Proc. Estonian Acad. Sci. Phys. Math., 2000, **49**, 2, 90–99
https://doi.org/10.3176/phys.math.2000.2.03

FORMAL MODELS OF DNA COMPUTING: A SURVEY

Tiina ZINGEL

Institute of Computer Science, University of Tartu, Liivi 2, 50409 Tartu, Estonia; tiinan@ut.ee

Received 24 May 1999, in revised form 19 November 1999

Abstract. DNA computing is a method of solving mathematical problems with the help of biological operations on DNA strands. The paper gives an overview of the existing formal models of DNA computing. The biological and mathematical background is briefly described; splicing systems, insertion–deletion systems, and a model of DNA computing based on equality checking, DNA-EC, are considered more closely.

Key words: DNA computing, splicing systems, H systems, insertion–deletion systems.

1. INTRODUCTION

DNA computing was introduced by Adleman in 1994 [1] who showed how to use tools of molecular biology to solve difficult computational problems. In his experiment he solved an instance of the Directed Hamiltonian Path Problem, using DNA strands to encode the problem and biological operations to simulate the computation. Since then, Adleman's results have motivated much research in the area.

The advantages of DNA computing are its high speed, energy efficiency, and economical information storing. For example, at the moment when Adleman carried out his experiment, DNA computing was approximately 1 200 000 times faster than the fastest supercomputers. A DNA computer could permit information storing 10^{12} times more effectively and take 10^{10} times less energy than the existing computers.

In the beginning there appeared two fundamental questions about DNA computing:

1. Can any algorithm be simulated by means of DNA computing? In other words, is the DNA computing computationally complete?

2. Is it possible to design a programmable molecular computer? In other words, does there exist a universal DNA system in the same sense as a universal Turing Machine: given a computable function, it can simulate the actions of that function for any argument?

To answer these questions, several models of DNA computing, which in general can be divided into two classes, have been proposed. The first class includes the models that use operations similar to those of Adleman and are successfully implemented already in the laboratory. Models of the second class are more formal. Their properties are much easier to study, but only the first steps have been taken towards their practical implementation. In this paper we focus mostly on models of the second class.

In Section 2 we summarize the basic biological concepts necessary to understand DNA computing. In Section 3 we describe shortly Adleman's model as an example of models of the first class, while in the next three sections we focus on the models of the second class. Splicing systems are introduced in Section 4 and insertion–deletion systems in Section 5. Section 6 is addressed to a model based on equality checking, DNA-EC.

In the following we will need some notations of formal language theory. An *alphabet* Σ is a finite nonempty set and its elements are called letters or symbols. The free monoid Σ^* generated by the alphabet Σ consists of all sequences which can be obtained by catenating the letters of Σ. The elements of Σ^* are called *words*. The empty word is denoted with λ. A *language* over the alphabet Σ is defined as a subset of Σ^*.

The union, intersection, difference, and complement of languages are defined as usual. The catenation of the languages L_1 and L_2 is the set of words $L_1L_2 = \{pq | p \in L_1, q \in L_2\}$. With FIN, REG, and RE respectively the families of finite, regular, and recursively enumerable languages are denoted.

2. BIOLOGICAL PRELIMINARIES

DNA is a basic substance where the genetic information of all living beings is recorded. A double helix molecule of DNA is a (double) sequence of units called *nucleotides*. There are four types of nucleotides which differ in the chemical group called *base*. The four bases are *adenine* (A), *guanine* (G), *cytosine* (C), and *thymine* (T). Usually the names of bases are used when the nucleotides are mentioned. The pairs A, T and G, C are called *complementary*. The nucleotides form DNA strands which possess *polarity*, it means that beside the sequence also the direction of the strand is important. For example, AGC and CGA are different strands. The DNA molecule is a double spiral, formed by two complementary nucleotide strands with opposite polarity. The joining of two strands into one double helix is called *annealing* and the opposite process *melting*. Complementarity means that in the annealing process A bonds only with T and G only with C. So the annealing can take place only when in the strands there are the complementary nucleotides in the corresponding places.

We can think of DNA strands as of sequences represented by a combination of four symbols, A, G, C, and T. In mathematical terms it means that we have the

alphabet of four letters {A, G, C, T}. In models which deal with double strands an alphabet {[A/T], [C/G], [G/C], [T/A]} is used as well.

