CMST 11(1) 11-20 (2005) DOI:10.12921/cmst.2005.11.01.11-20

COMPUTATIONAL METHODS IN SCIENCE AND TECHNOLOGY **11(1)**, 11-20 (2005)

DNA computing

Piotr Formanowicz

Institute of Computing Science, Poznań University of Technology
Piotrowo 3A, 60-965 Poznań, Poland

Institute of Bioorganic Chemistry, Polish Academy of Sciences
Noskowskiego 12/14, 61-704 Poznań, Poland

(Rec. 20 May 2004)

Abstract: DNA computing is an alternative method of performing computations. It is based on the observation that in general it is possible to design a series of biochemical experiments involving DNA molecules which is equivalent to processing information encoded in these molecules. In classical computing devices electronic logic gates are elements which allow for storing and transforming information. Designing of an appropriate sequence or a net of "store" and "transform" operations (in a sense of building a device or writing a program) is equivalent to preparing some computations. In DNA computing the situation is analogous. The main difference is the type of computing devices, since in this new method of computing instead of electronic gates DNA molecules are used for storing and transforming information. From this follows that the set of basic operations is different in comparison to electronic devices but the results of using them may be similar. Moreover, the inherent massive parallelism of DNA computing may lead to methods solving some intractable computational problems. In this paper basic principles of DNA computing are described and examples of DNA based algorithms solving some combinatorial problems are presented.

Key words: DNA molecules, complementarity rule, combinatorial problems, algorithms, computational complexity

1. INTRODUCTION

The first person who spoke about the possibility of performing computations at the molecular level was probably Richard Feynman. His ideas had to wait over twenty years to be implemented. They evolved in two directions one of them being *quantum computing* and the other *DNA computing*. The latter one has been initialized by Leonard Adleman who solved the well-known hard combinatorial problem of finding Hamiltonian path between two vertices in a directed graph using DNA molecules [1]. His seminal paper influenced many researchers who realized that DNA molecules in principle may serve as new computing devices. Adleman has founded a new way of thinking about computations since in DNA computing some new paradigms are needed. One of them is a real massive parallelism where a huge number of encoding strings are processed at the same time. Shortly after the publication of Adleman's paper DNA based algorithms have been proposed for many classical hard combinatorial problems (cf. [8, 11]).

Nowadays the research effort in the area of DNA computing concentrates on four main problems: designing algorithms for some known combinatorial problems, designing new basic operations of "DNA computers" (they are some biochemical procedures whose results may be interpreted as results of some computations), developing new ways of encoding information in DNA molecules and reduction of errors in DNA based computations (cf. [9, 7, 10, 14]).

In this paper basic principles of DNA computing are presented and some exemplary algorithms are discussed. The organization of the paper is as follows. In Section 2 the main ideas of DNA computing are presented. In Section 3 the first DNA based algorithm, *i.e.* the one proposed by Adleman is described. In Section 4 a sticker model of DNA computing being one of the standard approaches to performing computations using DNA molecules is discussed. In Section 5 two algorithms for solving some problems of scheduling theory are presented and some extensions of them are also considered. The paper ends with conclusions in Section 6.

2. FOUNDATIONS OF DNA COMPUTING

2.1. The general idea

DNA based algorithms are composed of some basic operations, analogously to their classical counterparts. In this section the general idea of DNA computing will be briefly presented.

As it is easily to guess the nature of DNA molecules is crucial for DNA computing. In particular, the structure of these molecules, discovered by Watson and Crick [16], is fundamental for the possibility of using them for encoding and processing information. DNA molecules are composed of two strands, each of them being a sequence of *nucleotides* which are of four types denoted by A, C, G, T. From computer science point of view a DNA strand is a word

over alphabet Σ_{DNA} = {A, C, G, T}. One of the most important properties of nucleotides is their ability to join by hydrogen bonds. This property makes possible an existence of double stranded DNA molecules. To be more precise, within one DNA strand nucleotides are joined by strong phosphodiester bonds. But additionally, every nucleotide A from one strand may be joined with nucleotide T from another strand by two hydrogen bonds and C may be joined with G by three hydrogen bonds (this rule is called *Watson-Crick complementarity* and A-T and C-G are pairs of *complementary* nucleotides). These bonds are weaker than the phosphodiester ones and they allow for forming the double stranded DNA molecules. For example, if there is a single stranded molecule 5'-TTGCATTAACGAC-3' it is possible that another molecule 3'-AACGTAATTGCTG-5' will *hybridize* to it, *i.e.* it will form a duplex:

5'-TTGCATTAACGAC-3'
3'-AACGTAATTGCTG-5'

(Note, that in the above example the ends of the strands are denoted by 5' and 3'. In fact, DNA strands have orientation and by convention the left end of the molecule is denoted by 5' and the right one by 3'. In the example 3'-AACGTAATTGCTG-5' is a sequence complementary to 5'-TTGCATTAACGAC-3' – such a sequence has an opposite direction and at each position of the duplex nucleotides are complementary to each other.)

In all DNA based algorithms Watson-Crick complementarity allows for creating solutions encoded in double stranded DNA molecules composed of single stranded ones used for encoding a problem instance.

The idea of DNA computing is similar (to some extent) to the non-deterministic Turing machine. Indeed, in many DNA based algorithms at the beginning there are created DNA sequences encoding all feasible (not necessarily optimal) solutions to a given problem and a lot of other sequences which do not encode any feasible solutions. In the next steps the algorithm eliminates from this input set of sequences those which are not solutions to the considered problem and also those ones which do not encode optimal solutions. This elimination is performed successively and the power of DNA computers lies in their massive parallelism – (almost) all encoding sequences are processed simultaneously. At the end of the computations in the set of DNA molecules there remain only those which encode the optimal solution.

The elimination of "bad" sequences is usually done in such a way that the algorithm checks successively if the sequences posses some properties necessary for every string which potentially could encode the problem solution. If they do not have these properties, they are eliminated from the set.

