ELSEVIER

Future Generation Computer Systems 17 (2001) 823–834

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.nl/locate/future

DNA computing in vitro and in vivo☆

Lila Kari*

Department of Computer Science, University of Western Ontario, London, Ont., Canada N6A 5B7

Abstract

This is a review paper addressing two main aspects of DNA computing research: DNA computing in vitro (in the test tube) and in vivo (in a living organism). We describe the first successful in vitro DNA computing experiment [L.M. Adleman, Science 266 (1994) 1021–1024] which solved a mathematical problem, the Directed Hamiltonian Path Problem, solely by manipulation of DNA strands in test tubes. We then address DNA computing in vivo by presenting a model proposed by Head [in: G. Rozenberg, A. Salomaa (Eds.), Lindenmayer Systems, Springer, Berlin, 1991, pp. 371–383] and also by Landweber and Kari [in: L. Kari, H. Rubin, D.H. Wood (Eds.), Biosystems, Vol. 52, Nos. 1–3, Elsevier, Amsterdam, 1999, pp. 3–13] and developed by Landweber and Kari [in: L.F. Landweber, E. Winfree (Eds.), Evolution as Computation, Springer, Berlin, 1999], for the homologous recombinations that take place during gene rearrangement in ciliates. Results given by Kari, Kari and Landweber [in: J. Karhumaki, H. Maurer, G. Paun, G. Rozenberg (Eds.), Jewels are Forever, Springer, Berlin, 1999, pp. 353–363] and Landweber and Kari [in: L.F. Landweber, E. Winfree (Eds.), Evolution as Computation, Springer, Berlin, 1999] have shown that a generalization of this model that assumes context-controlled recombinations has universal computational power. We review results obtained by Kari and Kari [in: Words, Sequences, Languages: Where Computer Science, Biology and Linguistics Meet, Kluwer Academic Publishers, The Netherlands, in press] on properties of context-free recombinations and characterize the languages generated by context-free recombination systems. As a corollary, we show [J. Kari, L. Kari, in: Words, Sequences, Languages: Where Computer Science, Biology and Linguistics Meet, Kluwer Academic Publishers, The Netherlands, in press], that context-free recombinations are computationally weak, being able to generate only regular languages.

Keywords: DNA computing; In vitro computing; In vivo computing

1. Introduction

DNA computing, known also under the name of *bio-molecular computing* or *molecular computing*, is a new computational paradigm which proposes the use of bio-molecules, mainly DNA, for computational purposes. The main idea behind DNA computing is based on two observations. Firstly, a naturally occurring DNA strand is a succession of four different building blocks, called bases, arranged in a certain order that depends on the organism to which the DNA belongs. While in nature the order in which the bases occur determines the genetic information dictating the type of organism and its development, these bases can also be chemically synthesized and combined in strands. This puts at our disposal a four-letter alphabet with which, instead of "writing" genetic information, we can "write" suitably encoded numbers, inputs, intermediate data and outputs of computations. Secondly, the molecular biology laboratory is a rich tool-chest of techniques that have been developed to

☆ Research partially supported by Grant R2824AO1 of the Natural Sciences and Engineering Research Council of Canada.

* Website address: www.csd.uwo.ca/∼lila.
E-mail address: lila@csd.uwo.ca (L. Kari).

0167-739X/01/$ – see front matter
PII: S0167-739X(00)00061-3

manipulate and recombine DNA strands. Recently, it has been shown that these tools can also be used to simulate and perform computational processes.

This is a review paper mostly based on [10,11] that aims to give an overview of DNA computing by addressing two of the main aspects of DNA computing research: DNA computing in vitro and in vivo. The paper is organized as follows. Section 2 briefly introduces basic molecular biology notions like the structure of DNA and the main laboratory techniques used in DNA computing. Section 3 describes Adleman's in vitro (test tube) experiment [1] solving the Directed Hamiltonian Path Problem by manipulations of DNA strands. This section also explains in more detail some of the laboratory techniques that have been used in this experiment. Section 4 introduces the problem of in vivo (in the living organism) DNA computing by describing the process of gene unscrambling in ciliates [14] and its computational potential. The process entails inter- and intra-molecular recombination of DNA strands for which a formal model is defined [6,14]. It is then shown that, under some additional assumptions, a computational model based on these recombination operations, called *guided recombination system*, has the computational power of a Turing machine [15]. These assumptions consist of hypothesizing that the presence of certain control sequences, called contexts, is necessary to determine whether or not recombinations take place.

The presence of contexts that guide recombination, while consistent with the biological data, is not a certainty. In fact, the exact details of the mechanism of gene rearrangement in ciliates are still unknown [21]. Section 5 drops these assumptions and considers computational systems where recombinations are not controlled by contexts [11]. Besides studying properties of these recombinations, the main result of this section proves that such systems are computationally weak, having the power to generate only regular languages. This is one more indicator that most probably some kind of control factor, be it presence of contexts or something else, is necessary for correct recombination during the gene unscrambling process.

In any case, the results in Sections 4 and 5 demonstrate the computational capacity of in vivo DNA computing, which might potentially be harnessed for our own computational needs. Finally, Section 6 presents some arguments in favour of using DNA rather than electronic computers for some computational purposes.

2. Molecular biology notions

DNA (deoxyribonucleic acid) is found in every cellular organism as the storage medium for genetic information. It is composed of units called nucleotides, distinguished by the chemical group, or base, attached to them. The four bases are *adenine*, *guanine*, *cytosine* and *thymine*, abbreviated as A, G, C, and T. (The names of the bases are also commonly used to refer to the nucleotides that contain them.) Single nucleotides are linked together end-to-end to form DNA strands. A short single-stranded polynucleotide chain, usually less than 30 nucleotides long, is called an *oligonucleotide* (or, shortly, oligo). The DNA sequence has a *polarity*: a sequence of DNA is distinct from its reverse. The two distinct ends of a DNA sequence are known under the name of the $5'$ end and the $3'$ end, respectively. Taken as pairs, the nucleotides A and T, and the nucleotides C and G are said to be *complementary*. Two complementary single-stranded DNA sequences with opposite polarity will join together to form a double helix in a process called *base-pairing* or *annealing*. The reverse process — a double helix coming apart to yield its two constituent single strands — is called *melting* [10].