The models of DNA computing are based on different combinations of the following biological operations on DNA strands.

1. *Melting/annealing*: break apart/bond together two single DNA strands with complementary sequences.
2. *Synthesis* of a desired DNA strand of polynomial length.
3. *Separation* of the strands by length.
4. *Merging*: pour two (or more) test tubes into one.
5. *Extraction*: extract the strands that contain a given pattern as a substring.
6. *Amplifying*: make copies of DNA strands by using the Polymerase Chain Reaction (PCR).
7. *Polymerization*: transform a single strand that has a portion of double-stranded subsequence into an entire double-stranded molecule.
8. *Cutting*: cut DNA strands by using restriction enzymes.
9. *Ligation*: paste DNA strands with complementary sticky ends by using ligases.
10. *Substitution*: substitute, insert, or delete DNA sequences by using PCR site-specific oligonucleotide mutagenesis.
11. *Marking* single strands by hybridization.
12. *Destroying* the marked strands.
13. *Detection*: given a tube, check if it contains at least one DNA strand.

These operations are used to write "molecular programs", whose input is a tube with DNA strands or molecules and output is "yes", "no" or (set of) tube(s).

3. ADLEMAN'S MODEL

The first model successfully realized in practice was presented by Adleman [1] in 1994. It gives a solution to an instance of the Directed Hamiltonian Path Problem[1], which is known as NP-complete[2]. The (nondeterministic polynomial) algorithm used by Adleman to solve the problem is as follows.

Given a graph $G = (V, E)$ (with n vertices) and two vertices $v_{in}, v_{out} \in V$,

1. generate random paths through G;
2. keep only those graphs that begin in v_{in} and end in v_{out};
3. keep only those paths that enter exactly n vertices;
4. keep only those paths that enter each vertex of G at least once;
5. if any paths remain, the answer is "yes", otherwise "no".

---

[1] A directed graph with designated vertices v_{in} and v_{out} has a Hamiltonian path if and only if there exists a path that begins in v_{in} , ends in v_{out}, and enters every other vertex exactly once.

[2] A problem is NP-complete if all NP problems can be efficiently reduced to it. A computational problem is in the class NP if there is not known polynomial-time algorithm solving it, but there exists a nondeterministic algorithm for it.

To implement the algorithm, each vertex of the graph was encoded as a random DNA strand of 20 nucleotides. Then, for every edge $i \to j$ in G, a suitable strand was created, where the second half of the sequence encoding i and the first half of the sequence encoding j were concatenated. Next, copies of sequences complementary to strands encoding vertices of G were added into the test tube containing the strands encoding edges of G. As a result, double strands were created that encode all paths through G. To implement the steps of the algorithm, biological operations, such as ligation, amplification, separation, and extraction were used. For details we refer the reader to [1].

Adleman's ideas have been extended to solve some other difficult problems [2–5]. Similar models have also been proposed by different authors as a base for the construction of the molecular computer (for example, [6,7]).

4. SPLICING SYSTEMS

In 1987 Head [8] introduced a generative formalism called the *splicing system* to analyse the generative capacity of the recombinant behaviour of DNA molecules in terms of formal languages. Later, in 1995, splicing systems were suggested to represent DNA computations. For performing computations they use only the *splicing operation*, which is a combination of cutting and pasting and can be viewed as a mathemathical generalization of the DNA recombination.

The splicing systems and their extensions have been studied by many authors (for example, [9–19]). In [20] *wet splicing systems* are presented and it is shown that the splicing operation can be performed in a single-step laboratory procedure.

Before giving a formal definition of splicing systems, we have to clarify the notion of splicing operation. A *splicing rule* is a word in the form $a\#b\$c\#d$, where $a, b, c, d \in \Sigma^*$ and the symbols #, \$ do not belong to Σ. Let $x, y \in \Sigma^*$. In splicing the words x and y according to the rule r, both x and y are cut in the place determined by r and the first parts of x and y obtained in this way will be merged with the end parts of y and x, respectively.

To enable application of the splicing rule $a\#b\$c\#d$ to the words x and y, the word x must contain ab and the word y must contain cd. The symbol \$ separates the parts of the rule which apply to the first and the second word, respectively, and # marks the places of cutting.

