits such kind of model. Small DNA/RNA particles are then represented as vertices in a graph, their overlap is expressed by arcs, and a solution path corresponds to a resulting sequence of nucleotides. Graph models were used in most approaches for DNA sequencing and assembly, starting with first algorithms for the sequencing by hybridization (SBH) by Bains and Smith [5], Drmanac *et al.* [26], and well-known methods of Lysov *et al.* [37] and Pevzner [43]. The models invented for SBH were next widely applied in somehow similar problem of sequence assembly (including the next-generation sequencing).

Before going deeper into DNA sequencing, let us mention an older class of biologically inspired graphs. Historically first connection between graph theory and molecular biology happened in 1959, when Benzer proposed a method for identifying topology of genetic information in a bacteriophage [7]. His experiment led to the detection of its linear structure and, as a byproduct, to the invention of *interval graphs.* Interval graphs represent the relation of overlapping of intervals in linear space. Two vertices, corresponding to intervals, are joined by an edge if the intervals intersect. If a graph belongs to the class of interval graphs, the problem of verifying its Hamiltonicity becomes polynomially solvable [32], what makes this class useful in reducing computational complexity of related problems. Other combinatorial problems, which are in general NP-complete but easy in the case of interval graphs, include vertex coloring, independent set, or clique problem. Since 1959 interval graphs were used in modeling problems from different areas, where a linear arrangement is looked for, for example in computational biology for mapping by hybridization to unique probes.

Coming back to DNA sequencing, the story began with two papers of Lysov *et al.* [37] and Pevzner [43]. The algorithms presented there had been modeled with the use of two fundamental problems of graph theory. The approach by Lysov and co-authors refers to the *Hamiltonian path* problem. In a directed graph, built on the base of an ideal constant-length spectrum, vertices correspond to oligonucleotides and are connected by arcs if the associated oligonucleotides overlap. Formally, vertex v is labeled by oligonucleotide o_v of length l and every pair of vertices u and v, for which the suffix of length $l-1$ of o_u is equal to the prefix of length $l-1$ of o_v, is connected by arc (u, v). In such graph the solution of the problem, i.e. the order of oligonucleotides from the spectrum, corresponds to the Hamiltonian path (i.e. a simple path visiting all vertices of the graph).

The above algorithm accepts as the input data only the ideal spectrum, nevertheless its time complexity is exponential. The first and only polynomial-time algorithm solving the case of errorless, constant-length spectrum was proposed by Pevzner, what proved the search version of this variant of the DNA sequencing is computationally easy. In the algorithm, the *Eulerian path* is looked for in a directed graph (i.e. a path traversing all arcs of the graph exactly once), and now elements of the spectrum are associated with arcs in the graph. An arc, labeled by oligonucleotide o of length l, leaves the vertex labeled by the prefix of length $l-1$ of o and enters the vertex labeled by the suffix of length $l-1$ of o. Thus, the number of vertices is the same as the number of different (l–1)-letter substrings present in the spectrum.

This interesting transformation of the graph, in which the Hamiltonian path is looked for (the strongly NP-hard problem in general), to the graph, in which the Eulerian path is searched, which changed the computational complexity of the way the solution is produced, was not commented by Pevzner. The class of graphs, for which such transformation is possible (*labeled graphs*), was widely examined in [16, 13]. The graphs built on the base of the ideal spectrum according to Lysov's manner, called *DNA graphs*, belong to this class — they are labeled graphs over four-letter alphabet. Notice, that the class of Pevzner's graphs does not belong as a whole to the class of labeled/DNA graphs. This is because there is not 1-1 correspondence between the presence of arcs in such graph and the overlaps of vertex labels (see [16] for details).

Directed *de Bruijn graphs* constitute a class of labeled graphs which are complete with respect to the size of the alphabet and the length of labels [23]. For alphabet of size α and labels of constant length l, de Bruijn graph $B(\alpha, l)$ has α^l vertices, every one labeled by a different word over the alphabet. Arcs are present between all pairs of vertices u and v satisfying the identity of the (l–1)-length suffix of label of u and the (l–1)-length prefix of label of v. Because of their useful properties, these graphs were used in modeling communication networks, VLSI circuits or architecture of parallel computers. DNA graphs are *vertex-induced subgraphs of directed de Bruijn graphs* with $\alpha = 4$. Pevzner's graphs are subgraphs of DNA graphs.

As defined in [8] in the context of directed graphs, the *adjoint* $G = (V, A)$ of a graph $H = (U, V)$ is a 1-graph whose vertices represent arcs of H, and which has an arc from x to y if the terminal endpoint of the arc in H corresponding to x is the initial endpoint of the arc corresponding to y. The *directed line graph* is defined as an adjoint G of a 1-graph H. The Hamiltonian path/circuit problem is easily solvable in an adjoint by transforming the adjoint into its original graph and then by searching for an Eulerian path/circuit within it. The existence of an Eulerian path/circuit in the original directed graph is a necessary and sufficient condition of the existence of a Hamiltonian path/circuit in its adjoint [16].

The graph constructed by the method of Lysov *et al.* is the directed line graph of the graph by Pevzner for the same spectrum. For that pair of graphs, the two problems of looking for the Hamiltonian and Eulerian paths are equivalent. What is interesting, this transformation does not work for undirected graphs [9], unlike the statements present in [33, 27]. See, for example, the undirected graph with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_3, v_4\}\}$, and its line graph; the first one does not possess any Eulerian path while the other has a Hamiltonian path. This is why further work utilizing this transformation was restricted to directed graphs.

