Computer Science Journal of Moldova, vol.**18**, no.2(53), 2010

On computational properties of gene assembly in ciliates

Vladimir Rogojin

Abstract

Gene assembly in stichotrichous ciliates happening during sexual reproduction is one of the most involved DNA manipulation processes occurring in biology. This biological process is of high interest from the computational and mathematical points of view due to its close analogy with such concepts and notions in theoretical computer science as permutation and linked list sorting and string rewriting. Studies on computational properties of gene assembly in ciliates represent a good example of interdisciplinary research contributing to both computer science and biology. We review here a number of general results related both to the development of different computational methods enhancing our understanding on the nature of gene assembly, as well as to the development of new biologically motivated computational and mathematical models and paradigms. Those paradigms contribute in particular to combinatorics, formal languages and computability theories.

1 Introduction

We survey here a number of major results which address some computational properties of evolved DNA manipulation process happening in mating cells of stichotrichous ciliates [57, 27, 37, 59, 66]. Stichotrichous ciliates belong to Domain Eukaryote, Phylum Ciliophora, Subphylum Intramacronucleata, Class Spirotrichea, Subclass Stichotrichia [46].

DNA manipulation during gene assembly in stichotrichous ciliates represents a beautiful example of a computational process happening

in living organisms. Ciliates belong to one of the oldest and most diverse groups of eukaryotic cells [65]. Currently there are known around 8,000 species of ciliates [17]. Two unique features differ ciliates from other eukaryotes: nuclear dualism and possession of hairlike structures on their cellular surfaces which are called *cilia* [17, 58]. Germline and somatic nuclear functions are split between nuclei of two different types, called micro- and macronuclei respectively. Micronuclei keep their genetic data in highly encrypted form: genes are split into fragments separated by non-protein-encoding sequences, those fragments may be shuffled and some of them could be inverted [8]. In the same time macronuclear genome is organized in a very compact form: basically any macronuclear DNA represents one (rarely two) contiguous nucleotide sequences representing genes. Majority of non-stichotrichous ciliates have simpler organization of their micronuclear genome: gene fragments are still separated by non-coding sequences but follow in the orthodox order. When micronuclei get transformed into macronuclei, all non-coding blocks of DNA are being excised and gene fragments get spliced together to form contiguous sequences representing genes [57]. In stichotrichs and several other species of ciliates this process is even more involved due to the necessity to unscramble gene fragments. This process of gene assembly is especially of interest in stichotrichs from the computational point of view, since it could be interpreted formally as a string rewriting and permutation sorting procedure [61, 17].

We present briefly in this article three aspects related to studies of computational properties of gene assembly in ciliates.

First, we consider a restricted versions of the intramolecular operations of gene assembly (called simple and elementary models) which take into account only local intramolecular manipulations with DNA [22, 33, 45]. We describe a number of combinatorial properties of simple and elementary models, including the structure of gene patterns that can be assembled in these restricted models and form of their assembly strategies [32, 44, 33, 56, 63].

Secondly, we address several novel computational models based on gene assembly relying on either contextual molecular recombinations [36] or on non-deterministic sequence matching [4]. We show

that some variants of the models are Turing complete, while some others may be used to solve efficiently (albeit only theoretically) computationally intractable (in particular, NP-complete) problems. Moreover, we mention the result where it was shown that the concept of distributed computations from Tissue-like P systems substitutes well the contextual ingredient of molecular operations in the computational model when demonstrating the Turing universality of gene assembly in ciliates [2].

The third aspect, which we investigate in this paper, is related to an algorithmic approach for studying a graph-theoretical notion of parallel complexity of the gene assembly in ciliates [30].

2 Biological basics of gene assembly in ciliates

As it was mentioned above, ciliates posses two unique features: all ciliates have cilia and all of them are in possession of nuclei of two different types. Cilia represent a complex of moving hair-like organelles projecting from the cellular surface. The motion of cilia is synchronized so that they propel efficiently the cell through the aqueous environment and/or direct nutritious particles (like bacteria, algae or other ciliates) into the cell's oral apparatus [17, 58].

Those nuclei that perform germline function in ciliates are called micronuclei. Micronuclei practically do not participate in RNA transcription and most of the time are passive throughout the life cycle of a cell. However, when ciliates breed, micronuclei get activated. When breeding, two ciliate organisms of the same specie exchange their micronuclear genetical information. On the other hand, almost all RNA transcription in ciliates is carried on in macronuclei [17, 59].

There is a big difference in the internal organization of micronuclear and macronuclear genome. Micronuclear DNA are organized on chromosomes. Each micronuclear DNA represents very long molecule, which contains many genes separated by long non-protein-coding spacer nucleotide sequences. Each gene is broken into some number of fragments separated from each other by non-protein-coding blocks [8]. In stichotrichs ciliates and several other species gene fragments are

also shuffled throughout the molecule and some of the fragments are inverted [59, 17]. Contrary to micronuclear DNA macronuclear molecules are short and contain usually one, rarely two contiguous nucleotide protein-coding sequences [8].

The internal molecular structure of micronuclear genes suits well for robust preservation of the genetic information for future generations, while macronuclear gene structure is optimized for rapid RNA transcription, what should bring an evolutionary advantage for ciliate organisms [17].

Macronuclei from maternal organisms get disintegrated during sexual reproduction, while some copies of micronuclei from child organisms are being transformed into new macronuclei. During this transformation heavy editing of micronuclear DNA occurs, so that non-protein-coding sequences (called *Internal Eliminated Sequences*, IESs) get eliminated and new macronuclear DNA are being created from the micronuclear DNA as the result of splicing of micronuclear gene fragments (called *Macronuclear Destined Sequences*, MDSs). Thus, this DNA manipulation process is called *gene assembly* [57]. This process is particularly complex in stichotrichous ciliates because gene fragments on the micronuclear DNA are shuffled and some of the fragments are inverted [8].

The order in which MDSs should be spliced to each other to assemble a macronuclear gene is indicated by short nucleotide sequences (called *pointers*) placed on the edges of MDSs. Any two MDSs which stay next to each other in the assembled macronuclear gene share same pointer on their respective edges. In this way, pointers "tell" for each MDS to which other MDSs it should be spliced to. One can think that MDSs that follow directly one another in the macronuclear DNA "point" to each other by means of their pointers. Thus, a micronuclear gene pattern could be interpreted as a linked list data structure, and the gene assembly process could be seen as a list sorting procedure [61].

3 Molecular models for gene assembly

Two molecular models for gene assembly that suggest splicing of MDSs on their common pointers are called intermolecular [39, 42, 43] and intramolecular [20, 21]. As it follows from their names, the intermolecular model considers interaction and exchange of some pieces of DNA between several molecules during gene assembly, while the intramolecular model assumes that all manipulations are being carried on in all DNA independently from each other.

The intermolecular model

The intermolecular model considers three molecular operations [39, 42, 43]:

1. *Intramolecular recombination:* A block of DNA flanked by occurrences of same pointer gets excised in the form of a circular molecule. As the result, two molecules are produced, one is linear which contains the remaining nucleotide sequence, and another is the circular one containing the excised sequence. For details we refer to Figure 1.

2. *Intermolecular recombination:* As the inverse of the intramolecular recombination, a circular molecule gets inserted into a linear molecule if both molecules posses occurrences of the same pointer. The circular molecule is cut at the site of the occurrence of the pointer and gets inserted at the location of another occurrence of the pointer into the linear molecule. See Figure 2.

3. *Intermolecular recombination:* Two linear molecules recombine on their common pointer. As the result, two molecules interchange their tails starting on the common pointer. Note, that this operation is self-reversible, i.e., if applied on its resulting molecules on the same pointer, the initial two molecules can be obtained. See Figure 3.