A single strand of DNA can be likened to a string consisting of a combination of four different symbols: A, G, C, T. Mathematically, this means we have at our disposal a four-letter alphabet $X=\{A,G,C,T\}$ to encode information. Concerning the operations that can be performed on DNA strands, the existing models of DNA computation are based on various combinations of the following primitive *bio-operations* [10]:

- *Synthesizing* a desired polynomial-length strand.
- *Mixing*. Pour the contents of two test tubes into a third.
- *Annealing*(*hybridization*). Bond together two single-stranded complementary DNA sequences by cooling the solution.
- *Melting*(*denaturation*). Break apart a double-stranded DNA into its single-stranded components by heating the solution.
- *Amplifying*(*copying*). Make copies of DNA strands by using the polymerase chain reaction (PCR).

- *Separating* the strands by size using a technique called gel electrophoresis.
- *Extracting* those strands that contain a given pattern as a substring by using affinity purification.
- *Cutting* DNA double-strands at specific sites by using commercially available restriction enzymes.
- *Ligating*. Paste DNA strands with compatible sticky ends by using DNA ligases.
- *Substituting*. Substitute, insert or delete DNA sequences by using PCR site-specific oligonucleotide mutagenesis.
- *Detecting and reading* a DNA sequence from a solution.

The bio-operations listed above and possibly others will then be used to write *molecular programs* which receive a tube containing DNA strands as input and return as output a set of tubes [2]. A computation consists of a sequence of tubes containing DNA strands.

For further details of molecular biology terminology, the reader is referred to [4,10,13].

3. Adleman's in vitro DNA algorithm

The practical possibilities of encoding information in a DNA sequence and of performing simple bio-operations were used in [1] to solve a seven-node instance of the Directed Hamiltonian Path Problem by an in vitro DNA experiment. A directed graph G with designated vertices v_{in} and v_{out} is said to have a Hamiltonian path if and only if there exists a sequence of compatible "one-way" edges $e_1, e_2, \ldots, e_z$ (i.e. a path) that begins at v_{in}, ends at v_{out} and enters every other vertex exactly once.

The following (nondeterministic) algorithm solves the problem:

Step 1. Generate random paths through the graph.
Step 2. Keep only those paths that begin with v_{in} and end with v_{out}.
Step 3. If the graph has n vertices, then keep only those paths that enter exactly n vertices.
Step 4. Keep only those paths that enter all of the vertices of the graph at least once.
Step 5. If any paths remain, say "YES"; otherwise say "NO".

To implement Step 1, each vertex of the graph was encoded into a random 20-nucleotide strand (20-letter sequence) of DNA that was *synthesized*. Then, for each (oriented) edge of the graph, a DNA sequence was synthesized consisting of the second half of the sequence encoding the source vertex and the first half of the sequence encoding the target vertex.

Synthesis of a desired DNA strand is accomplished as follows. In standard solid-phase DNA synthesis, a desired DNA molecule is built up nucleotide by nucleotide on a support particle in sequential coupling steps. For example, the first nucleotide (monomer), say A, is bound to a glass support. A solution containing C is poured in, and A reacts with C to form a two-nucleotide (2-mer) chain AC. After washing the excess C solution away, one could have the C from the chain AC coupled with T to form a 3-mer chain (still attached to the surface) and so on [10].

Returning to Step 1 of the algorithm, by *mixing* together single strands encoding the edges and single strands encoding complements of vertices, DNA sequences corresponding to compatible edges were linked together.

Mixing is implemented by pouring the contents of two test tubes into a third one to achieve union. Mixing can be performed by rehydrating the tube contents (if not already in solution) and then combining the fluids together into a new tube, by pouring and pumping for example.

Linking together of the compatible edges was realized as follows. By construction, a complement of a vertex strand would bind to both a strand encoding an edge entering the vertex, and a strand encoding an edge exiting the vertex. Hence, the *ligation reaction* resulted in the formation of DNA molecules encoding random paths through the graph. In this process, both *annealing* and *ligation* played a role.

Annealing consists in binding together two single-stranded complementary DNA sequences by cooling the solution. Annealing in vitro is also known as *hybridization*.

Ligation amounts to pasting DNA strands with compatible sticky ends by using DNA ligase. A DNA double-strand can either have blunt ends, i.e. be fully double-stranded or can be partially double-stranded, i.e. it can have single-stranded overhanging ends (called sticky ends) at one or both of its extremities. The enzyme *DNA ligase* joins together, or ligates, the end of a DNA strand to another strand. DNA ligase

either ligates two blunt-ended double strands or two strands with compatible sticky ends [13].

To implement Step 2, the product of Step 1 was amplified by PCR. Thus, only those molecules encoding paths that begin with v_{in} and end with v_{out} were amplified.

Amplifying (*copying*) consists in making copies of DNA strands by using the PCR that uses the *DNA polymerase* enzyme. The *DNA polymerases* perform several functions including replication of DNA. The replication reaction requires a guiding DNA single strand called *template* and a shorter oligonucleotide called *primer* that is annealed to it. Under these conditions, DNA polymerase catalyzes DNA synthesis by successively adding nucleotides to one end of the primer. The primer is thus extended in one direction until the desired strand that starts with the primer and is complementary to the template is obtained [10].

PCR is an in vitro method that relies on DNA polymerase to quickly amplify specific DNA sequences in a solution. PCR involves a repetitive series of temperature cycles with each cycle comprising three stages: denaturation of the guiding template DNA to separate its strands, then cooling to allow annealing to the template of the primer oligonucleotides, which are specifically designed to flank the region of DNA of interest, and finally, extension of the primers by DNA polymerase. Each cycle of the reaction doubles the number of target DNA molecules, the reaction giving thus an exponential growth of their number [10].

For implementing Step 3, a technique called *gel electrophoresis* was used, that makes possible the separation of DNA strands by length. The molecules are placed at the top of a wet gel to which an electric field is applied, drawing them to the bottom. Larger molecules travel more slowly through the gel. After a period, the molecules spread out into distinct bands according to size.