2.2. Basic operations

Hence, the basic operations of DNA algorithms are usually designed for selecting sequences which satisfy some particular conditions. On the other hand, there may be different sets of such basic operations. In fact, any biochemical procedure which may be interpreted as a transformation (or storing) information encoded in DNA molecules may be treated as a basic operation of DNA based algorithms. One of the possible set of such operations is the following [12]:

MERGE – given two test tubes N_1 and N_2 create a new tube N containing all strands from N_1 and N_2.

AMPLIFY – given tube N create a copy of them.

DETECT – given tube N return *true* if N contains at least one DNA strand, otherwise return *false*.

SEPARATE – given tube N and word w over alphabet Σ_{DNA} create two tubes $+(N, w)$ and $-(N, w)$, where $+(N, w)$ consists of all strands from N containing w as a substring and $-(N, w)$ consists of the remaining strands.

LENGTH-SEPARATE – given tube N and positive integer n create tube $(N, \leq n)$ containing all strands from N which are of length n or less.

POSITION-SEPARATE – given tube N and word w over alphabet Σ_{DNA} create tube $B(N, w)$ containing all strands from N which have w as a prefix and tube $E(N, w)$ containing all strands from N which have w as a suffix.

Each of the above operations is a result of some standard biochemical procedure.

2.3. Encoding of instances

One of the fundamental differences between classical and DNA computing is a very close relationship between the algorithm and the molecules encoding the problem instance in the case of molecular computations.

In electronic computers the input data are stored in standard memory or processor registers and the programmer must not take care of their structure. (This is true for most cases. Some exceptions may take place in the case of programs written in low level languages. However, also in this case the situation is relatively simple since the registers have "uniform" structure, *i.e.* the same for all programs.)

While designing algorithms which will be implemented on a classical computer one also need not consider the structure of the registers for storing problem data. (Obviously, also in this case there may be some exceptions, especially when some non-standard computer architectures are used. But again, the architecture is the same for all algorithms.)

The situation is completely different in the case of DNA computing. First, there is no clear distinction between an algorithm and a program. Second, for each problem its own encoding scheme must be developed, which is very closely connected with the algorithm.

The reason is a lack of some standard architecture of the "DNA computer". Indeed, in some sense, for each

problem being solved by DNA molecules a new "hardware architecture" is developed.

So, the designing of DNA molecules for encoding problem instances is an important part of the process of algorithms developing and usually these two issues cannot be separated.

The molecules have to be designed in such a way that when poured into one test tube in a certain physical conditions they will create double stranded DNA molecules encoding potential solutions to the considered problem.

2.4. An example

The general idea of DNA computing may be illustrated by the following example. Let us consider a hypothetical combinatorial problem where certain permutation of some elements is looked for. For the sake of simplicity let us assume that the problem instance consists of only four elements *x, y, w* and *z* and let the optimal solution be *xwzy*. One of the possible way of encoding the instance in DNA molecules is as follows. To the elements of the instance there are assigned the following oligonucleotides:

x : 5'-ACCGATTA-3', *y* : 5'-GTACCATT-3', *w* : 5'-TGAACCTA-3',
z : 5'-AATTCGCG-3'

This set of oligonucleotides is not sufficient for the above mentioned process of forming DNA duplexes being potential solutions to the problem. There have to be added some additional oligonucleotides which would be able to hybridize to right half of some of the oligonucleotides shown above and to left half of some other. These oligonucleotides will bind consecutive oligonucleotides in a double stranded DNA molecule encoding potential solution.

In the presented example such "binding" oligonucleotides should be designed for every ordered pair of the molecules encoding the instance. They should be as follows:

x – *y*: 3'-TAATCATG-5', *x* – *w*: 3'-TAATACTT-5', *x* – *z*: 3'-TAATTTAA-5',
y – *x*: 3'-GTAATGGC-5', *y* – *w*: 3'-GTAAACTT-5', *y* – *z*: 3'-GTAATTAA-5',
w – *x*: 3'-GGATTGGC-5', *w* – *y*: 3'-GGATCATG-5', *w* – *z*: 3'-GGATTTAA-5',
z – *x*: 3'-GCGCTGGC-5', *z* – *y*: 3'-GCGCCATG-5', *z* – *w*: 3'-GCGCACTT-5'

When many copies of all these oligonucleotides are poured into test tube they will form all duplexes according to the complementarity rule. The complexes will be of various lengths, *i.e.* they will be composed of various numbers of oligonucleotides. Some of them may be as follows (in the parentheses there are indicated sequences of the instance elements which correspond to the created molecules):

(*yx*) 5'-GTACCATTACCGATTA-3'
3'-GTAATGGC-5'

(*xzwx*) 5'-ACCGATTAAATTCGCGTGAACCTAACCGATA-5'
3'-TAATTTAAGCGCACTTGGATTGGC-5'

(*xwzy*) 5'-ACCGATTATGAACCTAAATTCGCGGTACCATT-3'
3'-TAATACTTGGATTTAAGCGCCATG-5'

(*wzy*) 5'-TGAACCTAAATTCGCGGTACCATT-3'
3-GGATTTAAGCGCCATG-5'

(*zxwyx*) 5'-AATTCGCGACCGATTATGAACCTAGTACCATTACCGATTA-3'
3'-GCGCTGGCTAATACTTGGATCATGGTAATGGC-5'

(*wxz*) 5'-TGAACCTAACCGATTAAATTCGCG-3'
3'-GGATTGGCTAATTTAA-5'

It should be noted that in the real biochemical experiment much more various duplexes would be created but only one of these many types of the molecules would encode the solution to the problem.

The formation of the duplexes is the first step of the DNA based algorithm. As was mentioned earlier, the next steps should be designed to eliminate from this huge variety of molecules those ones which do not encode the problem solution. This is usually done successively by eliminating molecules which do not have some properties necessary for every string encoding a solution of the considered problem.