The problem of isothermic DNA sequencing by hybridization without errors in the spectrum was solved in [19]. The polynomial-time algorithm presented there used a graph model, in which vertices represented oligonucleotides and arcs were added in a similar way as in DNA graphs, but with special rules concerning vertices with labels being substrings of another ones. (The substrings are possible for two isothermic libraries differing in the melting temperatures by two degrees, which are used together in the isothermic approach.) In order to make such graph a directed line graph, some arcs must be deleted and some "temporary" arcs must be added. After the transformation of this graph into an original graph, a modified algorithm searching for the Eulerian path was applied. This work, which proved that the equivalence of the Hamiltonian and Eulerian paths in the two kinds of graphs holds true also for non-adjoints, became strong motivation for looking for a wider class of graphs with this property.

In [22] such a class was defined. The class of *quasi-adjoint graphs* is a generalization of, among others, adjoints and the graphs modeling the problem of isothermic SBH without errors in experimental data. A directed graph is a quasi-adjoint graph if, for any two vertices x and y, the following property holds:

$$N^+(x) \cap N^+(y) \neq \emptyset \quad \Rightarrow \quad \begin{array}{ll} N^+(x) = N^+(y) & \vee \\ N^+(x) \subset N^+(y) & \vee \\ N^+(y) \subset N^+(x), & \end{array}$$

where $N^+(x)$ is the set of immediate successors of vertex x. For the class of quasi-adjoint graphs a polynomial-time exact algorithm solving the Hamiltonian circuit problem (HCP) was proposed in [22], which also used the transformation between graphs.

One step toward a further extension of this rule was made in [21], where an algorithm for removing some additional superfluous arcs from a graph was examined. The algorithm takes as the input any directed 1-graph, and by solving a series of perfect matchings over its subgraphs reduces arcs, which are guaranteed to not compose any Hamiltonian circuit in the graph. The graphs on the output of the algorithm, named *reduced-by-matching graphs*, in the best case become quasi-adjoint graphs, but mostly they remain "hard" for the problem. However, even hard, the reduced graph becomes an easier instance for some exact or heuristic algorithm solving HCP. It was confirmed in tests, which also showed, that the algorithm is especially efficient for graphs with average outdegree equal to 2 (HCP is NP-hard for such graphs). Those graphs after the reduction had the mean outdegree close to 1, what made them easy (on average) for HCP algorithms.

In [21] also a systematization of several classes of digraphs referring to directed line graphs was provided, together with the proof of its correctness. In Fig. 1 this relationship is shown. For most of the classes, polynomial-time algorithms solving HCP exist.

Another field of computational biology, the DNA sequence assembly, also intensively uses models and methods of graph theory. The sequence assembly is the next step, after the sequencing, in the reconstruction of a genomic fragment. The output of the sequencing process (e.g. SBH) becomes the input for the sequence assembly and the latter is somehow similar to the former but on a much larger scale. Now the input sequences generally have different lengths, misreadings (insertions, deletions, and

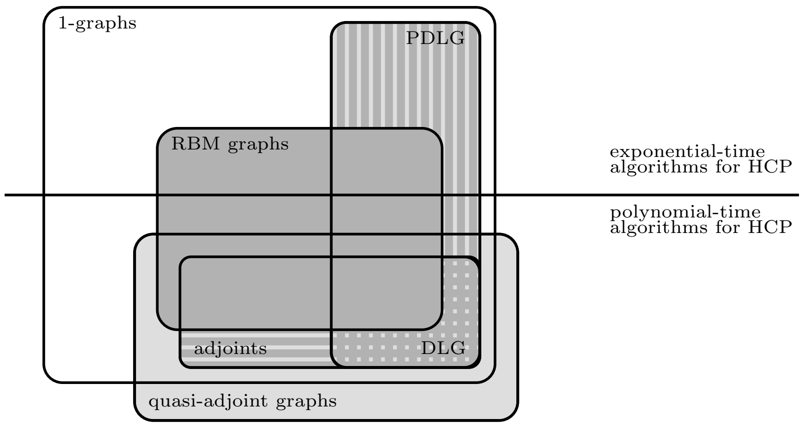

1-graphs
PDLG
RBM graphs
exponential-time algorithms for HCP
polynomial-time algorithms for HCP
adjoints
DLG
quasi-adjoint graphs

Figure 1: [21] The relationship between a few digraph classes with reference to HCP solvability. DLG stands for directed line graphs, PDLG for partial directed line graphs [4], and RBM graphs for reduced-by-matching graphs. DNA graphs lie within DLG area.

substitutions of nucleotides), and come from any of the two strands of a fragment of a DNA helix (with no hint from which one). Huge amount of erroneous and incomplete data make the problem well known for its high complexity. It is strongly NP-hard even in the case of data without errors and derived from one DNA strand (compare with Shortest Common Superstring [27]).

The goal of the assembly is to compose input fragments into one resulting sequence (or a series of disjoint sequences — contigs) in a proper order. This problem can be modeled as a graph theoretic problem, the searching for a Hamiltonian path in a certain digraph (e.g. [31, 11]). Unlike in the sequencing, now due to inexact matchings allowed between fragments such graphs do not possess the useful property of being adjoints nor quasi-adjoint graphs. The inexact matchings are the result of sequencing errors (insertions, deletions, substitutions of nucleotides) and researchers sought ways to overcome this disadvantage. One successful idea, applied next in many assembly algorithms (e.g. [29, 45, 52]), was to decompose the input sequences into series of shorter overlapping constant-length oligonucleotides and to build the graph with only exact matchings allowed, like in the Pevzner's approach. Erroneous information present in the input data was then either corrected by some heuristic procedures or ignored. Finally, the Eulerian path was searched for. The assembly graphs constructed with the decomposition of input sequences are — as the Pevzner's graphs — subgraphs (but not vertex-induced) of de Bruijn graphs over four-letter alphabet. Novel next-generation sequencing approaches, like 454 sequencing, Illumina's one or SOLiD, well suit to this model thanks to a small guaranteed experimental error rate.