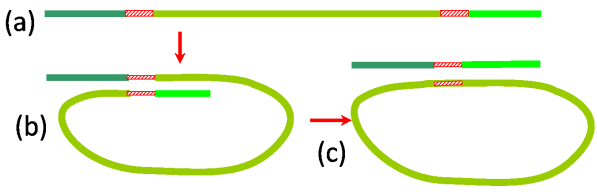

Figure 1. *Intramolecular recombination.* (a) Initial molecule: a pointer is in direct repeat. (b) Folding: the molecule folds forming a loop so that occurrences of the pointer get next to each other. (c) The result: A part of the molecule flanked by the occurrences of the pointer gets excised in the form of a circular molecule.

The intramolecular model

The intramolecular model considers three operations, called *ld*, *hi* and *dlad* [20, 21]:

1. *Loop, direct-repeat excision, ld:* This operation is applicable to a molecule having either an IES flanked by repeating occurrences of a pointer (called *simple ld*) or having a block containing all MDSs flanked by occurrences of a pointer, and this pointer occurs twice with the same orientation. The molecule folds forming a loop, so that occurrences of the pointer get next to each other. Then the recombination happens, and as the result the block flanked by occurrences of the pointer gets excised as a circular molecule, see Figure 4. Note, that this operation resembles the intramolecular recombination from the intermolecular model, except that its simple version does not excise DNA blocks containing any MDS. During gene assembly this operation is used to get rid of non-coding blocks.

2. *Hairpin, inverted-repeat excision/reinsertion, hi:* This operation is applicable to a molecule having two occurrences of the same pointer with opposite orientations. The molecule folds forming a hairpin loop, so that both occurrences of the pointer are brought next to each other and have the same spacial orienta-

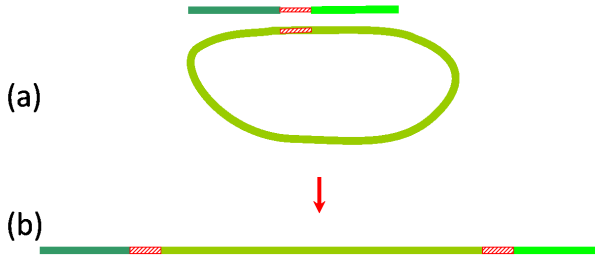

Figure 2. *Intermolecular recombination. The inverse of the intramolecular recombination* (a) Initial molecules: a circular and a linear molecule having occurrences of the same pointer (b) Result: the circular molecule gets inserted into the linear molecule at the site of occurrences of the pointer.

Figure 3. *Intermolecular recombination.* (a) Initially two linear molecules with occurrences of the same pointer. (b) Result: molecules exchange their tails starting at occurrences of the pointer.

tion. Then the recombination is possible, and as the result, piece of the molecule flanked by occurrences of the pointer in the inverted repeat is inverted, see Figure 5. This operation is used throughout the assembly process in order to restore the proper orientation of MDSs.

3. *Double loop, alternating direct-repeat excision-reinsertion, dlad:* This operation can be applied when the molecule has two pointers occurring in an overlapping direct repeat. I.e., when block flanked by occurrences of the same orientation of a pointer overlaps with the block flanked by occurrences of the same orientation of the other pointer. The molecule folds forming a double loop so that occurrences of both pointers get next to each other so that the recombination be possible. In the result of this recombination, non-overlapping parts of blocks flanked by their pointers

exchange their places, see Figure 6. This operation is used in the assembly in order to sort MDSs in proper order.

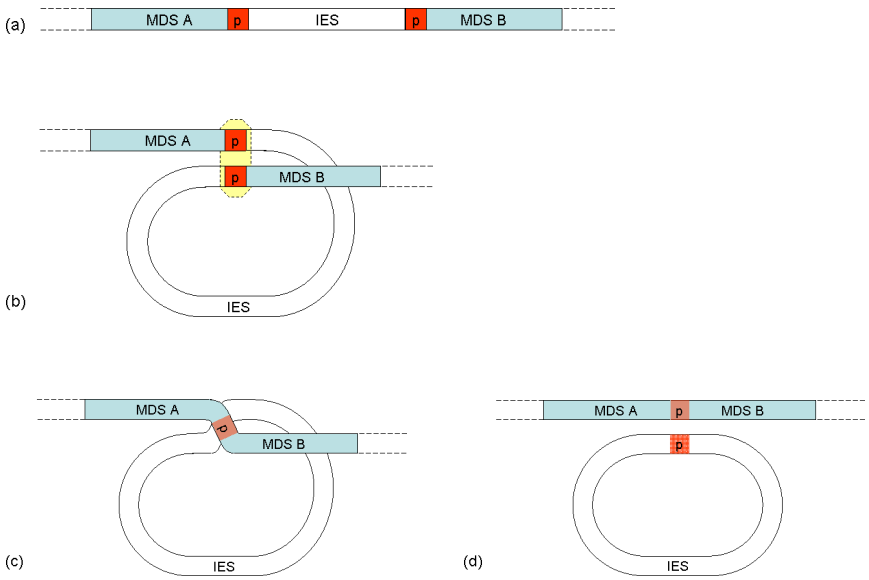

Figure 4. *Simple loop recombination [64].* (a) Initial molecule: an IES flanked by occurrences of pointer p. (b) Loop-folding, alignment of occurrences of pointer p. (c) Recombination by pointer p. (d) Result: the IES is excised in the form of circular molecule, MDS A and MDS B are spliced on the common pointer p in the linear molecule. One occurrence of p is present in the linear molecule and one occurrence of p is present in the circular molecule, but neither of them acts as a pointer.

Note, that contrary to the intermolecular model, the intramolecular operations are not reversible. Application of any of *ld*, *hi* or *dlad* reduces the number of MDSs by gluing two or more MDSs on their common pointers into bigger composite MDSs. A pointer is considered to stop acting as a pointer, when its occurrence gets either inside of an composed IES or of an composed MDS.

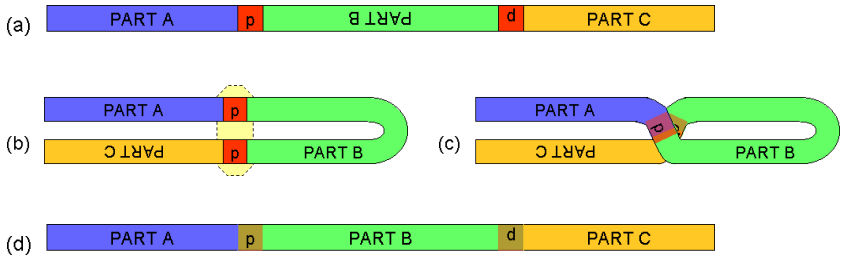

Figure 5. *Hairpin recombination [64].* (a) Initial molecule: one occurrence of p in an orthodox orientation and another in the inverted orientation. (b) Hairpin-folding, alignment of occurrences of pointer p. (c) Recombination by pointer p. (d) Resulting molecule: the orientation of *PART B* is changed, the rest of the molecule is not affected. As a result of the inversion of *PART B* MDSs having pointer p are spliced together. Occurrences of p and their orientations are retained, but neither of the occurrences acts as a pointer.