In the example some initial steps of the algorithm should eliminate the molecules which are composed of less than four oligonucleotides, because they cannot encode any permutation of the instance elements. Next, the molecules composed of more than four oligonucleotides also should be removed, since they consist at least one repetition of some of the elements. Moreover, there has to be also a possibility of detection and removal of the molecules encoding solutions not maximizing (or minimizing) the value of the criterion function (not defined in this simple example). This issue is usually strongly problem-dependent and will be discussed in Section 5.

3. ADLEMAN'S ALGORITHM

In this section the first DNA based algorithm, *i.e.* the one proposed by Adleman [1] will be described. As has been mentioned earlier, Adleman solved an instance of a problem of finding a Hamiltonian path between two vertices in a directed graph. The problem is NP-complete in the strong sense and is formulated as follows [5]:

INSTANCE: Directed graph $G = (V, A)$, two distinguished vertices $v_s, v_t \in V$.

ANSWER: YES, if G contains a Hamiltonian path starting in vertex v_s and ending in vertex v_t, otherwise NO.

The algorithm proposed by Adleman consists of the following steps [1]:

1. Create set S of all paths in G.
2. Remove from set S those paths which do not start in v_s and do not end in v_t.
3. Remove from S paths which consists of less or more than $|V|$ vertices.
4. Remove from S those paths that traverse some vertex more than once.

5. If set S is not empty, then the answer is YES, otherwise the answer is NO.

The instance of the problem solved by Adleman consisted of only seven vertices and obviously every electronic computer is able to easily solve it. The importance of the Adleman's idea lies in the demonstration of a potential computing power of DNA molecules rather than in solving really hard instance of the problem.

The algorithm described above is some abstract procedure which should be implemented as a sequence of biochemical experiments. According to the Adleman's idea the implementation can be as follows [1]:

1. For every vertex $v_i \in V$, $i = 1, 2, \ldots, |V|$ randomly generated sequence $s_i = s_i(1)s_i(2)\cdots s_i(k)$ is assigned and for every arc $(v_i, v_j) \in A$ there is assigned sequence

$$s_{ij} = s_i\left(\tfrac{k}{2}+1\right)s_i\left(\tfrac{k}{2}+2\right)\cdots s_i(k)s_j(1)s_j(2)\cdots s_j\left(\tfrac{k}{2}\right)$$

(here we assume that k is an even integer). In case of arcs starting in vertex v_s, *i.e.* the arcs of the form $(v_s, v_x) \in A$ for some $x = 1, 2, \ldots, |V|$ the corresponding sequence is longer and has the form

$$s_{sx} = s_s(1)s_s(2)\cdots s_s(k)s_x(1)s_x(2)\cdots s_x\left(\tfrac{k}{2}\right).$$

Similarly, in case of arcs ending in v_t, *i.e.* the ones of the form $(v_x, v_t) \in A$ the corresponding sequence has the form $s_{xt} = s_x\left(\tfrac{k}{2}+1\right)s_x\left(\tfrac{k}{2}+2\right)\cdots s_x(k)s_t(1)s_t(2)\cdots s(k).$

For every sequence s_i, $i = 1, 2, \ldots |V|$ there is generated its reverse complementary sequence

$$\overline{s_i} = \overline{s_i}(1)\overline{s_i}(2)\cdots\overline{s_i}(k)$$

(here, $\overline{s_i}(j)$ denotes nucleotide complementary to nucleotide $s_i(j)$).

On the base of sequences $\overline{s_i}$ and s_{ij} there are created oligonucleotides (in Adleman's experiment 50 pmol for each sequence). It is easy to note that oligonucleotides corresponding to sequences $\overline{s_i}$ serve as binders of oligonucleotides corresponding to sequences s_{ij}.

From this follows that when all these molecules are poured into one test tube they will create duplexes corresponding to all paths present in graph G (as a result of hybridization and ligation reactions).

2. In the second step the molecules created in step 1. are multiplied by PCR with starters corresponding to sequences s_s and $\overline{s_t}$. As a result only strands encoding paths which start in vertex v_s and end in v_t are multiplied.

3. In the third step the product of step 2. is run on a gel which makes possible to distinguish molecules encoding paths of various lengths. From these molecules there are taken those which have length equal to $|V|k$.

4. In the product of step 3. there are selected those molecules which contain oligonucleotides corresponding to all sequences s_i, $i = 1, 2, \ldots, |V|$. This selection is made using hybridization probes.

5. The product of step 4. is multiplied by PCR and run on a gel in order to check if any DNA molecules corresponding to the solution of the problem remained in the test tube.

4. STICKER MODEL

In this section another model of DNA computing will be described, *i.e.* the *sticker model* [15, 12]. As in every DNA based computations also in this model a crucial property of DNA molecules is their ability to making duplexes according to the Watson-Crick complementarity rule. The main difference between the sticker model and the one previously described is that in the sticker model there is a kind of a random access memory and the computations do not depend on molecules extension like in the approach proposed by Adleman and described in the previous section.

In the sticker model there are DNA strands which serve as registers. The single stranded DNA molecules called *memory strands* serve as such registers and another type of single stranded DNAs called *stickers* are used to write and erase information in the registers.

A memory strand is composed of several segments in each of which one bit of information can be stored. Each of these segments is composed of l nucleotides. So, if a memory strand can store p bits its length has to be $n \geq lp$ nucleotides since the segments have to be non-overlapping.

Stickers are composed of l nucleotides and each of them is a DNA strand complementary to exactly one of the segments in a memory strand.

Memory strands are sequences of bits whose values are set by stickers. If a sticker hybridizes to complementary segment in memory strands, then the bit corresponding to this segment is set to 1. On the other hand, if to a given segment no sticker hybridized, then the value of this bit is 0.