With the progress in sequencing technology, the form of the data change and theoretical models have to adjust accordingly. Paired-end sequencing protocol has been theoretically and experimentally proved to be much more efficient for de novo sequencing than standard single-read approach. In the paired-end sequencing, instead of a set of single fragments at the input, one gets a set of paired fragments placed close to each other in the original genome (within some predefined approximate distance). Theoretical models proposed for such kind of data usually skipped this additional information at the beginning, built the assembly graphs as defined above and used the pairs just to guide the process of searching for a path. They did not lead to a simplification of the graph.

One of the latest papers brought a new concept of so-called *paired de Bruijn graphs*, a modification of the assembly graphs incorporating the information about paired fragments [38]. (Recall that these graphs, similarly as decomposition-based assembly graphs, are not in fact de Bruijn graphs due to the incompleteness of both the vertex and arc sets.) First, the input pairs of fragments are decomposed into pairs of oligonucleotides: "left" oligonucleotides in the pairs come from the decomposition of the "left" fragments, the corresponding "right" oligonucleotide lies at the same position as the "left" one but within the corresponding "right" fragment. In the graph, arcs correspond to the pairs of oligonucleotides and vertices to pairs of their prefixes and suffixes. Such graphs, thanks to the more restricted rule of joining arcs, are indeed less tangled than the regular decomposition-based assembly graphs.

As we see, there is a number of graph classes inspired by biological problems. What is more, these graphs are not only nice theoretical models of real-world problems, but — first of all — they led to polynomial-time solutions of these problems. The class most known beyond the area of computational biology (and the oldest) is the one of interval graphs. The new class of quasi-adjoint graphs also may become significant for non-biological community, since it enables polynomial-time exact solution of the Hamiltonian circuit problem, commonly used in modeling real-world problems.

3 Errorless vs. erroneous instances

Most of the graph models presented in the previous section are related to these problems of computational biology, in which no errors in the experimental data are assumed. In the case of erroneous data, either the construction of these graphs is not possible, or the accompanying polynomial-time algorithm solving the problem no longer works. For example, the mapping by hybridization to unique probes with uncomplete or incorrect data may result in a non-interval graph. For the DNA sequencing and errors, both Lysov's graph and Pevzner's graph can be built and the transformation between them is still possible (we mean of course the transformation between a directed line graph and its original graph). However, now these graphs do not contain (in general) any Hamiltonian/Eulerian path.

In practice almost all results of biological experiments contain errors. Unfortunately, the corresponding combinatorial problems usually cannot be solved easily. Let us look at the subset of problems for recognizing one-dimensional structure of a DNA. DNA sequencing by hybridization with standard oligonucleotide libraries is polynomially solvable only in the case without any errors [43]. The problem (in its search version) is strongly NP-hard when errors are allowed in instances: either when restricted to only negative errors or only positive ones [17]. Even variants of the problem with an additional knowledge: the promise of uniqueness of the solution for an instance, are computationally hard [17]. The problem with negative errors restricted to only errors coming from repetitions of oligonucleotides within a DNA sequence (without other missing oligonucleotides in a spectrum corresponding to experimental errors) is open [15].

Similar situation we have for SBH with isothermic oligonucleotide libraries. The problem without errors is polynomially solvable while the (search) variants with only negative errors or only positive ones are both strongly NP-hard [19]. The next stage in recognizing genomes is the sequence assembly. This problem is strongly NP-hard even as a very restricted variant, where there are no errors and all reads come from the same DNA strand (compare with Shortest Common Superstring [27]). A real-world variant of the sequence assembly with all the experimental errors allowed is much more algorithmically complicated. The assembled sequences are next ordered at the stage of DNA mapping. The mapping can be realized by the hybridization to unique probes or with the use of restriction enzymes. The mapping by hybridization is easy in the case of no errors and strongly NP-hard otherwise, the latter has been solved by a transformation to Traveling Salesman Problem with Hamming distances calculated for probes [3]. The restriction mapping can be based on different variants of a biological experiment and formulated as the problems: Double Digest Problem (DDP), Partial Digest Problem (PDP) and Simplified Partial Digest Problem (SPDP). Both DDP and SPDP have been proved to be strongly NP-hard in case of no errors (in search versions) [12, 18]. PDP (without errors) is a well-known open problem till now, also in the field of computational geometry where it has its counterpart. However, for PDP in the presence of experimental errors the NP-hardness proofs were published [25, 24].

The computational biology problems in the variants of instances without any experimental errors can be polynomially solvable (in their search versions), but, as exemplified above, they may be not as well. What is interesting, the problems from this area need a special way of proving their computational hardness when an error model is restricted. We cannot do it in a standard way, by a transformation from the decision version of a known intractable problem to the decision version of our problem, if the decision counterpart of the biological problem is polynomially solvable.

Let us take as the example the mapping problem SPDP. Disregarding here details of the corresponding biological experiment, the problem consists in reconstructing an original order of input sequences (basing only upon their lengths) within a studied fragment of a genome. In fact, because we know that the biological particles really exist (the genomic fragment and the shorter sequences), together with the additional knowledge that the experiment has been carried out without any experimental error we are sure, that the resulting map always can be reconstructed. One of possible solutions for the problem is yet present in real world. Therefore, the answer for the decision version of the problem is always "yes".