We say that the words z and w are obtained by *splicing* the words x and y according to the splicing rule r and write

$$(x, y) \to_r (z, w)$$

if

$$x = x'abx'', \qquad z = x'ady'',$$

$$y = y'cdy'', \qquad w = y'cbx'',$$

for some x', x'', y', y''.

We must notice that *ab* can appear in the word x more than once, as can *cd* in the word y. In such case the cutting place is selected nondeterministically and the result of splicing the words x and y may be a set containing more than one pair (z, w).

Informally, a splicing system consists of a set of words (axioms) and a set of splicing rules. The *splicing language*, that is the language generated by the splicing system, is formed by all words, which can be derived from axioms and/or words obtained in the previous steps by using splicing rules.

As the system uses only cutting and pasting, we have only a restricted number of every DNA strand. So it is proper to use multisets in the system. A *multiset* M on Σ^* can be viewed as the set Σ^* with a function $M : \Sigma^* \to N \cup \{\infty\}$ (N is the set of natural numbers), where $M(w)$ represents the number of occurrences of the word $w \in \Sigma^*$ in the multiset M. The set $supp(M) = \{w \in \Sigma^* : M(w) \neq 0\} \subseteq \Sigma^*$ is the *support* of M.

Definition 1. *A splicing system is the quadruple* $\sigma = (\Sigma, T, A, R)$, *where* Σ *is an alphabet,* $T \subseteq \Sigma$ *is the terminal alphabet,* A *is a multiset on* Σ^* (*axioms*), *and* $R \subseteq \Sigma^*\#\Sigma^*\$\Sigma^*\#\Sigma^*$ *is the set of splicing rules.*

Remark. Splicing systems are also referred to as *H systems*. In the literature, the systems that we have defined here are known as extended H systems, which means that the alphabet Σ includes some nonterminals (# and $). Sometimes it is stressed that the set of axioms is a multiset, and such systems are indicated as mH systems. The systems where A is not a multiset have been studied as well.

A splicing system σ defines a binary relation $=>_\sigma$ on multisets. Let M and M' be multisets. The relation $M =>_\sigma M'$ holds iff there exist the words $x, y \in supp(M)$ and $z, w \in supp(M')$ so that

1. $M(x) \geq 1$, $M(y) \geq 1$, if $x \neq y$ [$M(x) \geq 2$, if $x = y$, respectively];
2. $(x, y) \to_r (z, w)$ according to the splicing rule $r \in R$;
3. $M'(x) = M(x) - 1$, $M'(y) = M(y) - 1$, if $x \neq y$ [$M'(x) = M(x) - 2$, if $x = y$, respectively];
4. $M'(z) = M(z) + 1$, $M'(w) = M(w) + 1$, if $z \neq w$ [$M'(z) = M(z) + 2$, if $z = w$, respectively].

Informally, if we apply splicing to the existing words (both have multiplicity at least 1), these words are consumed (their multiplicity decreases by 1) and the products of splicing are added to the multiset (their multiplicity increases by 1).

The splicing language is defined as

$$L(\sigma) = \{w \in T | A =>_\sigma^* M \text{ and } w \in supp(M)\},$$

where A is the set of axioms and $=>_\sigma^*$ is the reflective and transitive closure of $=>_\sigma$. So L contains all terminal words which can be obtained from axioms with a final number of steps.

In order to give some basic results about the properties and the computational power of splicing systems, we will need a classification of splicing systems

according to the type of the sets A and R. For the families F_1 and F_2 we will denote the family of languages that can be generated by extended H systems by $\mathrm{EH}(F_1, F_2) = \{L(\sigma) \mid \sigma = (\Sigma, T, A, R), A \in F_1, R \in F_2\}$ (A is not a multiset)

$$\mathrm{EH}(mF_1, F_2) = \{L(\sigma) \mid \sigma = (\Sigma, T, A, R), supp(A) \in F_1, R \in F_2\}$$

$$\mathrm{EH}(m[k], F_2) = \{L(\sigma) \mid \sigma = (\Sigma, T, A, R), card(supp(A)) \leq k, R \in F_2\}$$

The following results are obtained:

1. $\mathrm{EH}(mF_1, F_2) = \mathrm{EH}(m[2], F_2) = \mathrm{RE}$, for all families F_1, F_2 such that $\mathrm{FIN} \subseteq F_1 \subseteq \mathrm{RE}$, $\mathrm{FIN} \subseteq F_2 \subseteq \mathrm{RE}$ [9].
2. $\mathrm{EH}(\mathrm{FIN}, \mathrm{REG}) = \mathrm{RE}$ [11].
3. For every given alphabet T there exists a splicing system, with finitely many axioms and finitely many rules, that is universal for the class of systems with the terminal alphabet T [9]. It means that thoretically a molecular computer can be designed based on splicing.