Simple and elementary intramolecular models

The intramolecular operations allow in their general formulation that DNA blocks participating in the recombination may contain any number of MDSs. However, arguing on the principle of parsimony, *simple intramolecular operations* were introduced in [22] suggesting that all the recombinations are applied "locally". Simple operations follow the same folding and recombination events as the operations in the general intramolecular model, however, blocks of DNA that are being inverted or relocated may contain exactly one MDS. Even further simplification of the intramolecular operations led to so called *elementary operations*, where only the micronuclear (non-composite) MDSs are allowed to be inverted/relocated [33]. I.e., as soon as two MDSs are combined into a composite MDS, this MDS cannot be rearranged by the elementary operations.

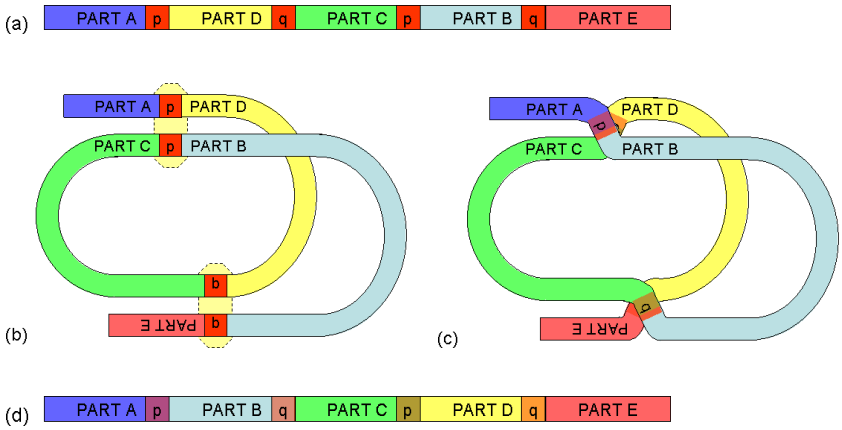

Figure 6. *Double-loop recombination [64].* (a) Initial molecule: blocks flanked by occurrences of pointers p and q respectively overlap, their common nucleotide sequence is *PART C*. (b) The molecule forms a double-loop folding so that occurrences of p and q get aligned for the recombination. (c) Recombination by pointers p and q. (d) Resulting molecule: *PART B* and *PART D* exchange places. As a result of the translocation of *PART B* and *PART D* MDSs having pointer p are spliced together, and MDSs having pointer q are spliced together. Occurrences of p and q remain in the molecule, but none of them act as a pointer.

Template-guided recombination models

Since nucleotide sequences representing pointers are very short (from 2 to 20 base-pairs) and may occur also in the middle of MDSs and IESs [6], there is a need for some kind of guiding mechanism which "helps" ciliates to identify correctly MDSs and to align "real" occurrences of the pointers next to each other and splice "real" MDSs in the right order. The template-guided recombination models suggest such mechanism [7, 60]. These models rely on the concept of templates - macronuclear DNA or RNA remained from maternal organisms. The presence of those molecules in the newly formed macronuclei and their

critical role for gene assembly have been shown experimentally [49]. The template-guided recombination represents *triple-splicing*, where two recombining molecules (or different parts of the same molecule) get aligned next to each other and recombine with the help of the template molecule which being as a product of an earlier equivalent recombination, shows the way the alignment and the recombination should be done [7, 60].

In the first template-guided recombination model proposed in [60] a double-stranded DNA serves as a template. This DNA is assumed to be a copy of the assembled gene from the parental macronucleus. In this model, the template DNA is placed in-between recombining DNA. As the result of the recombination, one double stranded DNA whose nucleotide sequence contains the left part of one of the recombining molecules and the right part of the other of the recombining molecules and which matches the template is obtained. During this recombination, the recombining molecules and the template molecule exchange some of their physical parts, thus the resulting DNA contains some parts of the template, and the template is reconstructed to its original form, so that it could be reused for other recombinations. Also, the right part of the first of the recombining molecules and the left part of the second of the recombining molecules that do not much the template get excised. In this template-guided recombination model the parts of the recombining molecules that do not match the template are not spliced together after the recombination, what contradicts both the intermolecular and the intramolecular models.

A modification of the template-guided model from above was suggested in [7]. Instead of double-stranded DNA either a single- or double-stranded RNA serves as the template. During the recombination, the spacial position of the double-stranded template is "above" the recombining molecules contrary to the previous model. No physical parts of the template get incorporated inside the resulting molecules. Parts of the recombining molecules that do not match the template are also spliced together.

4 Formalizing gene assembly

As we have mentioned above, gene assembly process can be interpreted from the computational point of view as a permutation sorting or string (or multiset of strings) rewriting process. We concentrate in this manuscript on the intramolecular model. Hereby, we present briefly here several formalisms for the intramolecular model at different levels of abstraction. We start from the permutation-based formalism [33], then we continue with MDS-descriptors [20, 21], after that we switch to double occurrence strings [18, 23] and contextual string rewriting rules [41, 47, 51], and finally we present graph-based formalization [18, 23] for the intramolecular model.

Gene assembly as sorting of permutations

The most suitable formalism to handle simple and elementary intramolecular operations is through rewriting rules for signed permutations [33, 45]. A given gene pattern could be represented by a signed permutation which indicates the order and orientation of its micronuclear MDSs. The gene assembly process itself could be interpreted as a sorting of the signed permutation.

We formalize here only the elementary intramolecular model. As the elementary operations rearrange only micronuclear MDSs, this leads to the following formalization of elementary operations:

Definition 1 *1. For each $p \geq 1$, eh_p is defined as follows:*

$$\begin{aligned}
\mathsf{eh}_p(xp\overline{(p+1)}z) &= xp(p+1)z,\\
\mathsf{eh}_p(x\overline{p}(p+1)z) &= xp(p+1)z,\\
\mathsf{eh}_p(x(p+1)\overline{p}z) &= x\overline{(p+1)}\overline{p}z,\\
\mathsf{eh}_p(x\overline{(p+1)}pz) &= x\overline{(p+1)}\overline{p}z,
\end{aligned}$$

where x, z are signed strings over Π_n. We denote $\mathsf{Eh} = \{\mathsf{eh}_p \mid 1 \leq p \leq n\}$.

2. *For each* p, $2 \leq p \leq n-1$, ed_p *is defined as follows:*

$$\mathsf{ed}_p(xpy(p-1)(p+1)z) = xy(p-1)p(p+1)z,$$
$$\mathsf{ed}_p(x(p-1)(p+1)ypz) = x(p-1)p(p+1)yz,$$
$$\mathsf{ed}_p(x\overline{p}y\overline{(p+1)}\,\overline{(p-1)}z) = xy\overline{(p+1)}\,\overline{p}\overline{(p-1)}z,$$
$$\mathsf{ed}_p(x\overline{(p+1)}\,\overline{(p-1)}y\overline{p}z) = x\overline{(p+1)}\,\overline{p}\,\overline{(p-1)}yz,$$

where x, y, z *are signed strings over* Π_n. *We denote* $\mathsf{Ed} = \{\mathsf{ed}_p \mid 1 < p < n\}$.

Since the permutation-based formalism captures only the MDS inversion and relocation events throughout the gene assembly, *LD* operation is not being formalized because it neither inverts, nor relocates MDSs [64].

Example 1 *[33]*

(i) Permutation $\pi_1 = \overline{4}\,\overline{5}\,6\,\overline{1}\,2\,\overline{3}$ *is sortable and a sorting composition is* $(\mathsf{eh}_4 \circ \mathsf{eh}_5 \circ \mathsf{eh}_2 \circ \mathsf{eh}_1)(\pi_1) = 4\,5\,6\,1\,2\,3$. *Permutation* $\pi_1' = \overline{4}\,\overline{5}\,6\,\overline{1}\,\overline{2}\,\overline{3}$ *is unsortable. Indeed, only* $\mathsf{eh}_4 \circ \mathsf{eh}_5$ *is applicable to* π_1', *but it does not sort it.*

(ii) There exist permutations with several sorting compositions, even leading to different (cyclically) sorted permutations. One such permutation is $\pi_2 = 2\,4\,1\,3$. *Indeed,* $\mathsf{ed}_2(\pi_2) = 4\,1\,2\,3$. *At the same time,* $\mathsf{ed}_3(\pi_2) = 2\,3\,4\,1$.