However, the process of finding this map is not easy from the computational complexity point of view [18]. The same also holds for many other problems of computational biology, see e.g. [12, 17, 19], but for standard or isothermic SBH problems it is the case when errors of one kind are excluded. Decision versions of the problems where any errors are allowed may have the answer "no", because then any non-realistic instances are possible.

This was the reason for studying NP-hardness of search versions of several problems. The process of proving this fact gone through an artificial decision problem, allowing also instances with the answer "no", but the set of instances with positive answers having identical as in the original biological problem. From its strong NP-completeness it followed the strong NP-hardness of the original problem. A similar reasoning was formed in [30] for the Hamiltonian Circuit Problem. As David Johnson stated there, even if we knew a graph contains a Hamiltonian circuit (in this case the decision version is, of course, easy), we could not find it in polynomial time unless P=NP.

It must be stressed, that the way of proving computational hardness of computational biology problems depends strongly on their mathematical formulation. The special way of proceeding the proofs as outlined above followed from the easiness of their decision versions. In case of other problems the approach should be appropriately selected.

4 Nondeterministic Turing machines and DNA computing

For many years nondeterministic models of computations, in particular *nondeterministic Turing machines*, were used solely for problem classification as a purely theoretical tool [2]. Recently, the concept of *DNA computing* [1] brought to our attention a potential possibility of the application of this concept in practice.

In this section, we will briefly analyze the relation between the two concepts, pointing out their differences which limit practical applicability of the latter. Taking into account the background of potential readers we will only briefly recall basic concepts of the computational complexity theory (referring to excellent monographs [2, 27, 41]), explaining in a greater detail biological issues necessary for problem description.

We start with a recollection of Turing machines being universal models of computations. *Deterministic Turing machine* (DTM) is usually depicted as a "device" having a *control*, a *tape* with an unlimited number of *cells*, and a *head* reading and writing symbols in the cells, one at a time. More formally, DTM is defined as a triple $M = (\mathcal{Q}, \Gamma, \delta)$, where $\mathcal{Q}$ is a finite set of *states* (containing three distinguished states: starting state q_0 and two final states q_y and q_n with "yes" and "no" answer, respectively), Γ is a finite *alphabet* of symbols used by DTM, and δ — a transition function which decides about the moves and symbols to be written in the tape cells:

$$\delta : (\mathcal{Q} - \{q_y, q_n\}) \times \Gamma \to \mathcal{Q} \times \Gamma \times \{-1, +1\}.$$

This simple model reads one symbol from the tape and depending on it and the state of its control, writes an appropriate symbol in the cell under the head, moves the head to left (-1) or right ($+1$) and changes the state according to function δ. Despite its simplicity this model is able to simulate any other realistic computing model, or in other words, an arbitrary program (algorithm) written in any reasonable programming language. We see that the definition of DTM captures both: the computing model (the hardware: control, tape, head) and the program to be performed on it (the software: $\mathcal{Q}, \Gamma, \delta$). DTM is supposed to stop its computations either in state q_y — answer "yes" — or in state q_n — answer "no", thus, it is especially useful when solving *decision problems* or recognizing *formal languages* [2, 10, 27, 41].

Two important parameters characterizing any algorithm, i.e. any corresponding DTM, are its time and memory (space) complexity functions. *Time complexity function of algorithm M* (or corresponding DTM) solving decision problem Π is a function mapping any problem instance size $|x|$ (x being encoded string of the instance of Π) onto the maximum number of DTM steps required to solve an instance of this size.

Similarly, *memory (space) complexity function of algorithm M* (or corresponding DTM) solving decision problem Π is a function mapping any problem instance size $|x|$ onto the maximum number of DTM cells used for solving an instance of this size. (Of course, it is equal to the length of the output string written in the cells of the tape.)

In both cases two main classes of complexity functions can be distinguished: *polynomial* (complexity function can be bounded from above by a polynomial in the input size) and *exponential* (no such bound exists). Here, few remarks are in order.

For practical purposes, only algorithms with polynomial time and memory complexity functions can be used. It is rather easy to observe that for a given algorithm (DTM) M, its memory complexity function cannot be greater than its time complexity function. What is more, it can be proved that most of all practically formulated problems can be solved in polynomial space (i.e. requiring polynomially bounded memory), thus, they all belong to the class PSPACE of problems (cf. e.g. [41]). Hence, in practice only time constraints are really important. Decision problems, which can be solved in polynomial time belong to class P. To characterize other (more complex) problems one needs a more sophisticated computing model, the nondeterministic Turing machine.

The *nondeterministic Turing machine* (NDTM) can be defined in many ways. We use the one proposed in [27] since it is the closest to the DNA computing paradigm. In this presentation NDTM is simply a DTM with an addition of a generating module. The other components of the model, i.e. $\mathcal{Q}, \Gamma$, and δ remain intact. However, the action of NDTM is composed of two phases. In the first phase, the generating module generates a random, finite string of symbols from Γ. In the second phase, the deterministic part of NDTM verifies whether or not this string is a solution with an answer "yes" for an input instance x. Such a verification takes place in parallel for all finite generated strings. The answer for x is "yes" if at least one verification for any string will end up with such an answer. Otherwise, the answer is "no". (We see that, if such a model could be realized in practice, it would involve infinitely many copies of DTM, each verifying a different string.)

As a time of nondeterministic computations for a given input instance x of problem Π, one takes a minimum number of steps of two phases (generation and verification) for some string, for which computations ended up with a "yes" answer. If for x no such string exists, then this time is undefined.