Step 4 was accomplished by iteratively using a process called *affinity purification*. This process permits single strands containing a given subsequence v (encoding a vertex of the graph) to be filtered out from a heterogeneous pool of other strands. After synthesizing strands complementary to v and attaching them to magnetic beads, the heterogeneous solution is passed over the beads. Those strands containing v anneal to the complementary sequence and are retained. Strands not containing v pass through without being retained [10].

To implement Step 5, the presence of a molecule encoding a Hamiltonian path was checked. This was done by amplifying the result of Step 4 by PCR and then determining the DNA sequence of the amplified molecules.

Detecting and *reading* (given the contents of a tube), consist in saying YES if it contains at least one DNA strand, and NO otherwise. PCR may be used to amplify the result and then a process called *sequencing* is used to actually read the DNA strands in solution. The basic idea of the most widely used sequencing method is to use PCR and gel electrophoresis. Assume we have a homogeneous solution, i.e. a solution containing mainly copies of the strand we wish to sequence, and very few contaminants (other strands). For detection of the positions of As in the target strand, a blocking agent is used that prevents the templates from being extended beyond As during PCR. As a result of this modified PCR, a population of subsequences is obtained, each corresponding to a different occurrence of A in the original strand. Separating them by length using gel electrophoresis reveals the positions where A occurs in the strand. The process can then be repeated for each of C, G, and T, to yield the sequence of the strand. Recent methods use four different fluorescent dyes, one for each base, which allows all four bases to be processed simultaneously. As the fluorescent molecules pass a detector near the bottom of the gel, data are output directly to an electronic computer [10].

4. In vivo DNA computing

The preceding section presented an example of in vitro DNA computing. This and the following section address the issue of in vivo DNA computing. We namely describe a formal system intended to model the guided homologous recombinations that take place during gene rearrangement in ciliates, and study the computational power of such a system.

Cilliates are a diverse group of a few thousand types of unicellular eukaryotes (nucleated cells) that emerged more than 10^9 years ago [21]. Despite their diversity, ciliates remain united by two features: the possession of a hair-like cover of cilia used for moving and food capture, and the presence of two nuclei

[21]. The *micronucleus* is functionally inert and becomes active only during sexual exchange of DNA, while the active *macronucleus* contains the genes needed for the development of the ciliate. When two cells mate, they exchange micronuclear information and afterwards develop new macronuclei from their respective micronuclei.

In some of the few studied ciliates, the protein-coding segments of the genes (or MDSs for macronuclear destined sequences) are present also in the micronucleus interspersed with large segments of noncoding sequences (IESs for internally excised sequences). Moreover, these segments are present in a permuted order in the micronucleus. The function of the various eliminated sequences is unknown and they represent a large portion of the micronuclear sequences: in the *Oxytricha* species ∼96% and in *Stylonychia lemnae* ∼98% of the micronuclear sequences are eliminated [21].

As an example, the micronuclear encoding of the αTBP gene in *Oxytricha nova* is composed of 14 MDSs separated by IESs and arranged in the order 1–3–5–7–9–11–2–4–6–8–10–12–13–14, the proper order being defined by 1–14 arrangement of the spliced MDSs in the functional macronuclear gene [16,21].

Instructions for unscrambling the micronuclear gene are apparently carried in the gene itself [21]. At the end of each ith MDS ($1 \leq i \leq 13$) is a sequence of 6–19 b.p. that is identical to a sequence preceding the $(i+1)$th MDS (which occurs somewhere else in the gene). Homologous recombinations between these pairs of direct repeats (called also junction sequences) will then join the MDSs in the correct order and eliminate one member of each repeat pair.

Before introducing the formal model of the recombinations that take place during gene rearrangement, we summarize our notation. An alphabet Σ is a finite, nonempty set. A sequence of letters from Σ is called a string (word) over Σ and in our interpretation corresponds to a linear strand. The length of a word w is denoted by $|w|$ and represents the total number of occurrences of letters in the word. A word with zero letters in it is called an empty word and is denoted by λ. The set of all possible words consisting of letters from Σ is denoted by Σ^*, and the set of all nonempty words by Σ^+. We also define circular words over Σ by declaring two words to be equivalent if and only if (iff) one is a cyclic permutation of the other. In other words, w is equivalent to w' iff they can be decomposed as $w=uv$ and $w'=vu$, respectively. Such a circular word $\bullet w$ refers to any of the circular permutations of the letters in w. Denote by $\Sigma\bullet$ the set of all circular words over Σ.

For a linear word $w \in \Sigma^*$, $\mathrm{Pref}(w)=\{x \in \Sigma^* | w=xv\}$, $\mathrm{Suff}(w)=\{y \in \Sigma^* | w=uy\}$ and $\mathrm{Sub}(w)=\{z \in \Sigma^* | w=uzv\}$.

For a circular word $\bullet w \in \Sigma\bullet$, we define $\mathrm{Pref}(\bullet w)=\mathrm{Suff}(\bullet w)=\emptyset$ and

$$\mathrm{Sub}(\bullet w) = \{x \in \Sigma^* | \bullet w = \bullet uxv,\ u, v \in \Sigma^*\}$$

as the set of prefixes and suffixes, respectively, subwords of $\bullet w$.

For more notions of formal language theory the reader is referred to [22]. With this notation we introduce several operations studied in [14,15] in the context of gene unscrambling in ciliates.

Definition 1. If $x \in \Sigma^+$ is a junction sequence, then the recombinations guided by x are defined as follows:

(1) $uxv+u'xv' \Rightarrow uxv'+u'xv$ (linear/linear) (Fig. 1),
(2) $uxvxw \Rightarrow uxw+\bullet vx$ (linear/circular) (Fig. 2),
(3) $\bullet uxv + \bullet u'xv' \Rightarrow \bullet uxv'u'xv$ (circular/circular) (Fig. 3).

Note that all recombinations in Definition 1 are reversible, i.e. the operations can be performed also in the opposite directions.

For example, operation (2) in Fig. 2 models the process of intramolecular recombination. After x finds its second occurrence in $uxvxw$, the molecule undergoes a strand exchange in x that leads to the formation of two new molecules: uxw and a circular DNA molecule $\bullet vx$. Intramolecular recombination accomplishes the deletion of either sequence vx or xv from the original molecule $uxvxw$ and the positioning of w immediately next to ux. This implies that (2) can be used to rearrange sequences in a DNA molecule thus accomplishing gene unscrambling.