A memory strand with some bits set to 1 or 0 is called a *memory complex*. An example of a memory strand and stickers for writing information in this strand is shown below:

GGGTGTTTCCCACACCTTTGGGGGAACCCACA

memory strand

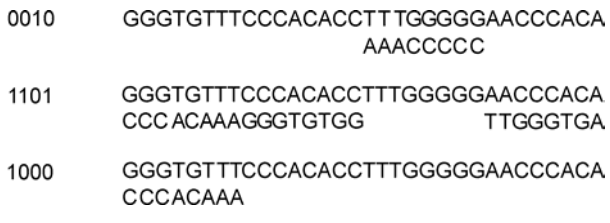
CCCACAAA AAACCCCC

GGGTGTGG TTGGGTGT

stickers

The nucleotide sequences of the memory strand segments are important and should be carefully chosen. It should be such that it would be impossible for a sticker to hybridize partially to one segment and to its neighbor. So, the sequences should differ from each other sufficiently in order to avoid such situations. Another issue which should be taken into account is a melting temperature of the du-

plexes created by stickers and memory strands. In the ideal case the temperature should be the same for all duplexes, which would made all of them equally stable in given physical conditions. Such an equal stability reduces the ratio of possible hybridization errors. Examples of memory complexes encoding binary numbers 0010, 1101 and 1000 are shown below:



The basic operations used in the sticker model differ from those described in Section 2. There are four such operations: MERGE, SEPARATE, SET and CLEAR.

Operation MERGE is the same as the one described in Section 2, *i.e.* it combines two test tubes into one tube.

Operation SEPARATE is different in sticker model. Given test tube N and integer i, $1 \le i \le p$ it produces two test tubes $+(N, i)$ and $-(N, i)$. Tube $+(N, i)$ consists of all memory complexes from N where the bit in position i is set to 1, while tube $-(N, i)$ contains those complexes, where this bit is set to 0.

Given test tube N and integer i, $1 \le i \le p$, operation SET produces test tube $set(N, i)$, where the bit in position i of each memory complex is set to 1.

Analogously, operation clear produces test tube $clear(N, i)$, where in each memory complex the bit in position i is set to 0.

Obviously, computations in the sticker model are sequences of basic operations. Such a sequence must start with some initial set of memory complexes and must produce a set of such complexes being the answer to the problem under consideration.

The initial set of memory complexes is called (p, q) *library* of strings. Such a library consists of memory complexes composed of p segments, where q of them are set to 0 or 1 in all possible ways (*i.e.* in each complex there is written q-bit binary number in a random way, so in the whole (p, q) library all q-bit binary numbers are written). The remaining p-q segments (bits) are set to 0. The idea is to search all 2^q possible input strings in parallel and remove those ones which do not fulfill the criteria necessary for the solution to the considered problem.

In order to read the answer it is necessary to isolate a memory complex from the final tube and determine in which positions the stickers hybridized.

Let us illustrate the idea of the sticker model by its application to solving the NP-complete Minimum Cover problem, which is formulated as follows [5]:

INSTANCE: collection $C = \{C_1, C_2, \ldots, C_\alpha\}$ of subsets of finite set $S = \{1, 2, \ldots, \beta\}$, positive integer $K \le \alpha$.

ANSWER: YES, if C contains a cover for S of size K or less, *i.e.* a subset $C' \subseteq C$ with $|C'| \le K$ such that every element of S belongs to at least one member of C', otherwise, NO.

In the search version of the problem the answer is the subset $C' \subseteq C$ of minimal cardinality containing all elements of S or equivalently the minimal set $I \subseteq \{1, 2, \ldots, \alpha\}$ such that $\bigcup_{i \in I} C_i = S$.

In order to solve the search version of the problem using the sticker model memory complexes representing all possible 2^α subsets of C should be created [15, 12]. So, the first α bits in each memory complex in initial test tube N_0 indicate some subset C' of collection C, which is equivalent to set $I \subseteq \{1, 2, \ldots, \alpha\}$ such that $C' = \{C_{i_1}, C_{i_2}, \ldots C_{i_z}\}$, where $I = \{i_1, i_2, \ldots, i_z\}$. The last β bits in each complex are initially set to 0. In a given memory complex representing set I bits in positions $\alpha + j$ will be set to 1 if $j \in \bigcup_{i \in I} C_i$.

The algorithm solving the Minimum Cover problem is based on the following idea [15, 12]. In the test tube there are sequentially checked all bits in the memory strands, which represent the subsets of collection C (*i.e.* bits in positions 1, 2, …, α). If some bit in position i, $1 \le i \le \alpha$ is set to 1 in a given memory complex then the operation SET is used to set to 1 all bits in positions $\alpha + j$, where $j \in C_i$. After processing all of the first α bits in this way it is checked if all bits in positions $\alpha + 1$, $\alpha + 2$, …, $\alpha + \beta$ are set to 1. If in a given memory complex it is the case it means that subset C' represented by this complex covers set S. It remains to find among such complexes those ones which represent subsets of minimal cardinality.

Formally, the algorithm is as follows (c_i^j denotes the j-th element of set C_i, assuming, that the elements are in some way indexed) [15, 12]:

```
1.  for i = 0 to α do begin
2.      separate +(N0, i) and −(N0, i)
3.      for j = 1 to |Ci| do begin
4.          set (+(N0, i), α + ci^j)
5.      end
6.      N0 ← merge(+(N0, i), −(N0, i))
7.  end
8.  for i = α + 1 to α + β do begin
9.      N0 ← +(N0, i)
10. end
11. for i = 0 to α − 1 do begin
12.     for j = i down to 0 do begin
13.         separate +(Nj, i +1) and −(Nj, i + 1)
14.         Nj+1 ← merge(+(Nj, i + 1), Nj+1)
15.         Nj ← −(Nj, i + 1)
16.     end
17. end
18. read N1
```

19. else if it was empty read N_2
20. else if it was empty read N_3
.....
$17 + \alpha$. else if it was empty read N_α

It is interesting to note that in the presented approach all 2^α subsets of collection C are checked. If some classical algorithm would be used for this purpose its time complexity would be $O(2^\alpha)$. On the other hand, in the described DNA based algorithm all subsets containing set C_i, $1 \leq i \leq \alpha$ are processed in parallel, so the complexity of the approach is $O(\alpha(\alpha+\beta))$.