Time complexity function of a nondeterministic algorithm M (or equivalently NDTM) solving decision problem Π is a function mapping any problem instance size $|x|$ on the maximum time as defined above. If for size $|x|$ no instance with a "yes" answer exists, then 1 is chosen instead. Similarly, memory used by a nondeterministic algorithm (NDTM) M for a given instance x of problem Π is calculated as a minimum memory requirement over all accepted strings, while *memory complexity function of this algorithm* is a function mapping any problem instance size $|x|$ onto the maximum memory requirement over all instances of this size.

Again, polynomial and exponential nondeterministic algorithms can be distinguished for both measures: time and memory (space), respectively. Decision problems which can be solved in polynomial time nondeterministically compose class NP. It is clear that P $\subseteq$ NP, however, "P $\subset$ NP?" is a long standing open problem in theoretical computer science. On the other hand, it seems improbable that P=NP and the most difficult problems in NP, the NP-complete problems, are considered to be unsolvable deterministically in polynomial time (i.e. in practice).

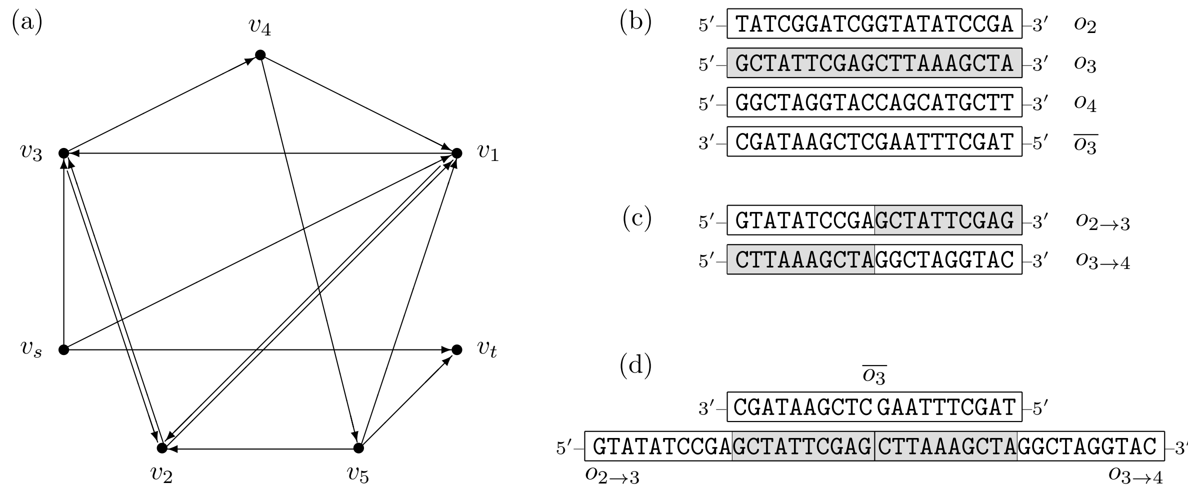
In case of memory complexity, decision problems that can be solved in polynomial space nondeterministically compose class NPSPACE. Following the result of Savitch [47], we have PSPACE = NPSPACE. Taking into account previous results we see:

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NPSPACE}.$$

Thus, again time seems to be a crucial resource in all complex computations.

Two remarks are in order. As we said, there are other representations of NDTM. First, assume that instead of function δ, NDTM has relation Δ (cf. [41]). Or, add to the standard DTM an additional instruction *fork* resulting in a creation of an additional copy of DTM solving the same instance of problem Π [48]. All the models are equivalent with respect to time and memory complexity function and polynomial factor depending on $|x|$. The second remark is concerned with a practical implementation of NDTM. For many decades the model was considered as a purely theoretical one, having no practical meaning. However, in last years, a discovery of DNA computing paradigm put a question mark on the above statement. We will analyze this question in a more detail below.

The idea of *DNA computing* was materialized in the experiment of Leonard Adleman [1]. In fact, he designed a new paradigm of computing, defining the "DNA computer" with only biological elementary operations, manipulating DNA chains. Using a discrete nature of DNA (four letters A, C, G, and T coding respective bases, cf. Section 1) and the basic properties of the DNA molecule, Adleman was able



(a)
v_4
v_3
v_1
v_s
v_t
v_2
v_5
(b)
5′– TATCGGATCGGTATATCCGA –3′ o_2
5′– GCTATTCGAGCTTAAAGCTA –3′ o_3
5′– GGCTAGGTACCAGCATGCTT –3′ o_4
3′– CGATAAGCTCGAATTTCGAT –5′ $\overline{o_3}$
(c)
5′– GTATATCCGAGCTATTCGAG –3′ $o_{2\to3}$
5′– CTTAAAGCTAGGCTAGGTAC –3′ $o_{3\to4}$
(d)
$\overline{o_3}$
3′– CGATAAGCTC GAATTTCGAT –5′
5′– GTATATCCGAGCTATTCGAG CTTAAAGCTAGGCTAGGTAC –3′
$o_{2\to3}$
$o_{3\to4}$

Figure 2: (a) The graph of Adleman's approach. The unique Hamiltonian path is the sequence of vertices (v_s, v_1, v_2, v_3, v_4, v_5, v_t). (b–d) Exemplarily chosen DNA sequences used by Adleman to encode parts of the graph. Note, that $5'$ and $3'$ stand for chemical structures present at the ends of a single-stranded DNA and determine the orientation the DNA fragments are read (from $5'$-end to $3'$-end). (b) 20-mers of DNA encoding vertices v_2, v_3, v_4, together with the complementary 20-mer for v_3. (c) The labels of arcs (v_2, v_3) and (v_3, v_4). (d) An exemplary join of 20-mers representing two arcs and one vertex (here the complementary oligonucleotide), resulting in a fragment of a double-stranded DNA.

to design a wet lab experiment solving an instance of the Hamiltonian path problem with known start and end vertices (its decision version). What is even more astonishing, the number of elementary (although biological in nature) operations was polynomial in the input length. (See also [6] for a solution of the Hamiltonian path problem on a "bacterial computer".)