The above operations are similar to the "splicing operation" introduced by Head [6] and "circular splicing" and "mixed splicing" [7,18–20,23]. Refs. [5,17] and subsequently Ref. [24] showed that some of these models have the computational power of a universal Turing machine (see [8] for a review).

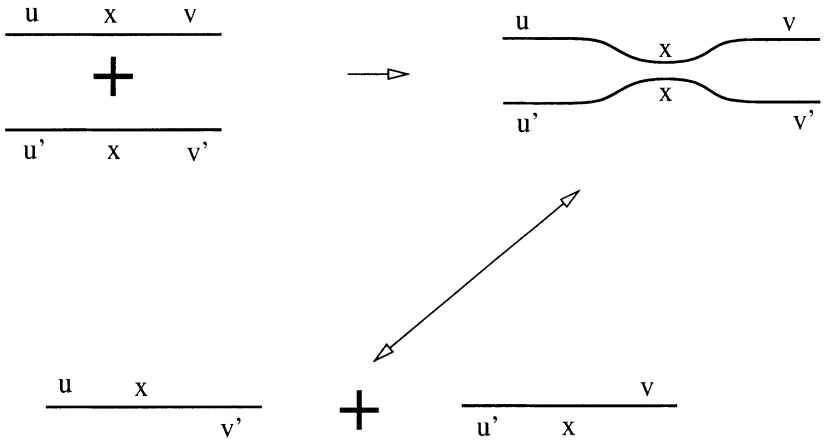

Fig. 1. Linear/linear recombination (from Ref. [11]).

To return to the biological reality we are modelling, the question of how the cell discriminates between sequences to be destroyed and sequences to be preserved is still unanswered [21]. Existing data suggest that the presence of direct repeats alone is not enough to guide the unscrambling process. Instructions for unscrambling the genes are seemingly carried by the gene itself [21] and they might be either structural "instructions" or control factors like the presence of certain sequences in the vicinity of the direct repeats. Indeed, in the coded αTBP gene in *O. nova*, the model proposed by Mitcham et al. [16] conjectures a spiral-like arrangement of the micronuclear strand which would allow the physical alignment of the pairs of direct repeats that guide the splicing in correct order. In other cases, the excision of transposon-like elements (which may qualify as IESs) is influenced by the presence of certain sequences flanking the IES.

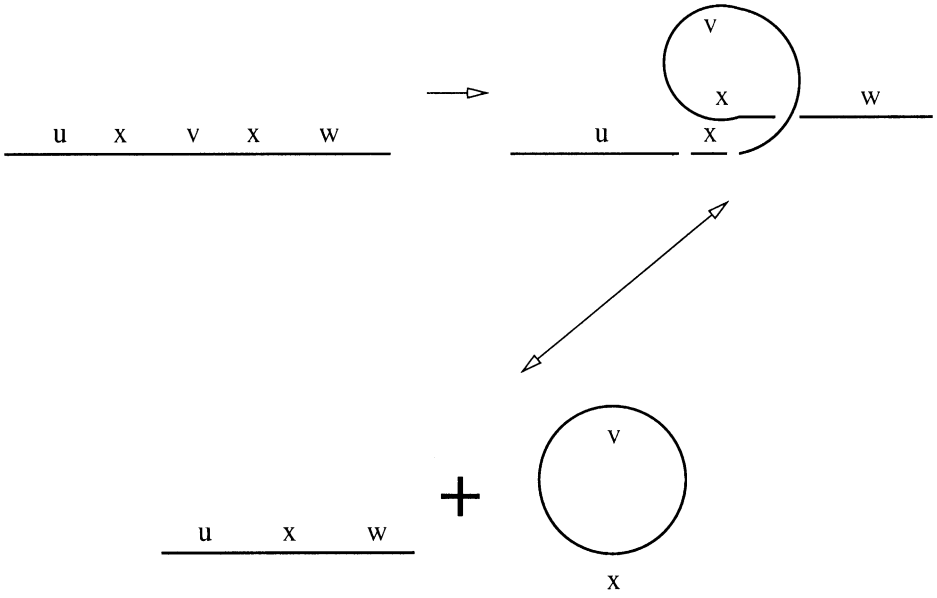

Fig. 2. Linear/circular recombination (from Ref. [11]).

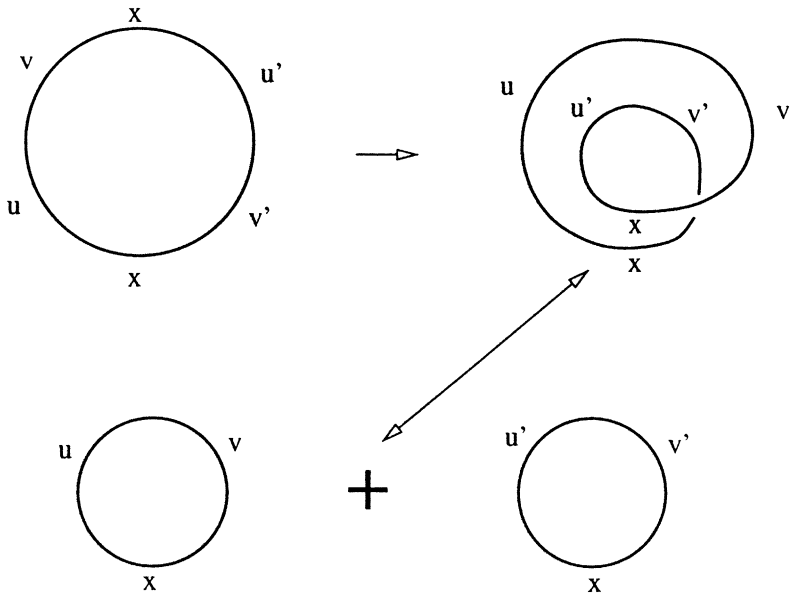

Fig. 3. Circular/circular recombination (from Ref. [11]).

For example, in *Tetrahymena* and *E. crassus* species, within the micronuclear DNA, the two ends of the sequence to be eliminated are immediately flanked by a specific 5-mer sequence which is a crucial element of recognition [9,21].