(iii) There are permutations having both sorting compositions and non-sorting compositions leading to unsortable permutations. If $\pi_3 = 2\,4\,1\,3\,5$, *then* $\mathsf{ed}_3(\pi_3) = 2\,3\,4\,1\,5$ *is a unsortable permutation. However,* π_3 *can be sorted, e.g., by the following composition:* $(\mathsf{ed}_4 \circ \mathsf{ed}_2)(\pi_3) = 1\,2\,3\,4\,5$.

(iv) Applying a cyclic shift to a permutation may render it unsortable. Indeed, permutation $2\,1\,3$ *is sortable, while* $3\,2\,1$ *is not.*

Gene assembly as MDS descriptor rewriting process

Here we represent a gene pattern through its sequence of MDSs. Each MDS we represent through its incoming and outgoing pointers (opening and closing pointers respectively), as well as through the sequence of pointers incorporated in the MDS on which micronuclear MDSs spliced to form this composite MDS [20, 21, 4]. Formally, let $\Sigma_P = \{p_1, p_2, \ldots, p_n\}$ be the set of pointers, and $b, e \in \Sigma_P$ be so called *begin* and *end* markers (denoting opening sequence of very first MDS and closing sequence of very last MDS respectively). Then we represent an MDS by a triple $M = (p, u, q)$, where $p \in \Sigma_P \cup \{b\}$, $q \in \Sigma_P \cup \{e\}$ are called *active pointers* and $u \in \Sigma_P^*$ is called the *content*. It is said, that p is an *incoming* and q is an *outgoing* pointer of M. The length of M is denoted as $|M| = |puq|$. We denote the set of all MDSs over Σ_P as $\Sigma_M = \{(p, u, q) | p \in \Sigma_P \cup \{b\}, q \in \Sigma_P \cup \{e\}, u \in \Sigma_P^*\}$. The inversion of MDS $M = (p, u, q)$ we denote as $\overline{M} = (\overline{q}, \overline{u}, \overline{p})$ and set of inverted MDSs as $\overline{\Sigma_M} = \{\overline{M} | M \in \Sigma_M\}$.

The gene pattern we represent by its MDS descriptor – a sequence of MDSs and their orientations. Formally, an MDS descriptor is a string over alphabet $\Sigma_M \cup \overline{\Sigma_M}$.

Note, that the assembled gene is represented by any of the composite MDSs $(b, p_1 p_2 \ldots p_n, e)$ and $(\overline{e}, \overline{p_n} \ldots \overline{p_2 p_1}, \overline{b})$. In this way, a gene assembly process may be interpreted as an MDS descriptor rewriting process that leads to any of the two MDSs from above.

The intramolecular operations we formalize on MDS descriptors as follows:

1. *ld* operation on pointer p is formalized as ld_p:

$$\psi_1(q, u, p)\psi_2(p, v, r)\psi_3 \Rightarrow^{\mathsf{ld}_p} \psi_1(q, upv, r)\psi_3$$

2. *hi* operation on pointer p is formalized as hi_p:

   - $\psi_1(p, u, q)\psi_2(\overline{p}, \overline{v}, \overline{r})\psi_3 \rightarrow^{\mathsf{hi}_p} \psi_1\overline{\psi_2}(\overline{q}, \overline{u}\ \overline{p}\ \overline{v}, \overline{r})\psi_3$;
   - $\psi_1(q, u, p)\psi_2(\overline{r}, \overline{v}, \overline{p})\psi_3 \rightarrow^{\mathsf{hi}_p} \psi_1(q, upv, r)\overline{\psi_2}\psi_3$;

3. *dlad* operation on pointers p and q is formalized as $dlad_{p,q}$:

- $\psi_1(p,u_1,r_1)\psi_2(q,u_2,r_2)\psi_3(r_3,u_3,p)\psi_4(r_4,u_4,q)\psi_5 \Rightarrow^{\mathsf{dlad}_{p,q}} \psi_1\psi_4(r_4,u_4qu_2,r_2)\psi_3(r_3,u_3pu_1,r_1)\psi_2\psi_5$;
- $\psi_1(p,u_1,r_1)\psi_2(r_2,u_2,q)\psi_3(r_3,u_3,p)\psi_4(q,u_4,r_4)\psi_5 \Rightarrow^{\mathsf{dlad}_{p,q}} \psi_1\psi_4\psi_3(r_3,u_3pu_1,r_1)\psi_2(r_2,u_2qu_4,r_4)\psi_5$;
- $\psi_1(r_1,u_1,p)\psi_2(q,u_2,r_2)\psi_3(p,u_3,r_3)\psi_4(r_4,u_4,q)\psi_5 \Rightarrow^{\mathsf{dlad}_{p,q}} \psi_1(r_1,u_1pu_3,r_3)\psi_4(r_4,u_4qu_2,r_2)\psi_3\psi_2\psi_5$;
- $\psi_1(r_1,u_1,p)\psi_2(r_2,u_2,q)\psi_3(p,u_3,r_3)\psi_4(q,u_4,r_4)\psi_5 \Rightarrow^{\mathsf{dlad}_{p,q}} \psi_1(r_1,u_1pu_3,r_3)\psi_4\psi_3\psi_2(r_2,u_2qu_4,r_4)\psi_5$;
- $\psi_1(p,u_1,r_1)\psi_2(q,u_2,p)\psi_4(r_4,u_4,q)\psi_5 \Rightarrow^{\mathsf{dlad}_{p,q}} \psi_1\psi_4(r_4,u_4qu_2pu_1,r_1)\psi_2\psi_5$;
- $\psi_1(p,u_1,q)\psi_3(r_3,u_3,p)\psi_4(q,u_4,r_4)\psi_5 \Rightarrow^{\mathsf{dlad}_{p,q}} \psi_1\psi_4\psi_3(r_3,u_3pu_1qu_4,r_4)\psi_5$;
- $\psi_1(r_1,u_1,p)\psi_2(q,u_2,r_2)\psi_3(p,u_3,q)\psi_5 \Rightarrow^{\mathsf{dlad}_{p,q}} \psi_1(r_1,u_1pu_3qu_2,r_2)\psi_3\psi_2\psi_5$;

For more details on this formalism we refer to [20, 21, 4].