5. ALGORITHMS FOR SOME SCHEDULING PROBLEMS

5.1. Formulation of the problems

In this section DNA based algorithms for some problems of scheduling tasks on a single machine with limited availability will be presented. (Good overviews on scheduling theory are given in [2, 13].) The algorithms use the same model of computation as the Adleman's approach. Let us note that the algorithm described in Section 3 solves a decision problem, while the algorithms presented in this section solve search problems which are formulated as follows (here we use the standard three-field notation to denote the problems (cf. [2, 4])):

PROBLEM 1, $h_1||C_{\max}$:

INSTANCE: Set of n tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ to be processed, processing time p_j for each $T_j \in \mathcal{T}$, starting time of a period of the machine non-availability s and length of this period h.

ANSWER: A feasible schedule of minimal length.

PROBLEM 1, $h_k||C_{\max}$:

INSTANCE: Set of n tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ to be processed, processing time p_j for each $T_j \in \mathcal{T}$, starting times of the periods of machine non-availability $s_1, s_2, \ldots, s_K$ and lengths of these periods $h_1, h_2, \ldots, h_K$.

ANSWER: A feasible schedule of minimal length.

In the above formulations feasible schedule means such a schedule where the machine processes at most one task at any time and no task is processed when the machine is not available. Both of these problems are computationally hard, *i.e.* the first of them is NP-hard in the ordinary sense and the second one is NP-hard in the strong sense [6].

As has been mentioned before one of the most important aspects of DNA computing are strong connections between encoding scheme, *i.e.* the ways in which the instance of the considered problem is encoded in DNA molecules and the algorithm itself. Obviously, such connections exists also in the case of classical algorithms but in DNA computing the encoding scheme influences the algorithm usually much more than in the classical case. Hence, the scheme should be developed very carefully.

In the following subsections first there will be described the encoding schemes and then the algorithms for the problems under consideration [3].

5.2. The algorithm for problem 1, $h_1||C_{\max}$

The general idea of the algorithm for this problem follows from the obvious observation that in order to solve it one should construct a partial schedule for the time period between a starting point of the schedule and the starting point of the non-availability period. In this partial schedule the idle time should be as short as possible and the sequence of tasks scheduled before the non-availability period may be arbitrary (the sequence of the remaining tasks, *i.e.* those scheduled after the period, obviously also may be arbitrary). It is easy to see that the criterion function, *i.e.* the schedule length, is minimized when the idle time is minimized. Hence, in order to solve the problem optimally it suffices to choose a subset $\mathcal{T}' \in \mathcal{T}$ such that $\Sigma_{T_i \in \mathcal{T}'} p_i$ is maximal, but not greater than s.

The algorithm follows the "standard" framework of DNA computing described in Section 2, *i.e.* at the beginning sequences encoding all tasks and some auxiliary sequences are developed and oligonucleotides corresponding to them are synthesized and poured into a test tube [3]. As a result of biochemical reactions in the tube there are created longer DNA molecules which are concatenations of oligonucleotides encoding the tasks (in one strand) and the auxiliary oligonucleotides (in the complementary strand). Most of these molecules do not encode any feasible solution to the problem but some of them do. Moreover, there is also a small fraction of molecules which encode the optimal solutions. The goal of the algorithm is to remove from the test tube all DNA molecules which do not encode those solutions. In order to read the information encoded in the resulting DNA molecule a standard *DNA sequencing* procedure can be applied.

Encoding scheme

The general principle of the encoding scheme is that each task T_i, $i = 1, 2, \ldots, n$ is encoded by a unique DNA sequence (oligonucleotide) [3]. For the sake of simplicity of the method description in what follows we will not distinguish between sequences and their corresponding oligonucleotides since the latter are physical "implementations" of the former. (Note that the oligonucleotides described here should be used in many copies in the biochemical experiment, as usual in DNA computing.) We will denote such a sequence by o_i. The length of o_i is equal to dp_i, where d is an even integer constant. Moreover, for

building the solution some auxiliary oligonucleotides are necessary which join two consecutive oligonucleotides encoding tasks in the sequence encoding the potential solution. Each task from set $\mathcal{T}$ has to appear exactly ones in the solution to the problem. In order to avoid a repetition of some tasks in the schedule for each T_i, $i = 1, 2, \ldots, n$ it is necessary to create oligonucleotides joining o_i with o_j for $j = i + 1, i + 2, \ldots, n$. It means that it is necessary to create DNA sequences complementary to the right half of o_i and the left half of o_j. We will denote such a sequence by

$$u_{i,j} = \overline{o}_i\left(\frac{dp_i}{2}+1\right)\overline{o}_i\left(\frac{dp_i}{2}+2\right)\cdots\overline{o}_i(dp_i)\overline{o}_j(1)\,\overline{o}_j(2)\ \cdots\overline{o}_j\left(\frac{dp_j}{2}\right).$$

All these oligonucleotides are poured into test tube N where they form a variety of double stranded DNA molecules (some of them will encode the optimal solution of the problem). The algorithm solving problem 1, $h_1||C_{\max}$ is as follows [3]:

Algorithm

1. input (N)
2. $N \leftarrow (N, \leq ds)$
3. $N \leftarrow (N, \max)$

In step 1. the algorithm "reads" an input, *i.e.* the test tube N containing DNA strands encoding all potential solutions to the problem.

In step 2. from the solution N there are extracted only those sequences whose lengths do not exceed ds nucleotides. The other ones are removed and the result of this operation is assigned again to tube N.