Taking the latter type of "control" of recombination into account, in [15] the strand operations in Definition 1 were generalized by assuming that homologous recombination is influenced by the presence of certain contexts. In particular, in [15] we defined the notion of a *guided recombination system* based on operation (2), Definition 1, with the additional constraint that certain contexts need to be present in order for recombination to occur. The main result in [12,15] proved that such systems have the computational power of a Turing machine, the most widely used theoretical model of electronic computers. This implies that, in principle, these unicellular organisms may have the computational power of an electronic computer.

5. Context-free recombinations

The previous section discussed a computational system modelling the recombinations taking place during gene rearrangement in ciliates that makes the assumption that recombinations are influenced by the presence of contexts flanking the recombination sites. However, this is not a certainty, but an assumption which seems consistent with the biological data. In fact, the exact details of the molecular machinery that accomplishes unscrambling are unknown [21].

In this section, we review results in [11] that consider the case where all types of recombinations (linear/linear, linear/circular, circular/circular) are allowed and moreover no context restrictions apply. We study properties of such recombinations. This study complements results obtained in [8,18–20,23] on linear splicing, circular splicing, self-splicing and mixed splicing. The results obtained that the computational power of context-free recombination systems is very weak (which is unlikely), strengthen the conjecture that the presence of direct repeats is insufficient for accurate splicing during gene unscrambling.

In our intuitive image of context-free recombinations, we can view strings as cables or "extension cords" with different types of "plugs". Given a set of junction sequence J, each $x \in J$ defines one type of "plug". Strings, both linear and circular, can then be viewed as consisting of "elementary" cables that only have plugs at their extremities. A circular strand consists of elementary cables connected to form a

loop. A recombination step amounts to the following operations: take two connections using identical plugs (the connections can be in two different cables or in the same cable); unplug them; cross-plug to form new cables.

In view of Lemma 3, we will assume, without loss of generality, that all sets of plugs J are subword-free.

Definition 2 (Kari and Kari [11]). Let $J \subseteq \Sigma^+$ be a set of plugs. We define the set of elementary cables (left elementary cables and right elementary cables, respectively) with plugs in J as

$$E_J = (J\Sigma^+ \cap \Sigma^+ J)\backslash \Sigma^+ J \Sigma^+,$$

$$L_J = \Sigma^* J \backslash \Sigma^* J \Sigma^+,$$

$$R_J = J\Sigma^* \backslash \Sigma^+ J \Sigma^*.$$

Note that an elementary cable in E_J is of the form $z_1 u = v z_2$, where $z_1, z_2 \in J$ are plugs. In other words, an elementary cable starts with a plug, ends with a plug, and contains no other plugs as subwords. The start and end plug can overlap.

A left elementary cable is of the form wz, where $z \in J$ is a plug and wz does not contain any other plug as a subword. In other words, if we scan wz from left to right, z is the first plug we encounter.

Analogously, a right elementary cable is of the form zw, where $z \in J$ is a plug and wz does not contain any other plug as a subword.

Definition 3 (Kari and Kari [11]). For a set of plugs $J \subseteq \Sigma^+$ and a linear word $w \in \Sigma^+$, the set of elementary cables with plugs in J occurring in w is defined as

$$E_J(w) = E_J \cap \mathrm{Sub}(w),$$

while the set of left and right elementary cables occurring in w are respectively:

$$L_J(w) = L_J \cap \mathrm{Pref}(w),$$

$$R_J(w) = R_J \cap \mathrm{Suff}(w).$$

Note that $L_J(w)$ and $R_J(w)$ are both singleton sets.

Example 1 (Kari and Kari [11]). If $\Sigma = \{a,b\}$ and $J = \{b\}$, then $L_J(aba) = ab$, $R_J(aba) = ba$, $E_J(aba) = \emptyset$. Also, $L_J(ab) = ab$, $R_J(ab) = b$, $E_J(ab) = \emptyset$ and $L_J(ba) = b$, $R_J(ba) = ba$, $E_J(ba) = \emptyset$.

Definition 4 (Kari and Kari [11]). For a set of plugs $J \subseteq \Sigma^+$ and a circular word $\bullet w \in \Sigma \bullet$, we define the elementary cables occurring in $\bullet w$ as follows:

(1) If $\exists x \in J \cap \mathrm{Sub}(\bullet w)$, the elementary cables with plugs in J occurring in $\bullet w$ are defined as $E_J(\bullet w) = E_J(www)$, $L_J(\bullet w) = R_J(\bullet w) = \emptyset$.
(2) If $J \cap \mathrm{Sub}(\bullet w) = \emptyset$, then $E_J(\bullet w) = L_J(\bullet w) = R_J(\bullet w) = \emptyset$.

Example 2 (Kari and Kari [11]). If $\Sigma = \{a,b\}$ and $J = (aba, baa)$, then $E_J(\bullet aba) = \{abaa, baaba\}$.

If $\Sigma = \{a,b,c,d\}$ and $J = \{abc, bcdab\}$, then $E_J(\bullet abcd) = \{abcdab, bcdabc\}$.

From the above examples, we see that in circular words, start and end plugs are allowed to overlap.

The definitions for elementary cables, left and right elementary cables can be easily generalized to languages. For a language $L \subseteq \Sigma^* \cup \Sigma\bullet$,

$$E_J(L) = \bigcup_{w \in L} E_J(w), \qquad L_J(L) = \bigcup_{w \in L} L_J(w),$$

$$R_J(L) = \bigcup_{w \in L} R_J(w).$$

The following two lemmas introduces some properties of elementary cables.

Lemma 1 (Kari and Kari [11]). *Given a set of plugs* $J \subseteq \Sigma^+$,

(1) *If* $u, v \in \Sigma^*$ *and* $u \in \mathrm{Sub}(v)$, *then* $E_J(u) \subseteq E_J(v)$.
(2) *If* $u \in \mathrm{Pref}(v)$, *then* $L_J(u) = L_J(v)$.
(3) *If* $u \in \mathrm{Suff}(v)$, *then* $R_J(u) = R_J(v)$.