Example 2 *[64]*

Let us consider a micronuclear gene pattern of the actin I gene from Stylonychia lemnae*. Its MDS descriptor is*

$$\delta = (3,4)(4,5)(5,6)(7,8)(\overline{3},\overline{2})(b,2)(6,7)(8,e).$$

Composition $\Phi = \mathsf{ld}_5 \circ \mathsf{ld}_4 \circ \mathsf{hi}_{\overline{7}} \circ \mathsf{hi}_{\overline{2}} \circ \mathsf{hi}_3 \circ \mathsf{dlad}_{6,8}$ *reduces* δ *to MDS* $(b,234567,e)$*. Indeed,*

$$\begin{aligned}
\delta' = \mathsf{dlad}_{6,8}(\delta) &= (3,4)(4,5)(5,6,7)(\overline{3},\overline{2})(b,2)(7,8,e)\\
\delta'' = \mathsf{hi}_3(\delta') &= (\overline{7},\overline{6},\overline{5})(\overline{5},\overline{4})(\overline{4},\overline{3},\overline{2})(b,2)(7,8,e)\\
\delta''' = \mathsf{hi}_{\overline{2}}(\delta'') &= (\overline{7},\overline{6},\overline{5})(\overline{5},\overline{4})(\overline{4},\overline{32},\overline{b})(7,e)\\
\delta^{iv} = \mathsf{hi}_{\overline{7}}(\delta''') &= (b,23,4)(4,5)(5,67,e)\\
\delta^{v} = \mathsf{ld}_4(\delta^{iv}) &= (b,234,5)(5,67,e)\\
\delta^{vi} = \mathsf{ld}_5(\delta^{v}) &= (b,234567,e)
\end{aligned}$$

Gene assembly by contextual intramolecular operations on double occurrence strings

If we abstract from the information about MDSs and concentrate only on pointers occurrences in a gene pattern, then we obtain a string based formalism, where each letter and its sign represent an occurrence of a pointer and its orientation [23]. If we follow the idea that pointer alignment during gene assembly is context-guided, then we may come to the similar concept of contextual intramolecular operations [36].

Formally, set of templates may be represented by so-called *splicing scheme* which is a set of *splicing relations*. A splicing rule is represented by a pair of triplets $(\alpha, p, \beta) \sim (\alpha', p, \beta')$ which means that the recombination at pointer p is possible if and only if, one of its occurrences is flanked by sequences α and β and another of its occurrences is flanked by α' and β'.

The contextual *ld* on pointer p is formalized as del_p: $del_p(xpupy) = xpy$ with respect to splicing scheme R if and only if there is a splicing relation $(\alpha, p, \beta) \sim (\alpha', p, \beta')$ in R such that $x = x'\alpha$, $u = \beta u' = u''\alpha'$ and $y = \beta' y'$.

The contextual *dlad* on pointers p and q is formalized as $trl_{p,q}$: $trl_{p,q}(xpuqypvqz) = xpvqypuqz$ with respect to splicing scheme R if and only if there are splicing relations $(\alpha, p, \beta) \sim (\alpha', p, \beta')$ and $(\gamma, q, \delta) \sim (\gamma', q, \delta')$ in R such that $x = x'\alpha$, $uqy = \beta u' = u''\alpha'$, $vqz = \beta' v'$, $xpu = x''\gamma$, $ypv = \beta y' = y''\gamma'$ and $z = \delta' z'$.

Formalizing contextual *hi* is beyond our scope in this manuscript.

For non-contextual formalizations of *ld*, *hi* and *dlad* we refer to [18, 23].

Gene assembly as graph reduction process

If we abstract from positions of pointers in a gene pattern and concentrate only on their overlapping relations, then we obtain graph-based formalism for gene assembly [18, 23]. We recall that two pointers p and q overlap if and only if the interval flanked by pointer p overlaps (but does not include and is not included) with interval delimited by

pointer q. I.e., we say that p and q overlap if in the gene pattern we have either scattered subsequence $pqpq$ or $qpqp$.

Formally, let u be a string representing occurrences of pointers in a gene pattern. We define for it signed overlap graph $G = (V, E, \sigma)$ as follows:

- $V = dom(u)$, i.e., each node p from G corresponds to a pointer p from u;
- $E = \{\{p, q\} | pqpq \leq_s u \text{ or } qpqp \leq_s u\}$, i.e., $\{p, q\}$ is an undirected edge in G if and only if pointers p and q overlap in u;
- $\sigma : V \to \{+, -\}$, where $\sigma(p) = -$ if and only if both occurrences of p have the same sign (i.e., the same orientation) in u.

Operations LD, HI and $DLAD$ are represented on the abstraction level of signed overlap graphs as follows:

Let $G = (V, E, \sigma)$ be a signed overlap graph:

- Operation ld on pointer p is formalized as graph reduction operation gnr: the operation gnr is applicable to a vertex $p \in V$ if $\sigma(p) = -$ and $N(p) = \emptyset$. In this case, $\mathsf{gnr}_p(G) = G - \{p\}$.
- Operation hi on pointer p is formalized as graph reduction operation gpr: the operation gpr is applicable to vertex $p \in V$ if $\sigma(p) = +$. In this case, $\mathsf{gpr}_p(G) = loc_p(G) - \{p\}$, where $loc_p(G) = (V, E, \sigma')$ with $\sigma'(x) = -\sigma(x)$ if x is a neighbor of p, otherwise $\sigma'(x) = \sigma(x)$, for all $x \in V$. Here $-\sigma(x) = -$ if and only if $\sigma(x) = +$.
- Operation $dlad$ on pointers p and q is formalized as graph reduction operation gdr: The operation $\mathsf{gdr}_{p,q}$ is applicable to adjacent vertices $p, q \in V$ if $\sigma(p) = \sigma(q) = -$. In this case, $\mathsf{gdr}_{p,q}(G) = G'$ where $G' = (V \setminus \{p, q\}, E', \sigma)$. Here for all pairs of x and y from $V \setminus \{p, q\}$ such that $x \in N_G(p) \setminus N_G(q)$ and $y \in N_G(q) \setminus N_G(p)$ we have edge $\{x, y\} \in E'$ if and only if x and y are not neighbors in G. For all other pairs x, y from $V \setminus \{p, q\}$ we have $\{x, y\} \in E'$ if and only if $\{x, y\} \in E$.

Example 3 *[64]*

Here we show how an overlap graph G which represents the micronuclear gene pattern of the actin I gene from Stylonychia lemnae*, can be reduced to the empty graph:*

Step 1: *Vertices* 6 *and* 8 *are negative and adjacent in G. In this way, we can apply* $\mathsf{gdr}_{6,8}$. *The resulting graph* $\mathsf{gdr}_{6,8}(G)$ *is represented in Figure 7(a);*

Step 2: *Vertex* 3 *is positive in* $\mathsf{gdr}_{6,8}(G)$. *Then, we can apply* gpr_3. *Vertex* 7 *changes its sign since it is adjacent to* 3. *Then we obtain graph represented in Figure 7(b);*

Step 3: *Vertices* 2 *and* 7 *are positive in* $\mathsf{gpr}_3(\mathsf{gdr}_{6,8}(G))$. *Then we can apply* gpr_2 *and* gpr_7. *The resulting graph is represented in Figure 7(c);*

Step 4: *Finally, only two negative isolated vertices* 4 *and* 5 *remain in* $(\mathsf{gpr}_7 \circ \circ \mathsf{gpr}_2 \circ \mathsf{gpr}_3 \circ \mathsf{gdr}_{6,8})(G)$. *By applying* gnr_4 *and* gnr_5 *we reduce the graph to the empty one.*

5 Complexity of gene assembly

Intuitively, by "complexity" of gene assembly one can understand the "effort" needed to assemble a gene. In literature exist several concepts for the complexity of gene assembly related to types of molecular operations involved into the process as well as to the order and the manner in which those operations are applied [64]. In this section we address such gene assembly complexity related topics as simple [22] and elementary [33] gene assembly as well as the parallel complexity of gene assembly [30, 28].