5. INSERTION–DELETION SYSTEMS

As a model of DNA computing an insertion–deletion (insdel) system was introduced by Kari and Thierrin [21] in 1996. An insdel system is a generative mechanism based only on two operations, (contextual) insertion and (contextual) deletion, which can also be implemented in the laboratory, using standard techniques of molecular biology.

Let u and v be words over the alphabet Σ and $(x, y) \in \Sigma^* \times \Sigma^*$ be a pair of words called a context. The (x, y)-*contextual insertion* of v into u is defined as

$$u \leftarrow_{(x,y)} v = \{u_1 x v y u_2 \mid u = u_1 x y u_2, u_1, u_2 \in \Sigma^*\}.$$

So the insertion of a word takes place only if certain contexts are present. If the word u does not contain xy as a subword, the result of the (x, y)-contextual insertion is the empty set.

In a similar manner the contextual deletion is defined. Let $(x, y) \in \Sigma^* \times \Sigma^*$ be a context. The (x, y)-*contextual deletion* of $v \in \Sigma^*$ from $u \in \Sigma^*$ is defined as

$$u \rightarrow_{(x,y)} v = \{u_1 x y u_2 \mid u_1, u_2 \in \Sigma^*, u = u_1 x v y u_2\}.$$

An *insertion* (*deletion*) *rule* is a triple (x, z, y) where (x, y) is the context and z is the word to be inserted (deleted).

Definition 2. *An insdel system is a quintuple*

$$ID = (\Sigma, T, A, I, D),$$

where Σ is an alphabet, $T \subseteq \Sigma$ is the terminal alphabet of ID, $A \subseteq \Sigma^*$ is the set of axioms, $I \subseteq \Sigma^* \times \Sigma^* \times \Sigma^*$ is the set of insertion rules, and $D \subseteq \Sigma^* \times \Sigma^* \times \Sigma^*$ is the set of deletion rules.

We say that $v \in \Sigma^*$ is directly derived from $u \in \Sigma^*$ and write $u => v$ iff one of the following cases holds:

1. $u = u_1xyu_2$, $v = u_1xzyu_2$, for some $u_1, u_2 \in \Sigma^*$ and $(x, z, v) \in I$ (an insertion step);

2. $u = u_1xzyu_2$, $v = u_1xyu_2$, for some $u_1, u_2 \in \Sigma^*$ and $(x, z, v) \in D$ (a deletion step).

Denoting by $=>^*$ the reflexive and transitive closure of the relation $=>$, we define the language generated by *ID* as follows:

$$L(ID) = \{w \in T^* \mid u =>^* w \text{ for some } u \in A\}.$$

An insdel system *ID* is of weight (n, m, p, q) if

$$\max\{|z| : (u, z, v) \in I\} = n,$$

$$\max\{|u| : (u, z, v) \in I \text{ or } (v, z, u) \in I\} = m,$$

$$\max\{|z| : (u, z, v) \in D\} = p,$$

$$\max\{|u| : (u, z, v) \in D \text{ or } (v, z, u) \in D\} = q.$$

INS(m, n)DEL(p, q) denotes the family of languages $L(ID)$ generated by insdel systems of weight (n', m', p', q') such that $n' \leq n$, $m' \leq m$, $p' \leq p$, $q' \leq q$. The family of all insdel languages is INS(∞, ∞)DEL(∞, ∞).

Various results about the properties and characterizations can be found in [22]. For example,

1. RE = INS(∞, ∞)DEL(∞, ∞), RE = INS$(2, 1)$DEL$(1, 1)$. Also, the insdel systems that insert and delete only strings over a one-letter alphabet, are computationally complete.

2. There exist universal insdel systems.

6. DNA-EC

The model of DNA computation based on equality checking, DNA-EC, was proposed by Yokomori and Kobayashi [23]. DNA-EC makes use of the test operations and set operations that we are going to describe below. Beside these operations, in the model only amplification mentioned in Section 2 is applied.