In step 3. in tube N there are kept only those sequences which have maximal length.

As one can notice in the algorithm there is used only one type of the basic operations described in Section 2. In step 2. LENGTH-SEPARATE is applied in order to remove those molecules which are to long for encoding the partial schedule which could fit into the time slot before the non-availability period. In step 3. the variant of this operation is used which selects the longest molecule. This molecule encodes the optimal subset of tasks which should be scheduled before the non-availability interval. The sequence of these tasks does not affect the value of the criterion function. The remaining tasks can be scheduled arbitrarily after the non-availability period. The schedule length is equal to $s + h + \Sigma_{T_i \in \mathcal{T}-T'} p_i$.

5.3. The algorithm for problem 1, $h_k||C_{\max}$

The general idea of the algorithm is to some extent similar to the previous one [3]. Indeed, optimal solution to the problem 1, $h_k||C_{\max}$ may be decomposed into partial schedules which fit to the time intervals between any two consecutive non-availability periods (and between the starting point of the schedule and the starting point of the first non-availability interval). Hence, the problem can be formulated as the one of finding partial schedules for the availability periods such that the sum of the idle times in all slots are minimal. However, arbitrary number of the non-availability periods in the problem makes the algorithm solving it much more complicated than the one for problem 1, $h_1||C_{\max}$.

Encoding scheme

Besides the oligonucleotides encoding the tasks several additional types of molecules are used in the algorithm. All the oligonucleotides necessary for the computations are listed below [3]:

1. For each task T_i, $i = 1, 2, \ldots, n$ there are created oligonucleotides σ_i of length dp_i. For encoding any pair of tasks T_i and T_j for which $p_i = p_j$ different oligonucleotides encoding them should be used.
2. There is created oligonucleotide ε of length equal to d nucleotides corresponding to a unit idle time (the sequence of ε has to be different from all o_i, $i = 1, 2, \ldots, n$.)
3. For each task T_i, $i = 1, 2, \ldots, n$ it is necessary to create oligonucleotides joining o_i with o_j for $j = 1, 2, \ldots, n$ and $i \neq j$. It means that it is necessary to create DNA sequences complementary to the right half of o_i and the left half of o_j. We will denote such a sequence by

$$u_{i,j} = \overline{o}_i\left(\frac{dp_i}{2}+1\right)\overline{o}_i\left(\frac{dp_i}{2}+2\right)\cdots\overline{o}_i(dp_i)\overline{o}_j(1)\,\overline{o}_j(2)\ \cdots\overline{o}_j\left(\frac{dp_j}{2}\right).$$

4. For each task T_i, $i = 1, 2, \ldots, n$ oligonucleotides joining o_i with ε and ε with o_i are created. These oligonucleotides are denoted by

$$u_{i,\varepsilon} = \overline{o}_i\left(\frac{dp_i}{2}+1\right)\overline{o}_i\left(\frac{dp_i}{2}+2\right)\cdots\overline{o}_i(dp_i)\overline{\varepsilon}(1)\,\overline{\varepsilon}(2)\ \cdots\overline{\varepsilon}\left(\frac{d}{2}\right)$$

and $u_{\varepsilon,i} = \overline{\varepsilon}\left(\frac{d}{2}+1\right)\overline{\varepsilon}\left(\frac{d}{2}+2\right)\cdots\overline{\varepsilon}(d)\overline{o}_i(1)\,\overline{o}_i(2)\ \cdots\overline{o}_i\left(\frac{dp_i}{2}\right)$,

respectively.

5. There is generated an oligonucleotide joining ε with ε, denoted by

$$u_{\varepsilon} = \overline{\varepsilon}\left(\frac{d}{2}+1\right)\overline{\varepsilon}\left(\frac{d}{2}+2\right)\cdots\overline{\varepsilon}(d)\overline{\varepsilon}(1)\overline{\varepsilon}(2)\cdots\overline{\varepsilon}\left(\frac{d}{2}\right).$$

6. For each task T_i, $i = 1, 2, \ldots, n$ oligonucleotide $u_i = \overline{o}_i(1)\overline{o}_i(2)\cdots\overline{o}_i\left(\frac{dp_i}{2}\right)$ complementary to the left half of o_i is created.
7. Oligonucleotide $u_{\varepsilon-} = \overline{\varepsilon}(1)\overline{\varepsilon}(2)\cdots\overline{\varepsilon}\left(\frac{d}{2}\right)$ complementary to the left half of ε is also created.

Algorithm

According to the described above encoding scheme the lengths of the non-availability intervals are not represented by any oligonucleotides [3]. Indeed, it is assumed that all of these lengths are equal to zero. This assumption simplifies the encoding scheme and the algorithm. On the other hand, the lengths may be easily added to the resulting schedule,

so the final schedule will be the real solution to the problem.

The reduction of all non-availability periods lengths to zero causes the change of the periods' starting times (except the first one). The new ones may be computed according to the following formulae:

$$q_1 = s_1,$$

$$q_i = s_i - \sum_{k=1}^{i-1} h_k, \quad i = 2, 3, \ldots, K.$$

The algorithm also uses the lengths of time intervals between any two consecutive non-availability periods. They may be determined using formulae:

$$v_1 = s_1,$$

$$v_i = s_i - (s_{i-1} + h_{i-1}), \quad i = 2, 3, \ldots, K.$$

The input to the algorithm are the following four test tubes:

N – it consists of o_i for $i = 1, 2, \ldots, n$ and ε,
N_0 – it consists of o_i for $i = 1, 2, \ldots, n$,
N_1 – it consists of u_i for $i = 1, 2, \ldots, n$ and u_{ε^-},
N_2 – it consists of $u_{i,j}$ for $i = 1, 2, \ldots, n, j = 1, 2, \ldots, n$, $i \neq j$, $u_{i,\varepsilon}$, $u_{\varepsilon,i}$ for $i = 1, 2, \ldots, n$ and u_{ε}.