Simple and elementary gene assembly

In its general formulation, intramolecular operations may invert and translocate blocks of DNA containing any number of MDSs. It has

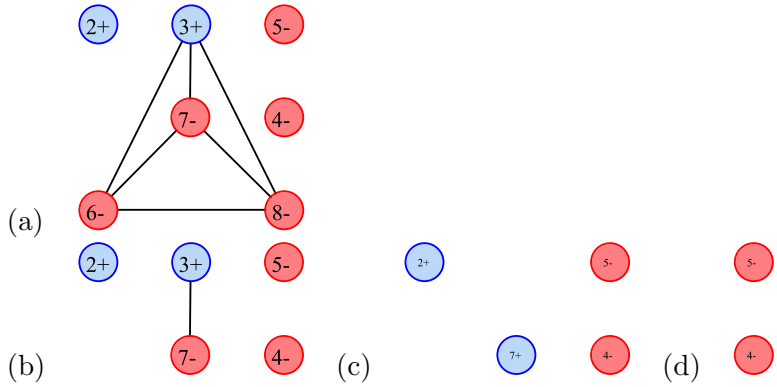

Figure 7. Graph reduction strategy [64]: (a) Graph G which represents the micronuclear gene pattern of the actin I gene from *Stylonychia lemnae*, (b) Graph $\mathsf{gdr}_{6,8}(G)$, (c) graph $\mathsf{gpr}_3(\mathsf{gdr}_{6,8}(G))$, (d) graph $(\mathsf{gpr}_7 \circ \mathsf{gpr}_2 \circ \mathsf{gpr}_3 \circ \mathsf{gdr}_{6,8})(G)$.

been shown that the intramolecular model is complete in a sense that any given hypothetical micronuclear gene pattern can be assembled to the macronuclear gene [17]. The restriction from general intramolecular to simple model has one immediate consequence, simple model is not complete. I.e., there are some hypothetical gene patterns for which there is no assembly strategy consisting of solely simple operations which lead to assembled macronuclear gene. However, simple operations can assemble all currently discovered from ciliates gene patterns [17, 22]. This fact enables the hypothesis that ciliates use simple operations to assemble their genes. Then it might be interesting to characterize those gene patterns that can be assembled by simple operations.

There may exist many different intramolecular assembly strategies applicable to the same micronuclear gene pattern [17]. In general intramolecular model, any assembly strategy applicable to a gene pattern leads to an assembled gene. However, since simple model is not complete, there are gene patterns for which there is no simple assembly

strategy leading to the assembled gene. Interestingly, any gene pattern which can be assembled by simple operations to the macronuclear gene has only successful simple strategies (i.e., those leading to the macronuclear gene) [44]. In this way, characterizing those gene patterns that can be assembled by simple strategies is straightforward: just try a simple assembly strategy and see if it assembles successfully a gene pattern to the macronuclear gene. However, the situation is totaly different for elementary operations.

The slight difference between the definitions of simple and elementary operations (elementary *hi* and *dlad* invert/relocate blocks of DNA containing only one non-composite MDS) generates the following important outcome: a gene pattern may have both successful and unsuccessful applicable elementary assembly strategies [33]. This means that characterization of those gene patterns that can be assembled by elementary operations is not as straightforward as it is in the case of simple operations. It may be necessary to try some number of different elementary assembly strategies applicable to a gene pattern before finding a successful one. However, we have found an efficient combinatorial procedure to decide whether a gene pattern may be assembled by elementary operations without actually trying any of elementary assembly strategies [56, 63].

Our decision method is basing on the concept of a *dependency graph* associated to a gene pattern [33]. The dependency graph is a directed graph reflecting the information about the order in which operations can be used in strategies applicable to the gene pattern as well as about those operations that are never used in any strategy for this pattern. In this manuscript we present results addressing gene patterns without inverted MDSs. For characterization of gene patterns with the inverted MDSs we refer to [33].

As it was mentioned above, signed permutations is a suitable formalism to represent gene patterns when working with elementary operations. The dependency graph associated to permutation π is defined as $\Gamma_\pi = (V_\pi, E_\pi)$, where $V_\pi = dom(\pi)$ and

$$E_\pi = \{(1,1),(n,n)\} \cup \{(i,i) | (i+1)(i-1) \leq_s \pi\} \cup$$

$$\cup\{(j,i)|(i-1)j(i+1) \leq_s \pi\}.$$

Also, we denote the subgraph induced from Γ_π by set $T_{\pi,p} = \{q|\text{there is a path from } q \text{ to } p \text{ in } \Gamma_\pi\}$ as $\Gamma_{\pi,p}$.

Example 4 *[64]*
Let $\pi = 1\,10\,3\,5\,7\,12\,2\,9\,4\,11\,6\,13\,8$. *Then*

$$V_\pi = \mathsf{dom}(\pi) = \{1,2,3,4,5,6,7,8,9,10,11,12,13\}$$

and

$E_\pi =$	$\{$ $(1,1)$	
	$(10,2)$,	*since* $1\,10\,3 \leq_s \pi$
	$(9,3)$,	*since* $2\,9\,4 \leq_s \pi$
	$(11,5)$,	*since* $4\,11\,6 \leq_s \pi$
	$(13,7)$,	*since* $6\,13\,8 \leq_s \pi$
	$(2,8)$, $(12,8)$	*since* $12\,12\,2\,9 \leq_s \pi$
	$(9,9)$,	*since* $10\,8 \leq_s \pi$
	$(4,10)$,	*since* $9\,4\,11 \leq_s \pi$
	$(3,11)$, $(5,11)$, $(7,11)$,	*since* $10\,3\,5\,7\,12 \leq_s \pi$
	$(6,12)$,	*since* $11\,6\,13 \leq_s \pi$
	$(13,13)$, $\}$	*since* $n = 13$

The corresponding dependency graph $\Gamma_\pi = (V_\pi, E_\pi)$ *is shown in Figure 8.*

Since we consider here gene patterns without inverted MDSs, all gene assembly strategies that we address do not use *hi* operations. I.e., formally we consider Ed-sortable permutations. For dependency graph-based characterization of Eh, Ed-sortable permutations we refer to [33].

Here we present a constructive dependency graph-based characterization of Ed-sortable permutations from [56]. We use the notion of so-called "forbidden integers" for a permutation, integers on which ed

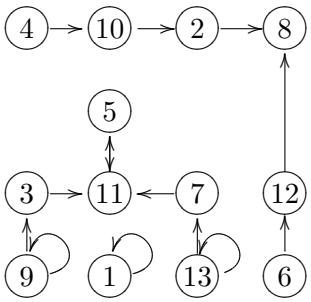

Figure 8. The dependency graph $\Gamma_\pi = (V_\pi, E_\pi)$ of permutation $\pi = 1\,10\,3\,5\,7\,12\,2\,9\,4\,11\,6\,13\,8$ [64]

operations are never used in any ed-strategy applicable to the permutation. I.e., an integer p from $\mathsf{dom}\pi$ for some unsigned permutation π we call forbidden if and only if there is no Ed strategy applicable to π where operation ed_p is used.

We can decide the set of forbidden integers for a permutation π as follows:

Theorem 1 *[56] For a permutation π over Σ_n and $p \in \Sigma_n$, p is forbidden in π if and only if the subgraph $\Gamma_{\pi,p} = (T_{\pi,p}, E_{\pi,p})$ is cyclic or $q-1, q \in T_{\pi,p}$ for some q.*

By $\pi|_U$ we denote a scattered substring of π containing only the letters from set U.

Now, we decide the Ed-sortability of a permutation π as follows:

Theorem 2 *[56]*

Permutation π is sortable if and only if $\pi|_{F(\pi)}$ is sorted.