Let T denote a test tube containing a set of strings over a fixed alphabet and $I(T)$ be its contents. We define two set test operations:

1. *Emptiness Test* (*EM*): Given a test tube T, return "yes" if it contains a string and "no" otherwise.

2. *Equivalence Test* (*EQ*): Given a test tube T that may contain double-stranded strings, return "yes" if $I(T)$ contains at least one complete double-stranded string and "no" otherwise.

Also, we consider the following set operations (here $k \geq 0$, T, T' are set variables for test tubes used in DNA-EC and $a, b, c, d \in \Sigma$, $u, v, w \in \Sigma^*$):

1. *Union*:	$T = T' \cup T''$	$I(T) = I(T') \cup I(T'')$
2. *Left cut*:	$T = a\backslash T'$	$I(T) = \{u \mid au \in I(T')\}$
3. *Left adding*:	$T = aT'$	$I(T) = \{a\}I(T')$
4. *Extraction*:	$T = \mathrm{Ex}(T', w)$	$I(T) = I(T') \cap \Sigma^*\{w\}\Sigma^*$
5. *Deletion*:	$T = \mathrm{De}(T', w)$	$I(T) = \{uv \mid uwv \in I(T')\}$
6. *2-bit replacement*:	$T = \mathrm{Rb}(T', ab, cd)$	$I(T) = \{xcdy \mid xaby \in I(T')\}$
7. *Replacement*:	$T = \mathrm{Re}(T', u, v)$	$I(T) = \{xvy \mid xuy \in I(T')\}$
8. *Reversal*:	$T = (T')^R$	$I(T) = \{u^R \mid u \in I(T')\}$
9. *Right cut*:	$T = T'/a$	$I(T) = \{u \mid ua \in I(T')\}$
10. *Right adding*:	$T = T'a$	$I(T) = I(T')\{a\}$
11. *Cut*:	$T = \mathrm{Cu}(T', a)$	$I(T) = \{u, v \mid uav \in I(T')\}$
12. *Initialization*:	$T = \mathrm{In}(\Sigma, k)$	$I(T) = \Sigma^k$

With *DNA*(O, T)-*EC* we denote the class of languages recognized by DNA-EC, with collections of set operations O and test operations T. In [23] computational completeness of DNA-EC is proved. More precisely,

1. DNA(O, *EM*)-EC = RE, where O = {union, left cut, left adding, extraction, replacement}.

2. DNA(O, *EQ*)-EC = RE, where O = {union, 2-bit replacement, left adding, left cut, reversal, extraction, cut}.

3. DNA(O, *EQ*)-EC = RE, where O = {union, left adding, left cut, extraction}.

7. CONCLUSIONS

We have seen that from the beginning there have existed quite different models of DNA computing. Some models are first implemented in the laboratory, later modified and studied to make them computationally complete or to find operations that are more error-free and easier to carry out. On the other hand, much work has been done to find better ways to describe DNA operations and to take advantage of the existing knowledge of mathematics and molecular biology. Most of the work done is theoretical, but such a diversity of models gives a hope that the DNA computers can be constructed.

ACKNOWLEDGEMENTS

I would like to thank the staff of the Institute of Computer Science of the University of Tartu, especially Uno Kaljulaid, for useful advice and the anonymous referees for their comments.