The considered *NP-complete* version of the problem was defined as follows: *For a given directed graph $G = (V, A)$ with two distinguished vertices v_s and v_t, does there exist a Hamiltonian path starting in v_s and finishing in v_t?* The exemplary graph used in Adleman's experiment is shown in Fig. 2 (a).

The computational idea Adleman used when implementing his DNA computer was in fact the NDTM model just described above. In particular, he used the following nondeterministic algorithm.

Step 1. Generate random paths through the graph.
Step 2. Keep only those paths, which begin with v_s and end with v_t.
Step 3. Keep only those paths, which enter exactly $|V|$ vertices.
Step 4. Keep only those paths, which enter every vertex of the graph at least once.
Step 5. If any paths remain, say "yes", otherwise say "no".

Now, we briefly overview the implementation details of this pioneering approach [1]. Step 1 is based in fact on the reverse idea of the sequencing by hybridization approach, cf. Section 2. Here, each vertex v_i a random label o_i was assigned, composed of 20 nucleotides (20-mer of single-stranded DNA, i.e. oligonucleotide of length 20). Each arc (v_i, v_j) got a corresponding 20-mer $o_{i\to j}$ composed of the 10-mer suffix of o_i followed by the 10-mer prefix of o_j. The labels of arcs leaving v_s or entering v_t were longer, in that case the whole o_s or o_t was taken instead of its suffix/prefix, respectively (these labels were 30-mers of DNA). To complement the whole encoding, complementary 20-mers $\overline{o_i}$ were defined for each vertex label o_i. An exemplary encoding of a fragment of the graph given in Fig. 2 (a) is presented in Fig. 2 (b–c).

All these DNA sequences, except oligonucleotides o_i, were physically generated (in quantities of 50 pmol each) and mixed together in a ligation reaction. The ligation reaction links together DNA strands that have a break in their sugar-phosphate backbone, by forming bonds between $3'$ and $5'$ ends of fragments. Together with the hybridization between complementary subsequences of oligonucleotides

Table 1: Comparison of the central characteristics of conventional electronic computers and DNA computers [39].

	Conventional computer	DNA computer
Storage (space needed for one bit 0 or 1)	10^{12} nm^3	1 nm^3
Speed (millions of instructions per second)	10^3	10^{14}
Energy (number of operations per joule)	10^9	$2 \cdot 10^{19}$

representing vertices and their incident edges, the DNA particles bound and built double-stranded DNA chains. These chains corresponded to all possible paths in the graph (a fragment of such a path is illustrated in Fig. 2 (d)).

In Step 2 of the algorithm, a polymerase chain reaction (PCR) was used in order to amplify only these DNA chains, which had correct start and end sequences. To do this, as PCR primers the oligonucleotides o_s and $\overline{o_t}$ were used.

In Step 3 only the chains of length 140 base pairs were selected (seven 20-mers for seven vertices in the graph). Here gel electrophoresis was performed, the chains with 140 base pairs were extracted and then again multiplied by the PCR technique.

Step 4 yielded a purification, which filtered out the DNA particles containing certain sequences. This process was done iteratively for single stranded DNA chains being a result of the previous step. In every step of the iteration, a biotin-avidin magnetic bead system was used to pick up the remaining chains, each time with different $\overline{o_i}$ conjugated to the beads. At the end only those single-stranded 140-nucleotide-long DNA chains remained, if any, which contained all oligonucleotides o_s, o_1, o_2, ..., o_t as their parts.

If any chains were left after Step 4 (in the Adleman's experiment they were), then the answer is "yes" to the decision version of the Hamiltonian path problem and the implemented instance. One could even construct this Hamiltonian path by sequencing one of the resulting DNA chains.

To sum up the Adleman's approach, let us make bit more general remarks. This new computing model has the following elementary operations: DNA synthesizing (constructing DNA labels), ligation, PCR amplification, separating by length (gel electrophoresis), and separating by sequence (purification by magnetic beads). The time of every operation can be bounded from above by a polynomial in the number of vertices in the graph. Thus, the time complexity function of solving the Hamiltonian path problem by the means of DNA computing is polynomial and the model achieves a massive parallelism of the nondeterministic Turing machine! What is more, other characteristics of DNA computing paradigm also prevail over classical electronic computers (cf. Table 1) [39].

Taking into account the above features of DNA computing no wonder that several other papers were devoted to the solution of classical NP-complete problems. Let us mention here the approach by Lipton, which solves a generalization of the satisfiability (SAT) problem [35], or another DNA-based algorithm solving the maximal clique problem [40]. There are also DNA algorithms for solving the graph coloring problem [36], the traveling salesman problem [34], and the Chinese postman problem [28]. To this end let us also mention the original approach to the network flow problem presented in [46].