The algorithm for problem 1, $h_k||C_{\max}$ works as follows [3]:

1. input(N, N_0, N_1, N_2)
2. $N_3 \leftarrow$ merge(N, N_2)
3. for $i = 1$ to K do begin
4. $N_{\text{temp}} \leftarrow$ amplify(N_3)
5. $M_i \leftarrow (N_{\text{temp}} = dv_i)$
6. end
7. $M_1 \leftarrow$ merge(M_1, N_1)
8. $M = \emptyset$
9. for $i = 1$ to K do begin
10. $M \leftarrow$ merge(M, M_i, N_2)
11. $M \leftarrow (M, =dq_i)$
12. end
13. $N \leftarrow$ merge(M, N_0)
14. $N \leftarrow$ merge(N, N_2)
15. for $i = 1$ to n do begin
16. $N \leftarrow +(N, o_i)$
17. end
18. $N \leftarrow (N, \min)$

In line 1. of the algorithm the input data are read, *i.e.* tubes N, N_0, N_1 and N_2.

In line 2. tube N_3 containing all oligonucleotides from N and N_2 is created.

The loop in lines 3.-6. is performed K times (K is the number of non-availability intervals). In this loop tubes M_1, M_2, …, M_K are created. These tubes contain partial schedules corresponding to the time slots between any pair of two consecutive non-availability intervals (except the first one, which contains the partial schedules for the time slot before the first non-availability interval). These partial schedules will be merged in the next steps of the algorithm.

In line 7. the terminators, *i.e.*, oligonucleotides u_i for $i = 1, 2, \ldots, n$ and u_{ε^-} are joined to the ends of the partial schedules for the first time slot (between the starting point of the schedule and the first non-availability period). The terminators guarantee that in the following steps of the algorithm potential partial schedules corresponding to the first time slot will be at the beginning of every schedule found by the algorithm (the terminators block left ends of these partial schedules and no DNA molecule can hybridize there). If the terminators were not joined at this stage of the algorithm it could happen that partial schedules corresponding to the second time slot would joined to the left ends of those ones which correspond to the first slot. In this way infeasible schedule would be created (because such a schedule would not correspond to the time slots between the non-availability periods).

In line 8. there is prepared a new test tube M, which at the beginning consists of no DNA strands.

The loop in lines 9.-12. is performed K times. In each iteration i of this loop a partial solution is extended by a partial schedule which fits to the i-th time slot. The solution which is built in the loop is kept in tube M. At iteration i all DNA strands whose lengths are not equal to the sum of lengths of the first i time slots are removed from M.

In lines 13. and 14. the partial solution is extended by the partial schedule corresponding to the last time slot, *i.e.* time interval which begins at the end of the last non-availability interval and the solutions are kept in tube N.

In the loop in lines 15.-17. all strands which do not contain at least one occurrence of every task are removed from tube N. If some task appears more than once every "additional" copies of the task should be interpreted as idle times.

In step 18. DNA strands with minimal length are extracted and the others are removed from N. This test tube after the step contains optimal solutions to the problem. Indeed, at the end of the algorithm tube N contains only those strands which encode each task at least once. Moreover, the way of creation the partial schedules in lines 3.-6. and joining them together in lines 9.-12. ensures that in the final schedule there are taken into account the moments when non-availability periods begin, *i.e.* this moments correspond to the ending point of some task or idle time in the schedule. Obviously, in order to obtain the real length of the schedule it is necessary to add the sum of all non-availability intervals to the length of the molecules from tube N divided by d.

The number of steps in this algorithm is proportional to the number of non-availability intervals (the loops in lines

3.-6. and 9.-12.) and the number of tasks (the loop in lines 15.-17.), *i.e*. its time complexity is $O(n + K)$.

5.4. Some extensions of the algorithms

It is easy to note that the algorithms presented in this section can be easily adopted to solve some other hard combinatorial problems.

First of all they can be used to solve the well-known NP-complete Partition problem and strongly NP-complete 3-Partition problem defined as follows [5]:

PARTITION:

INSTANCE: Finite set A and size $s(a) \in \mathbb{Z}^+$ for each $a \in A$.

ANSWER: YES, if there is subset $A' \subseteq A$ such that $\Sigma_{a \in A'} s(a) = \Sigma_{a \in A \setminus A'} s(a)$; otherwise, NO.

3-PARTITION:

INSTANCE: Set A of $3m$ elements, bound $B \in \mathbb{Z}^+$ for each $a \in A$ such that $\frac{B}{4} < s(a) < \frac{B}{2}$ and such that $\Sigma_{a \in A} s(a) = mB$.

ANSWER: YES, if A can be partitioned into disjoint sets A_1, A_2, …, A_m such that, for $1 \le i \le m$, $\Sigma_{a \in A_i} s(a) = B$; otherwise, NO.

In order to solve the Partition problem by the algorithm for problem 1, $h_1||C_{\max}$ it suffices to set $\mathcal{T} = A$ and $s = \frac{1}{2}\Sigma_{a \in A} s(a)$. In the algorithm we can omit step 3. and perform operation $N \leftarrow (N, = ds)$ instead of $N \leftarrow (N, \le ds)$ in step 2. If there remains any sequence in tube N, then the answer is YES. Otherwise the answer is NO.

The algorithm for problem 1, $h_k||C_{\max}$ solves the 3-Partition problem if we set $\mathcal{T} = A$, $K = m - 1$, $\forall_{i \in \{1, 2, \ldots, K\}} s_i = iB$ and $\forall_{i \in \{1, 2, \ldots, K\}} h_i = 0$. If tube N contains sequence of length dmB after step 18., then the answer is YES. Otherwise the answer is NO.

Moreover, let us also observe that the algorithm for problem 1, $h_1||C_{\max}$ can be also used to solve problem $P2||C_{\max}$ defined as follows:

PROBLEM $P2||C_{\max}$:

INSTANCE: Set of n tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ to be processed, processing time p_j for each $T_j \in \mathcal{T}$, set $\mathcal{M} = \{M_1, M_2\}$ of parallel identical machines.