Example 5 *[64]*

Let π be the permutation from Example 4. By Theorem 1, set $F(\pi) = \{1, 3, 5, 7, 9, 11, 13\}$, since 1, 9 and 13 are in self-loops, 5 and 11 form a cycle, and edges $(9, 3)$ and $(13, 7)$ belong to Γ_π. Clearly $\pi|_{F(\pi)} = 1\,3\,5\,7\,9\,11\,13$, which is sorted. Thus π is sortable. For instance, composition of ed *operations $\mathsf{ed}_8 \circ \mathsf{ed}_{12} \circ \mathsf{ed}_6 \circ \mathsf{ed}_2 \circ \mathsf{ed}_{10} \circ \mathsf{ed}_4(\pi)$ sorts π.*

This theorem does not provide any information about those ed strategies that sort π. The general form of all sorting strategies was presented in [63].

Parallel gene assembly

By parallel complexity of a gene pattern we understand the minimal number of parallel steps needed to assemble a gene. Formally, the parallelism of gene assembly is analyzed by means of the pointer reduction system. We say that a set of graph reduction operations may be applied in parallel to a graph if and only if these operations may be applied in any order to the graph. It was shown that for an initial graph the set of reduction operations, but not the order of their application determine the resulting graph. All these enables the notion of *parallel reduction* of a graph as a sequence of *parallel steps* leading to the empty graph [30, 28]. Here a parallel step is a set of operations applied in parallel. We represent a parallel reduction formally as $\Phi = S_k \circ \ldots \circ S_1$, where S_i is a set of operations applicable in parallel to graph $S_{i-1} \circ \ldots \circ S_1(G)$ and $\Phi(G)$ is an empty graph. The parallel complexity of Φ is k, we denote it as $\mathcal{C}(\Phi) = k$. By parallel complexity of graph G we understand the minimal complexity among all of its parallel reductions. Formally, the parallel complexity of G is $\mathcal{C}(G) = \min\{\mathcal{C}(R) \mid R \text{ is a parallel reduction of } G\}$.

One of the main open questions related to the topic of parallel gene assembly is whether *"the common finite upper bound for parallel complexities of all signed overlap graphs exists"*. This question was answered only for some particular types of graphs in [28, 31, 30]. The highest parallel complexity for negative trees is 2, and for positive trees is 3. The upper bound for arbitrarily signed trees is unknown. The parallel complexity of negative paths with $2n$ vertices is 1, and of negative paths with $2n + 1$ vertices is 2. The parallel complexity of any path with either $3n$ or $3n + 1$ vertices is 2, and of any positive path with $3n + 2$ vertices is 3. Any positive complete bipartite or tripartite graph has a parallel complexity of at most 3. Upper bounds for parallel complexities of some other types of graphs are also known.

Another open problem for the topic of parallel gene assembly is to find an efficient algorithm to decide the parallel complexity of a signed graph. Currently, the most optimal known algorithm computing the parallel complexity of a graph [5] has time complexity

$$O\left(\frac{n^{n+7/2}}{c^n}\right) \text{ for } c = \frac{e}{\sqrt{2}}.$$

This is an improvement in comparison to the basic brute force algorithm [3] where all applicable parallel strategies are being checked. In this improved algorithm we are considering parallelization of sequential graph reduction strategies. Any sequential strategy that we consider is split into parallel steps in such a manner that the number of parallel steps is minimal for this strategy. Moreover, we do not consider more than one sequential strategy having the same parallelization. This approach enables us to consider instead of many parallel reduction strategies with the same domain of operations just one of them with the lowest complexity. Moreover, we have improved also on the parallel applicability decision procedure. Instead of checking the applicability of all permutations of operations from a set, we have used another approach. We based on the fact that a set of operations S is applicable in parallel to a graph G if and only if for any subset $S' \subseteq S$ sequence of operations $r \circ lex(S')$ is applicable to G, where $r \in S \setminus S'$ and $lex(S')$ is the lexicographical order of operations from S'. In this way, we have to check $k2^{k-1}$ sequences of operations $r \circ lex(S')$ to decide the parallel applicability of S. The complexity estimate of the basic algorithm in [3] grows almost as fast as $(n^n)^2$, while the present estimate of the improved algorithm in [5] grows almost as fast as n^n.

6 Computing with gene assembly

In this section we concentrate on the topic of computing by gene assembly in ciliates. Inspired by the celebrated computational experiment of Adleman with DNA [1], we are interested whether and how we can compute by using evolved DNA manipulation during the gene assembly.

The research in this direction addresses such formal language-based topics as computability, hierarchies of classes, language equations, closure properties, as well as the results on developing methods to solve computationally hard mathematical problems.

Formal languages-related results

One of the first results concerning the computability of gene assembly was obtained for the intermolecular model [39]. The concept of contextual recombinations [54] was used to simulate computations by Turing machines by using intermolecular operations. Basing on contextual version of intermolecular operations an accepting system was defined: a multiset of strings is accepted if in result of application of a sequence of contextual intermolecular operations according to the given splicing scheme [26] a multiset containing the given axiom word is obtained.

Inspired by this result we have shown in a similar manner the Turing universality for the contextual intramolecular operations [36]. Since the intramolecular model operates on a single molecule, we used the formalism of contextual string rewriting rules representing intramolecular operations. Basing on these contextual string rewriting rules we have defined an accepting intramolecular recombination system incorporating the following ingredients: the *splicing scheme*, the *start word* and the *target word*. That system accepts all those words which being concatenated to the start word produce the target word in the result of a sequence of application of contextual string rewriting rules according to the given splicing scheme. Formally, an accepting intramolecular recombination system is denoted as $G = (\Sigma, \sim, \alpha_0, w_t)$, where $(\Sigma, \sim)$ is the splicing scheme, α_0 is the start word and w_t is the target word. G accepts the following language: $L(G) = \{w \in \Sigma^* | \alpha_0 w \Rightarrow_{\widetilde{R}}^* w_t\}$. Accepting intramolecular recombination systems were proved to be universal. Instead of using multiple copies of a string as in the case of the intermolecular model we used concatenation of multiple copies of the string. For any simulation of a Turing machine we assumed that we have as many concatenations of copies of the string as needed.

It was shown in [2] that the "contextual ingredient" of the inter-

molecular model if substituted by the "distribution ingredient" does not decrease the computational power of gene assembly. In [2] there were introduced tissue P systems with ciliate operations. In general, a tissue P system is represented by an undirected graph, where multisets of objects and sets of multiset rewriting (evolution) rules are associated to each node (called *region*, or *membrane*, or *cell*). Rules define the evolution of multisets and communication between neighboring regions through *communication channels* (the graph edges) [48]. The paradigm of P systems was motivated from such biological elements as cellular membranes and membranal structure, biochemical reactions, DNA and RNA manipulations, transmembrane transportation of chemicals and other phenomena in cellular biology [52, 53, 48]. We refer to [53, 62] for the detailed overview on the research topic of P systems. In its generic definition, P systems posses an abstract nature of their objects and evolution rules. In tissue P systems with ciliate operations the objects are strings representing DNA molecules and evolution rules are intermolecular operations with transmembrane communication. It is added to the definition of the intramolecular excision and the intermolecular insertion, the information about the regions-targets for the excised circular molecule and the molecule containing non-excised sequences, as well as the information about regions-sources for the recombining molecules.