Having said that, one must also analyze the drawbacks of the presented approach. First of all, the model as defined by Adleman is very specialized, not a universal one. In fact, separate DNA computers would be necessary to solve other combinatorial problems. Recently a substantial progress has been made and slowly these problems could be overcome. However, the main drawback of this approach lies in somewhat surprising area, namely its memory (space) requirement. Let us recall, that NDTM (also DTM) could solve all the problems from NP in polynomial space. This is not the case of the DNA computer. Look at Step 1 of the proposed Adleman's algorithm. Here, *all* paths in the graph had to be generated physically. It is easy to verify, that their number grows exponentially with the number of vertices in the graph. As M. Amos pointed out: "It has been estimated, that if you scaled up the Hamilton Path Problem to 200 cities from Adleman's seven, then the weight of DNA required to represent all the possible solutions would exceed the weight of the earth." [42].

Another problem lies in experimental errors. All the biochemical reactions are performed on a huge number of copies produced for every DNA sequence used in the experiment, in order to statistically "guarantee" the receiving of required results. For a single DNA particle, a reaction may not occur at all (despite its chemical predisposition) or may occur with several errors (e.g. two non-complementary single-stranded DNA chains join each other). In a specific case, as a result of a DNA algorithm we could obtain the wrong answer. Also we must remember, that a number of experimental errors grows with the amount of the material used, moreover, erroneous products of one step can be amplified and moved to the next one.

We may now compare the two paradigms: NDTM and that of DNA computing. While their time complexities behave quite similarly, space complexities differ very much. The DNA computing model, as a real working computer, achieves massive parallelism at the expense of excessive usage of really working computing units (in Adleman's example analyzed DNA sequences). Thus, the memory (space) complexity of this approach is enormous. On the other hand, NDTM definition of the space used is very artificial, reducing it to only one (the shortest) computation. Thus, this model remains of theoretical nature, as discussed earlier.

5 Conclusion

In the paper several complexity issues, emerging from research in the area of computational biology, have been presented. New models and algorithms developed for biological problems brought a new insight into classical computational complexity theory. On the other hand, molecular biology itself with its DNA computing approach entered the space reserved till now for theoretical computer science. The connection between both disciplines is alive and probably will provide new intriguing results also in the future.

6 Acknowledgements

The research has been partially supported by grant No. DEC-2011/01/B/ST6/07021 from the National Science Centre, Poland. The authors acknowledge also useful help of Andrea Sackmann in the preparation of Section 4.

References

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

[2] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading, Massachusetts (1974).

[3] F. Alizadeh, R.M. Karp, L.A. Newberg, D.K. Weisser, Physical mapping of chromosomes: a combinatorial problem in molecular biology, *Algorithmica* 13 (1995) 52–76.

[4] N. Apollonio and P.G. Franciosa, A characterization of partial directed line graphs, *Discrete Mathematics* 307 (2007) 2598–2614.

[5] W. Bains and G.C. Smith, A novel method for nucleic acid sequence determination, *Journal of Theoretical Biology* 135 (1988) 303–307.

[6] J. Baumgardner *et al.*, Solving a Hamiltonian Path Problem with a bacterial computer, *Journal of Biological Engineering* 3:11 (2009).

[7] S. Benzer, On the topology of the genetic fine structure, *Proceedings of the National Academy of Sciences of USA* 45 (1959) 1607-1620.

[8] C. Berge, *Graphs and Hypergraphs*, North-Holland Publishing Company, London (1973).

[9] A.A. Bertossi, The edge Hamiltonian path problem is NP-complete, *Information Processing Letters* 13 (1981) 157-159.

[10] J. Blazewicz, *Computational Complexity of Combinatorial Problems*, WNT, Warszawa (1988) (in Polish).

[11] J. Blazewicz, M. Bryja, M. Figlerowicz, P. Gawron, M. Kasprzak, E. Kirton, D. Platt, J. Przybytek, A. Swiercz, and L. Szajkowski, Whole genome assembly from 454 sequencing output via modified DNA graph concept, *Computational Biology and Chemistry* 33 (2009) 224–230.

[12] J. Blazewicz, E. Burke, M. Jaroszewski, M. Kasprzak, B. Paliswiat, and P. Pryputniewicz, On the complexity of the Double Digest Problem, *Control and Cybernetics* 33 (2004) 133–140.

[13] J. Blazewicz, P. Formanowicz, M. Kasprzak, and D. Kobler, On the recognition of de Bruijn graphs and their induced subgraphs, *Discrete Mathematics* 245 (2002) 81–92.

[14] J. Blazewicz, P. Formanowicz, M. Kasprzak, and W.T. Markiewicz, Sequencing by hybridization with isothermic oligonucleotide libraries, *Discrete Applied Mathematics* 145 (2004) 40–51.

[15] J. Blazewicz, P. Formanowicz, M. Kasprzak, P. Schuurman, and G.J. Woeginger, A polynomial time equivalence between DNA sequencing and the exact perfect matching problem, *Discrete Optimization* 4 (2007) 154-162.

[16] J. Blazewicz, A. Hertz, D. Kobler, and D. de Werra, On some properties of DNA graphs, *Discrete Applied Mathematics* 98 (1999) 1–19.

[17] J. Blazewicz and M. Kasprzak, Complexity of DNA sequencing by hybridization, *Theoretical Computer Science* 290 (2003) 1459–1473.

[18] J. Blazewicz and M. Kasprzak, Combinatorial optimization in DNA mapping — a computational thread of the Simplified Partial Digest Problem, *RAIRO — Operations Research* 39 (2005) 227–241.

[19] J. Blazewicz and M. Kasprzak, Computational complexity of isothermic DNA sequencing by hybridization, *Discrete Applied Mathematics* 154 (2006) 718–729.