ANSWER: A feasible schedule of minimal length.

In order to obtain a solution to this problem it suffices to schedule on the first machine tasks encoded by molecules remaining in tube N after step 3. The other tasks should be scheduled on the second machine. (Obviously, the order of the machines is arbitrary.)

One may also observe that the algorithm for problem 1, $h_k||C_{\max}$ can be used to solve decision problem $Pm||C_{\max} \le y$:

PROBLEM $Pm||C_{\max} \le y$:

INSTANCE: Set of n tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ to be processed, processing time p_j for each $T_j \in \mathcal{T}$, set of m parallel identical machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$.

ANSWER: A feasible schedule of length not greater than y.

In this case it suffices to set $K = m - 1$ and $v_i = y$ for $i = 1, 2, \ldots, K$. If the molecules obtained in step 18. have lengths not greater than dmy, then the answer is YES; otherwise, the answer is NO.

It is also easy to note that the same algorithm can be used to solve analogous problem with periods of machine non-availability, *i.e*. $Pm, h_{ik}||C_{\max} \le y$ defined as follows:

PROBLEM $Pm, h_{ik}||C_{\max} \le y$:

INSTANCE: Set of n tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ to be processed, processing time p_j for each $T_j \in \mathcal{T}$, set of m parallel identical machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$, number of non-availability periods K_i on machine M_i for $i = 1, 2, \ldots, m$, starting times s_{ik} and lengths h_{ik} of the periods of machine non-availability for $i = 1, 2, \ldots, m$ and $k = 1, 2, \ldots, K_i$.

ANSWER: A feasible schedule of length not greater than y.

In this case it is necessary to set values of K, v_i and q_i for $i = 1, 2, \ldots, K$ according to the following algorithm:

1. $K_0 = 0$
2. for $i = 1$ to m do begin
3. $\quad s_{i0} = 0$
4. $\quad h_{i0} = 0$
5. $\quad s_{i,K_i+1} = y$
6. end
7. for $i = 1$ to m do begin
8. $\quad$ for $k = 1$ to $K_i + 1$ do begin
9. $\quad\quad z = \sum_{l=0}^{i-1} K_l + (i-1) + k$
10. $\quad\quad v_z = s_{ik} - \left(s_{i,k-1} + h_{i,k-1}\right)$
11. $\quad\quad q_z = \sum_{p=1}^{z} v_p$
12. $\quad$ end
13. end
14. $K = \Sigma_{i=1}^{m} K_i + m$

If the molecules remaining in tube N after performing step 18. have lengths not greater than $d(my - \Sigma_{i=1}^{m} \Sigma_{k=1}^{K_i} h_{ik})$, then the answer is YES. Otherwise, the answer is NO.

6. CONCLUSIONS

In this paper basic concepts of DNA based computations have been presented and illustrated by some algorithms known from the literature. This new model of computing requires some new paradigms and offer, at least in principle, new possibilities for solving hard problems. One of the most important properties of DNA computing is

its real massive parallelism (in "DNA computers" billions of DNA molecules play the role of elementary processors). The model of DNA computing is similar, to some extent, to the non-deterministic Turing machine. Obviously, the sizes of instances possible to solve by the methods of this type are limited by the amount of DNA molecules necessary for encoding all potential solutions to the problem under consideration.

One of the open questions concerning DNA computing is its universality. It is still not clear if it will be possible to develop a kind of universal "DNA computer" able to solve any algorithmic problem. Most of the biochemical procedures solving some algorithmic problems proposed until now can be used to solve only one particular problem and can be seen as some simple specialized "DNA computers". It should be also noted that it is relatively easy to develop DNA based algorithms solving some set or graph theoretical problems, while it is much more difficult to design such methods solving arithmetic problems. It follows from the general framework of DNA computing where the operations on set of molecules are the most natural ones. On the other hand, there are attempts to model by DNA molecules deterministic Turing machines which may eventually lead to a construction of universal molecular computer.

References

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

[2] J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt and J. Węglarz, *Scheduling Computer and Manufacturing Processes*, Springer-Verlag, Berlin 1996.

[3] J. Błażewicz, P. Formanowicz, and R. Urbaniak, *DNA Based Algorithms for Some Scheduling Problems*, Lecture Notes in Computer Science **2611**, 673-683 (2003).

[4] P. Formanowicz, *Selected deterministic scheduling problems with limited machine availability*, Pro Dialog, **13**, 91-105 (2001).

[5] 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.

[6] C.-Y. Lee, *Machine scheduling with an availability constraint*, Journal of Global Optimization, **9**, 395-416 (1996).

[7] L. F. Landweber, E. B. Baum (eds.), *DNA Based Computers II*, American Mathematical Society, 1999.

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

[9] R. J. Lipton, E. B. Baum (eds.), *DNA Based Computers,* American Mathematical Society, 1996.

[10] A. Marathe, A. E. Condon and R. M. Corn, *On combinatorial DNA word design*, Journal of Computational Biology, **8**, 201-219 (2001).

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

[12] G. Păun, G. Rozenberg and A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Berlin 1998.

[13] M. Pinedo and Scheduling, *Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs, New Jersey 1995.

[14] S. Roweis and E. Winfree, *On the reduction of errors in DNA computation*, Journal of Computational Biology, **6**, 65-75 (1999).

[15] S. Roweis, E. Winfree, R. Burgoyne, N. V. Chelyapov, M. F. Goodman, P. W. K. Rothemund and L. M. Adleman, *A sticker-based model for DNA computation*, Journal of Computational Biology, **5**, 615-629 (1998).

[16] J. D. Watson and F. H. C. Crick, *A structure of deoxiribose nucleic acid*, Nature, **171**, 737-738 (1953).