REFERENCES

1. Adleman, L. Molecular computation of solutions to combinatorial problems. *Science*, 1994, **266**, 1021–1024.
2. Adleman, L. On constructing a molecular computer. In *DNA Based Computers: AMS Proceedings of a DIMACS workshop, Princeton, 1995* (Lipton, R. J. and Baum, E. B., eds.). *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, 1996, **27**, 1–22.
3. Boneh, D., Dunworth, C. and Lipton, R. Breaking DES using a molecular computer. In *DNA Based Computers: AMS Proceedings of a DIMACS workshop, Princeton, 1995* (Lipton, R. J. and Baum, E. B., eds.). *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, 1996, **27**, 37–65.
4. Boneh, D., Lipton, R., Dunworth, C. and Sgall, J. *On the Computational Power of DNA*. Technical Report TR-499-95, Princeton University, USA, 1995.
5. Lipton, R. J. DNA Solution of hard computational problems. *Science*, 1995, **268**, 542–545.
6. Beaver, D. A universal molecular computer. In *DNA Based Computers: AMS Proceedings of a DIMACS workshop, Princeton, 1995* (Lipton, R. J. and Baum, E. B., eds.). *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, 1996, **27**, 29–36.
7. Rothemund, P. and Wilhelm, P. A DNA and restriction enzyme implementation of Turing machines. In *DNA Based Computers: AMS Proceedings of a DIMACS workshop, Princeton, 1995* (Lipton, R. J. and Baum, E. B., eds.). *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, 1996, **27**, 75–120.
8. Head, T. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours. *Bull. Math. Biol.*, 1987, **49**, 737–759.
9. Freund, R., Kari, L. and Paun, G. DNA computing based on splicing: the existence of universal computers. *Theory Comput. Syst.*, 1999, **32**, 69–112.
10. Paun, G. On the splicing operation. *Discrete Appl. Math.*, 1996, **70**, 57–79.
11. Paun, G. Regular extended H systems are computationally universal. *J. Autom. Lang. Combin.*, 1996, **1**, 27–36.
12. Pixton, R. Splicing in abstract families of languages. In *SNAC Working Material. TUCS General Publ.*, 1997, **6**, 513–540.
13. Culik, K. and Harju, T. Splicing semigroups of dominoes and DNA. *Discrete Appl. Math.*, 1991, **31**, 261–277.
14. Dassow, J. and Mitrana, V. Splicing grammar systems. *Computers Artif. Intell.*, 1996, **15**, 109–122.
15. Georgescu, G. On the generative capacity of splicing grammar systems. *Lecture Notes in Comput. Sci.*, 1997, **1218**, 330–345.
16. Kari, L. DNA-computing: arrival of biological mathematics. *Math. Intelligencer*, 1997, **19**, 9–22.
17. Mateescu, A., Paun, G., Rozenberg, G. and Salomaa, A. Simple splicing systems. *Discrete Appl. Math.*, 1998, **84**, 145–163.
18. Paun, G., Rozenberg, G. and Salomaa, A. Computing by splicing. *Theor. Comput. Sci.*, 1996, **168**, 321–336.
19. Sakakibara, Y. and Ferretti, C. Splicing on tree-like structures. In *DNA Based Computers III: AMS Proceedings of a DIMACS workshop, Philadelphia, 1997* (Rubin, H. and Wood, D. H., eds.). *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, 1998, **48**, 361–371.
20. Laun, E. and Reddy, K. J. Wet splicing systems. In *DNA Based Computers III: AMS Proceedings of a DIMACS workshop, Philadelphia, 1997* (Rubin, H. and Wood, D. H., eds.). *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, 1998, **48**, 73–84.
21. Kari, L. and Thierrin, G. Contextual insertions/deletions and computability. *Inf. Comput.*, 1996, **131**, 47–61.
22. Kari, L., Paun, G., Thierrin, G. and Yu, S. At the crossroads of linguistics, DNA computing, and formal language theory: characterizing RE using insertion–deletion systems. In *DNA Based Computers III: AMS Proceedings of a DIMACS workshop, Philadelphia, 1997*

(Rubin, H. and Wood, D. H., eds.). *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, 1998, **48**, 329–346.
23. Yokomori, T. and Kobayashi, S. DNA-EC: a model of DNA-computing based on equality checking. In *DNA Based Computers III: AMS Proceedings of a DIMACS workshop, Philadelphia, 1997* (Rubin, H. and Wood, D. H., eds.). *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, 1998, **48**, 347–360.

FORMAALSED MUDELID DNA-ARVUTUSES: ÜLEVAADE

Tiina ZINGEL

DNA-arvutuseks nimetatakse meetodit, mis võimaldab matemaatilisi probleeme lahendada bioloogiliste operatsioonide abil, mida sooritatakse DNA ahelatel. Siinses artiklis on kirjeldatud DNA-arvutuse matemaatilisi ja bioloogilisi aluseid ning antud ülevaade olemasolevatest formaalsetest mudelitest. Lähemalt on vaadeldud osavahetussüsteeme, lisamis-kustutamissüsteeme ning hulkade võrduse kontrollil põhinevat mudelit DNA-EC.