Lemma 2 (Kari and Kari [11]). *If* $x \in J$ *is a plug, then*

(1) $E_J(uxv) = E_J(\{ux, xv\})$,
(2) $E_J(\bullet ux) = E_J(xux)$,
(3) $L_J(uxv) = L_J(ux)$,
(4) $R_J(uxv) = R_J(xv)$.

The proposition below shows that recombination of cables does not produce additional elementary cables, i.e. the set of the elementary cables of the result string equals the set of elementary cables of the strings entering recombination.

Proposition 1 (Kari and Kari [11]). *If* $J \subseteq \Sigma^+$ *is a set of plugs and* $x \in J$, *then*

(1) $E_J(uxvxw) = E_J(uxw) \cup E_J(\bullet vx)$,

(2) $E_J(\{uxv, u'xv'\})=E_J(\{uxv', u'xv\})$,
(3) $E_J(\{\bullet uxv, \bullet u'xv'\})=E_J(\{\bullet uxv'u'xv\})$,
(4) $L_J(uxvxw)=L_J(\{uxw, \bullet vx\})$,
(5) $L_J(\{uxv, u'xv'\})=L_J(\{uxv', u'xv\})$,
(6) $R_J(uxvxw)=R_J(\{uxw, \bullet vx\})$,
(7) $R_J(\{uxv, u'xv'\})=R_J(\{uxv', u'xv\})$.

We are now ready to define the notion of a context-free recombination system. This is a construction whereby we are given a starting set of sequences and a list of junction sequences (plugs). New strings may be formed by recombinations among the existing strands: if one of the given junction sequences is present, recombinations are performed as defined in Definition 1. Recombinations are context-free, i.e. they are not dependent on the context in which the junction sequences appear. The language of the system is defined as the set of all strands that can be thus obtained by repeated recombinations starting from the initial set.

Definition 5 (Kari and Kari [11]). A context-free recombination system is a triple

$$R = (\Sigma, J, A),$$

where Σ is an alphabet and $J\subseteq\Sigma^+$ is a set of plugs, while $A\subseteq\Sigma^+ \cup \Sigma\bullet$ is the set of axioms of the system.

Given a recombination system R, for sets S, $S'\subseteq\Sigma^+ \cup \Sigma\bullet$, we say that S derives S' and we write $S \Rightarrow_R S'$ iff there exists $x\in J$ such that one of the following situations holds:

(1) $\exists uxv,\ u'xv'\in S$ such that $uxv+u'xv'\Rightarrow uxv'+v'xv$ and $S'=S\cup\{uxv', u'xv\}$,
(2) $\exists uxvxw\in S$ such that $uxvxw\Rightarrow uxw+\bullet vx$ and $S'=S\cup\{uxw,\bullet vx\}$,
(3) $\exists uxw,\ \bullet vx\in S$ such that $uxw+\bullet vx\Rightarrow uxvxw$ and $S'=S\cup\{uxvxw\}$,
(4) $\exists\bullet uxv,\ \bullet u'xv'\in S$ such that $\bullet uxv+\bullet u'xv'\Rightarrow \bullet uxv'u'xv$ and $S'=S\cup\{\bullet\ uxv'u'xv\}$,
(5) $\exists\bullet uxv'u'xv\in S$ such that $\bullet uxv'u'xv\Rightarrow\bullet uxv+\bullet u'xv'$ and $S'=S\cup\{\bullet uxv,\bullet u'xv'\}$.

Definition 6 (Kari and Kari [11]). The language generated by a context-free recombination system R is defined as

$$L(R) = \{w \in \Sigma^* \cup \Sigma_\bullet | A \Rightarrow_R {}^*S,\ w \in S\}.$$

Lemma 3 (Kari and Kari [11]). *For any context-free recombination system $R=(\Sigma, J, A)$, there exists a context-free recombination system $R'=(\Sigma, J', A)$ such that J' is subword-free and $L(R)=L(R')$.*

As a consequence of the preceding lemma we may assume, without loss of generality, that a context-free recombination system has a subword-free set J of plugs. The following lemma will aid in the proof of our main result.

Lemma 4 (Kari and Kari [11]). *Let $R=(\Sigma, J, A)$ be a context-free recombination system. Let $ux=x'u'$ start and end with plugs $x', x\in J$, where $u, u'\neq\lambda$. If ux satisfies $E_J(ux)\subseteq E_J(A)$, then there exist $\alpha, \beta\in\Sigma^*$ such that $\alpha ux\beta$ or $\bullet\alpha ux\beta$ is in $L(R)$.*

The theorem below shows that a context-free recombination system characterized by a set of plugs J and a set of axioms A has the following property. Any cable that consists of elementary cables plugged together after each other and that is either linear or circular can be obtained from the axioms using cross-plugging. Conversely, no other types of cables can be obtained from the axioms.

Theorem 1 (Kari and Kari [11]). *Let $R=(\Sigma,J,A)$ be a context-free recombination system. Then $L(R)=X$, where*

$X=\{w\in\Sigma^* \cup \Sigma\bullet |$ either $E_J(w)=L_J(w)=R_J(w)=\emptyset$ and $w\in A$ or $E_J(w)$, $L_J(w)$, $R_J(w)$ are not all empty and $E_J(w)\subseteq E_J(A)$, $L_J(w)\subseteq L_J(A)$, $R_J(w)\subseteq R_J(A)\}$.

Proof. "$X\subseteq L(R)$". Let $w\in X$. If $E_J(w)=L_J(w)=R_J(w)=\emptyset$ and $w\in A$, then $w\in L(R)$. Assume now that $w\in\Sigma^*$, is a linear word such that $E_J(w)$, $L_J(w)$, $R_J(w)$ are not all empty and $E_J(w)\subseteq L_J(A)$, $L_J(w)\subseteq L_J(A)$, $R_J(w)\subseteq R_J(A)$. If w contains only one plug $x\in J$, then $w=uxv$ and $L_J(w)=ux$, $R_J(w)=xv$. As $L_J(w)\subseteq L_J(A)$, there exists an axiom $a_1\in A\cap\Sigma^*$ such that $a_1=uxt$. As $R_J(w)\subseteq R_J(A)$, there exists an axiom $a_2\in A\cap\Sigma^*$ such that $a_2=sxv$. We have $a_1+a_2=uxt+sxv\Rightarrow uxv+sxt$, which implies that $uxv=w\in L(R)$. If w contains more than one plug, then $w=u\gamma v$, where $\gamma=xl=rx'$, $x,x'\in J$, $ux=L_J(w)\subseteq L_J(A)$ and $x'v=R_J(w)\subseteq R_J(A)$. Consequently, there exist axioms $a_1,a_2\in A\cap\Sigma^*$ such that $a_1=uxt$, $a_2=sx'v$. By Lemma 4, there exist $\alpha,\beta\in\Sigma^*$ such that $\alpha\gamma\beta$ or $\bullet\alpha\gamma\beta$ is in $L(R)$. We can then