[20] J. Blazewicz and M. Kasprzak, Graph reduction and its application to DNA sequence assembly, *Bulletin of the Polish Academy of Sciences. Technical Sciences* 56 (2008) 65–70.

[21] J. Blazewicz and M. Kasprzak, Reduced-by-matching graphs: toward simplifying Hamiltonian circuit problem, *Fundamenta Informaticae*, in press.

[22] J. Blazewicz, M. Kasprzak, B. Leroy-Beaulieu, and D. de Werra, Finding Hamiltonian circuits in quasi-adjoint graphs, *Discrete Applied Mathematics* 156 (2008) 2573–2580.

[23] N.G. de Bruijn, A combinatorial problem, *Koninklijke Nederlandse Akademie v. Wetenschappen Proceedings* 49 (1946) 758–764.

[24] M. Cieliebak and S. Eidenbenz, Measurement errors make the Partial Digest Problem NP-hard, *Lecture Notes in Computer Science* 2976 (2004) 379–390.

[25] M. Cieliebak, S. Eidenbenz, and P. Penna, Noisy data make the Partial Digest Problem NP-hard, *Lecture Notes in Bioinformatics* 2812 (2003) 111–123.

[26] R. Drmanac, I. Labat, I. Brukner, and R. Crkvenjakov, Sequencing of megabase plus DNA by hybridization: theory of the method, *Genomics* 4 (1989) 114–128.

[27] M.R. Garey and D.S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco (1979).

[28] A. Han and D. Zhu, DNA encoding method of weight for Chinese postman problem, *Proceedings of the IEEE Congress on Evolutionary Computation*, IEEE Press, Vancouver (2006) 681–686.

[29] R. Idury and M. Waterman, A new algorithm for DNA sequence assembly, *Journal of Computational Biology* 2 (1995) 291-306.

[30] D.S. Johnson, The NP-completeness column: an ongoing guide, *Journal of Algorithms* 6 (1985) 291-305.

[31] J.D. Kececioglu and E.W. Myers, Combinatorial algorithms for DNA sequence assembly, *Algorithmica* 13 (1995) 7-51.

[32] J.M. Keil, Finding Hamiltonian circuits in interval graphs, *Information Processing Letters* 20 (1985) 201-206.

[33] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York (1976).

[34] J.Y. Lee, S.Y. Shin, T.H. Park, and B.T. Zhang, Solving traveling salesman problems with DNA molecules encoding numerical values, *Biosystems* 78 (2004) 39–47.

[35] R.J. Lipton, DNA solution of hard computational problems, *Science* 268 (1995) 542–545.

[36] Y. Liu, J. Xu, L. Pan, and S. Wang, DNA solution of graph coloring problem, *Journal of Chemical Information and Computer Sciences* 42 (2002) 524–528.

[37] Yu.P. Lysov, V.L. Florentiev, A.A. Khorlin, K.R. Khrapko, V.V. Shik, and A.D. Mirzabekov, Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides. A new method, *Doklady Akademii Nauk SSSR* 303 (1988) 1508–1511.

[38] P. Medvedev, S. Pham, M. Chaisson, G. Tesler, and P. Pevzner, Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers, *Lecture Notes in Computer Science* 6577 (2011) 238–251.

[39] N. Nagy and S.G. Akl, Aspects of biomolecular computing, *Parallel Processing Letters* 17 (2007) 185–211.

[40] Q. Ouyang, P.D. Kaplan, S. Liu, and A. Libchaber, DNA solution of the maximal clique problem, *Science* 278 (1997) 446–449.

[41] C. Papadimitriou, *Computational Complexity*, Addison Wesley, Reading, Massachusetts (1994).

[42] J. Parker, Computing with DNA, *EMBO reports* 4 (2003) 7–10.

[43] P.A. Pevzner, l-tuple DNA sequencing: computer analysis, *Journal of Biomolecular Structure and Dynamics* 7 (1989) 63–73.

[44] P.A. Pevzner, *Computational Molecular Biology: an Algorithmic Approach*, MIT Press, Cambridge (2000).

[45] P.A. Pevzner, H. Tang, and M.S. Waterman, An Eulerian path approach to DNA fragment assembly, *Proceedings of the National Academy of Sciences of the USA* 98 (2001) 9748–9753.

[46] A. Sackmann, *Network Algorithms and Bioinformatics Problems*, Ph.D.Thesis, Poznan University of Technology (2008).

[47] W.J. Savitch, Relationship between nondeterministic and deterministic tape classes, *Journal of Computer and System Sciences* 4 (1970) 177–192.

[48] J.D. Ullman, Complexity of sequencing problems, Chapter 4 *in* E.G. Coffman, Jr. (ed.) *Computer and JobShop Scheduling Theory*, John Wiley & Sons, New York (1976).

[49] M.S. Waterman, *Introduction to Computational Biology. Maps, Sequences, and Genomes*, Chapman & Hall, London (1995).

[50] T. Xia T, J. SantaLucia Jr., M.E. Burkard, R. Kierzek, S.J. Schroeder, X. Jiao, C. Cox, and D.H. Turner, Thermodynamic parameters for an expanded nearest-neighbor model for formation of RNA duplexes with Watson-Crick base pairs, *Biochemistry* 37 (1998) 14719–14735.

[51] J. Xiong, *Podstawy bioinformatyki*, transl. J. Bujnicki *et al.*, Wydawnictwa Uniwersytetu Warszawskiego, Warszawa (2009).

[52] D.R. Zerbino and E. Birney, Velvet: Algorithms for de novo short read assembly using de Bruijn graphs, *Genome Research* 18 (2008) 821-829.