recombine

$$uxt + \alpha\gamma\beta + sx'v = uxt + \alpha xl\beta + sx'v \Rightarrow uxl\beta$$
$$+\alpha xt + sx'v = urx'\beta + \alpha xt + sx'v \Rightarrow urx'v$$
$$+\alpha xt + sx'\beta = u\gamma v + \alpha xt + sx'\beta,$$

or, in the circular case,

$$uxt + \bullet\alpha\gamma\beta + sx'v = uxt + \bullet\alpha xl\beta + sx'v \Rightarrow uxl\beta\alpha xt$$
$$+sx'v = urx'\beta\alpha xt + sx'v \Rightarrow urx'v$$
$$+sx'\beta\alpha xt = u\gamma v + sx'\beta\alpha xt,$$

In both cases, $u\gamma v = w \in L(R)$. If $\bullet w \in \Sigma\bullet$ is a circular word that contains at least one plug ($E_J(\bullet w) \neq \emptyset$) then $\bullet w = \bullet ux$ for some $x \in J$. The word xux satisfies the conditions of Lemma 4 therefore $axux\beta$ or $\bullet\alpha xux\beta$ is in $L(R)$. Then we have either $\alpha xux\beta \Rightarrow \bullet ux + \alpha x\beta$ or $\bullet\alpha xux\beta \Rightarrow \bullet ux + \bullet\alpha x\beta$, which both imply that $\bullet ux = \bullet w \in L(R)$. For the converse inclusion "$L(R) \subseteq X$", note that if $w \in L(R)$, $E_J(w) = \emptyset$, $L_J(w) = \emptyset$, $R_J(w) = \emptyset$, and $w \in A$, then by definition $w \in X$. Otherwise, if some words in $L(R)$ belong in X, the result of their recombinations have the necessary properties that ensure their belonging to X by Proposition 1. Therefore, $L(R) \subseteq X$. □

The theorem above leads to the conclusion of this section after we show that the language X is regular being accepted by a finite automation.

Definition 7 (Kari and Kari [11]). Given a finite automaton F, the circular language accepted by F denoted by $L(F)\bullet$ is defined as the set of all words $\bullet w$ such that F has a cycle labelled by w.

The circular/linear language accepted by a finite automaton F is defined as $L(F) \cup L(F)\bullet$, where $L(F)$ is the linear language accepted by a finite automaton F defined in the usual way.

Definition 8 (Kari and Kari [11]). A circular/linear language $L \subseteq \Sigma^* \cup \Sigma\bullet$ is called regular if there exists a finite automaton F such that F accepts the circular and linear components of L, i.e. that accepts $L \cap \Sigma^*$ and $L \cap \Sigma\bullet$.

Theorem 2 (Kari and Kari [11]). *Let $J \subseteq \Sigma^*$ be a set of plugs and let $A \subseteq \Sigma^* \cup \Sigma\bullet$ be a finite axiom set. Then the set X defined as in Theorem 1 equals the linear/circular language accepted by a finite automaton F.*

Proof. If the set J is not finite, then we start by eliminating plugs that do not appear in any elementary cables of A. As the axiom set A is finite, the number of elementary cables is finite, and the set of (useful) plugs is finite as well. Consequently, we can assume, without loss of generality, that the set J is finite.

Let $F = (S, \Sigma, \delta, s_0, s_f)$ be a finite automaton constructed as follows.

The set of states is

$$S = \{s_x | x \in J\} \cup \{s_0, s_f\},$$

and the transition relation δ is defined as follows:

(1) $\delta(s_x, u) = (s_y |$ for each $e = xu = vy \in E_J(A)\}$,
(2) for each $ux \in L_J(A)$, we have the transition $\delta(s_0, ux) = s_x$,
(3) for each $xu \in R_J(A)$, we have $\delta(s_x, u) = s_f$.

From the above construction and Theorem 1, one can prove that $L(F) = X$. □

Theorem 2 shows that the context-free recombination systems are computationally weak, having only the power to generate regular languages. This is one more indicator that, most probably, the presence of direct repeats does not provide all the information needed for accurate splicing during gene rearrangement in ciliates.

6. Conclusions

Besides the novelty of the approach, and in spite of the technical difficulties that arise from the error rates of bio-operations [3,10], there are several reasons why computing with DNA might have advantages over electronic computing. These include memory capacity, massive parallelism, and power requirements.

Indeed, 1 μmol of DNA in 1 l of water contains about $6.02 \times 10^{17} \approx 10^{18}$ strands (Avogadro's number). If we consider every strand as a processor, and that operations take several minutes, say 1000 s, then such a DNA-based computer would execute 10^{15} operations per second. In comparison, a modern 500 MHz CPU with superscalar design executes on the order of 10^9 instructions per second. The Intel ASCI Teraflops Computer currently being built by the United States

Department of Energy's Sandia National Laboratory and Intel Corporation (which is as big as a good-sized starter home, weights about 44 ton and requires 300 ton of air conditioning to cool it) can achieve a peak performance of 1.8 teraflops (trillion floating point operations per second) which is about 10^{12} operations per second. This means that, due to its massive parallelism, a DNA computer could be between a million times and a thousand times faster.

Considering the storage capacity, 1 μmol of DNA has 10^{18} strands of DNA. If each strand has length 40, we can assume that each encodes 10 bytes. The total storage space is thus 10^{19} bytes per liter of dilute solution. Recently, a team of scientists and engineers from IBM, which develops, manufactures and sells data-storage products have achieved a density of 35 gigabits which means 4.375 GB or 4×10^9 bytes per square inch. To encode the same information that can be stored in 1 μmol of DNA using this technology, one would need a surface of 2.5×10^9 in.^{2} which is approximately 160 ha or 400 acres. The dilute DNA solution containing the same data fits in a milk carton.

Concerning the power requirements, a laptop CPU uses about 1 W and executes around 1 billion instructions per second achieving thus 1 billion operations per Joule. In [1], it is estimated that a DNA based medium has the potential to perform up to 20 billion billions operations per Joule. This is an idealized number and probably does not include the energy required to raise and lower temperatures, to mix solutions, to run equipment, etc. However, even if the real number is a million times larger, the DNA computer is still 1000 times more energy efficient.

The comparisons above, while based on preliminary data, give a glimpse into why bio-molecules might be a preferred medium for computations in some applications. It is envisaged that the in vitro and the in vivo DNA computing research of the kinds described in this paper are preliminary steps that will ultimately lead to making DNA computing a viable complementary tool for computation and/or provide more insight into the computational capacity of live organisms.

Acknowledgements

We thank Sorin Drăghici and Eric VanDerLoo for data and discussion on the comparison between electronic and DNA computers, and Mark Daley for Fig. 2.

References

[1] L.M. Adleman, Molecular computation of solutions to combinatorial problems, Science 266 (1994) 1021–1024.
[2] L.M. Adleman, On constructing a molecular computer, in: R.J. Lipton, E.M. Baum (Eds.), DNA Based Computers I, Proceedings of a DIMACS Workshop, Princeton, 1995, American Mathematical Society, Providence, RI, 1996, pp. 1–22.
[3] M. Amos, A. Gibbons, D. Hodgson, Error-resistant implementation of DNA computation, in: L.F. Landweber, E.B. Baum, (Eds.), DNA Based Computers II, Proceedings of a DIMACS Workshop, Princeton, 1996, American Mathematical Society, Providence, RI, 1998, pp. 87–101.
[4] C.R. Calladine, H.R. Drew, Understanding DNA: The Molecule and How it Works, Academic Press, New York, 1999.
[5] E. Csuhaj-Varju, R. Freund, L. Kari, G. Paun, DNA computing based on splicing: universality results, in: L. Hunter, T. Klein (Eds.), Proceedings of the First Pacific Symposium on Biocomputing, World Scientific, Singapore, 1996, pp. 179–190.
[6] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, Bull. Math. Biol. 49 (1987) 737–759.
[7] T. Head, Splicing schemes and DNA, in: G. Rozenberg, A. Salomaa (Eds.), Lindenmayer Systems, Springer, Berlin, 1991, pp. 371–383.
[8] T. Head, G. Paun, D. Pixton, Language theory and molecular genetics, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 2, Springer, Berlin, 1997, pp. 295–358.
[9] J.W. Jaraczewski, J. Frankel, Elimination of Tec elements involves a novel excision process, Genes Dev. 7 (1970) 95–105.
[10] L. Kari, DNA computing — the arrival of biological mathematics, Math. Intelligencer 19 (2) (1997) 9–22.
[11] J. Kari, L. Kari, Context-free recombinations, in: C. Martin-Vide, V. Mitranu (Eds.), Words, Sequences, Languages: where Computer Science, Biology and Linguistics Meet, Kluwer Academic publishers, The Netherlands, 2000, in press.
[12] L. Kari, J. Kari, L.F. Landweber, Reversible molecular computation in ciliates, in: J. Karhumaki, H. Maurer, G. Paun, G. Rozenberg (Eds.), Jewels are Forever, Springer, Berlin, 1999, pp. 353–363.
[13] L. Kari, R. Kitto, G. Gloor, A computer scientist's guide to molecular biology, in: G. Paun, T. Yokomori (Eds.), Soft Computing, Springer, Berlin, in press.
[14] L.F. Landweber, L. Kari, The evolution of cellular computing: nature's solution to a computational problem, in: L. Kari, H. Rubin, D.H. Wood (Eds.), Biosystems, Vol. 52, Nos. 1–3, 1999, Elsevier, Amsterdam, 1999, pp. 3–13.
[15] L.F. Landweber, L. Kari, Universal molecular computation in ciliates, in: L.F. Landweber, E. Winfree (Eds.), Evolution as Computation, Springer, Berlin, 1999.

[16] J.L. Mitcham, A.J. Lynn, D.M. Prescott, Analysis of a scrambled gene: the gene encoding α-telomere-binding protein in *Oxytricha nova*, Genes Dev. 6 (1992) 788–800.
[17] G. Păun, On the power of the splicing operation, Int. J. Comput. Math. 59 (1995) 27–35.
[18] D. Pixton, Linear and circular splicing systems, in: Proceedings of the First International Symposium on Intelligence in Neural and Biological Systems, IEEE Computer Society Press, Los Alamos, 1995, pp. 181–188.
[19] D. Pixton, Regularity of splicing, languages, Discrete Appl. Math. 69 (1-2) (1996) 99–122.
[20] D. Pixton, Splicing in abstract families of languages, in preparation.
[21] D.M. Prescott, The DNA of ciliated protozoa, Microbiol. Rev. 58 (2) (1994) 233–267.
[22] A. Salomaa, Formal Languages, Academic Press, New York, 1973.
[23] R. Siromoney, K.G. Subramanian, V. Rajkumar Dare, Circular DNA and splicing systems, in: Parallel Image Analysis, Lecture Notes in Computer Science, Vol. 654, Springer, Berlin, 1992, pp. 260–273.
[24] T. Yokomori, S. Kobayashi, C. Ferretti, Circular splicing systems and DNA computability, in: Proceedings of the IEEE International Conference on Evolutionary Computation '97, 1997, pp. 219–224.

Lila Kari received her MSc degree in Mathematics/Computer Science at the University of Bucharest, Romania. Her PhD thesis on insertions and deletions in formal languages, supervised by Arto Salomaa, was awarded the Nevanlinna Prize 1991 for the best PhD thesis in Mathematics in Finland. After 2 years at the University of Turku, Finland, she went to the University of Western Ontario, London, Canada, where she has been since. Complementing her work on DNA computing, her research interests are in the theory of computation and discrete mathematics with applications to computer science.