Formally, a tissue P system with ciliate operations is defined as a tuple $\Pi = (O, C, R, i_0)$, where O is a finite set of symbols, C is a finite set of regions, R is a finite set of excision/insertion rules and i_0 is the output region. To each region $c \in C$ there are associated finite sets of strings $inf(c)$ presented in infinite number of copies in c and initial finite multiset of strings $cfg(c)$. The strings can be both linear and circular and are defined over alphabet O. Each evolution rule from R has one of the following forms: *intramolecular excision rule* $i \rightarrow_p j/k$ is applicable on a string $upvpw$ (or $\circ upvpw$) in region $i \in C$ and produces strings upw (or $\circ upw$ respectively) and $\circ pv$ in region k; *intermolecular insertion rule* $j/k \rightarrow_p i$ is applicable on pair of strings upw (or $\circ upw$) in region j and $\circ pv$ in region k and produces string $upvpw$ (or $\circ upvpw$ respectively) in region i. Here p is a pointer and u, v, w are linear strings

over O. All rules from R are applied in parallel either synchronously or non-synchronously and either in maximally parallel or non-maximally parallel manner (depending on the type of the system). If in result of application of rules from R the system reaches a state where no rule from R could be used, then the multiset of words (or the number of words) in region i_0 is considered to be the *result of computation* of Π. Registers machines were chosen to be simulated by the tissue P systems with ciliate operations in order to prove the P systems Turing universality.

Among other results on gene assembly based on formal languages we can mention language operations inspired by molecular gene assembly operations and their closure properties as well as solutions of language equations [9, 12, 13, 24], language operations inspired by the template-guided DNA recombination [15, 16], generalization of intra- and intermolecular operations as synchronized insertion/deletion on linear strings [9], generalized versions of *ld* and *dlad* operations and families of languages defined by closure under those operations [10, 11].

Computing NP-complete problems

We are wondering how we can use gene assembly in order to solve computationally intractable problems. One of the advantages of gene assembly over DNA computing from the point of view of the experimental implementation might be the fact that such actions as amplification (producing a high number of clones of a DNA molecule) and filtering on the DNA molecules of interest which should be performed "manually" in the lab might be executed by ciliates "autonomously".

In this section we survey research results related to the development of computational methods for solving NP-complete problems [50] by means of gene assembly process. Generally, the gene assembly process is deterministic and confluent, i.e., starting from a gene pattern it always assembles the macronuclear gene (or in case of simple intramolecular operations all assembly strategies applicable to the gene pattern either lead to the assembled gene, or all of the strategies fail). The basic approach for solving instances of NP-complete problems is in

checking all the particular solutions for the instance of an NP-complete problem. In order to implement this approach by means of gene assembly, one has to make the gene assembly process non-deterministic and non-confluent, i.e., while starting from the same gene pattern different assembly strategies should produce different resulting molecules. This is achievable if allowing gene patterns with more than two occurrences of the same pointer (and even with several copies of the same MDS) [4]. Then, the computational method for solving NP-complete problems is looking as follows:

1. Encode an instance of the problem as a gene pattern;
2. Let the ciliate amplify the molecule with the encoded instance;
3. Let the gene assembly occur so that all copies of the molecule get assembled by different assembly strategies corresponding to different particular solutions to different resulting molecules;
4. Interpret the resulting molecules as the results of the corresponding particular solutions of the instance.

In [4] we have presented a computational method to solve instances of Hamiltonian Path Problem (HPP) inspired by the famous Adleman's experiment with DNA [1]. For a directed graph $G = (V, E)$ with n vertices and m edges a *hamiltonian path* in G from vertex p to vertex q is a noncyclic path from p to q containing all the vertices from G. The HPP is defined as the problem of deciding whether there exists a hamiltonian path from p to q.

Adleman has solved a small instance of HPP through an experimental assay on its DNA encoding. He implemented the following steps by using biotechnological tools:

Step 1: Generating random paths of the graph;

Step 2: Filtering set of paths generated at *Step 1* so that only paths from p to q remain;

Step 3: Filtering set of paths generated at *Step 2* and living only paths of exactly length n;

Step 4: Filtering set of paths generated at *Step 3* and retaining just paths containing all the vertices of the graph;

Step 5: The paths remaining after *Step 4* are the hamiltonian paths.

The instance of HPP problem was encoded as follows: for each vertex i of the graph there was designed a short single strand DNA sequence O_i of length 20 bp. An edge between vertices i and j was represented by a single strand molecule O_{ij} where the prefix O_{ij} of length 10 was the suffix of O_i, and the suffix of O_{ij} was the prefix of O_j. In his experimental asset, Adleman used a number of copies of molecules O_{ij} for each edge $(i, j) \in E$ and a number of copies $\overline{O_i}$ of complementary strands to O_i for each vertex $i \in V$.

Then, *Step 1* was implemented as follows. All the molecules O_{ij} were let to splice to the complementary sites at molecules $\overline{O_i}$ and $\overline{O_j}$ in a random way. In this way, two molecules O_{ij} and O_{jk} were attached to each other by means of molecule $\overline{O_j}$ and a double-stranded molecule representing path ijk from the graph G was produced. In this manner there were produced double-stranded DNA representing a number of paths from graph G. It was assumed that there were enough number of copies of vertex- and edge-representing molecules provided in the initial assay so that the probability of generating hamiltonian paths was sufficiently high.

Then, *Step 2* was implemented by *polymerase chain reaction*, *Step 3* by gel electrophoresis, and *Step 4* was performed by using a *biotin avidin magnetic beads system* in order to select molecules containing nucleotide sequences O_i for all $i \in V$. In this way, in result of the experiment only the molecules representing hamiltonian paths were obtained.

In this way, Adleman has demonstrated that in principle one can use DNA to compute *in vitro* computationally intractable mathematical problems. Adleman's result has motivated our work [4], where we followed similar principles and have demonstrated (albeit theoretically) how complex DNA manipulations naturally occurring in living cells could be used to compute solutions for NP-complete problems.

We developed several encodings for instances of HPP through *artificial* gene patterns, so that the results of gene assembly are the solutions of the problem, i.e., hamiltonian paths. In this way, the gene assembly is successful if and only if the problem has a solution (i.e., at least one hamiltonian path).

We used the formalism of MDS descriptors in order to encode HPP instances. For a directed graph $G = (V, E)$ we represented each vertex $p \in V$ as a pointer p, and each directed edge $(p, q) \in E$ we represented as an MDS (p, q). A path puq in the graph from vertex p to vertex q via vertices u we represented as a composite MDS (p, u, q), where u is a string over V.

Since a graph may contain more than two edges incident to a vertex p, then we may get in our encoding more than two occurrences of pointer p for the same gene pattern. Gene patterns with any number of occurrences of pointers yield non-deterministic assembly strategies, facilitating in this way the possibility to assemble molecules corresponding to many different paths in the graph. Moreover, we relaxed the conditions under which molecular operation LD could be applied on a molecule: we allowed LD to excise parts of a molecule which contain any number of MDSs.

For each of the following subsets of intramolecular operations LD, HI, DLAD, $\{\mathsf{LD}, \mathsf{DLAD}\}$, $\{\mathsf{LD}, \mathsf{HI}, \mathsf{DLAD}\}$ we have designed artificial gene patterns which could be assembled by the respective subsets of operations to a molecule represented either as $(b, p_1 u p_n, e)$ or as $(\overline{e}, \overline{p_n u p_1}, \overline{b})$, where in our HPP instance we are interested in hamiltonian paths from p_1 to p_n. Among all the assembled MDSs $(b, p_1 u p_n, e)$ and $(\overline{e}, \overline{p_n u p_1}, \overline{b})$ we choose those which contain all pointers $p \in V$ in string $p_1 u p_n$ and the length of $p_1 u p_n$ equals n. I.e., we choose the assembled MDSs which correspond to hamiltonian paths in G.

The length of the encoding on gene patterns for an instance of HPP is $O(m)$ for all subsets of intramolecular operations with the exception of only LD operations. Encoding for LD-only strategies has length $O(mn)$.

The result we presented here is purely theoretical. In order to implement this method in a lab, we have to clarify experimentally the

following questions: will a ciliate accept our artificially designed gene pattern and can two ciliates assemble our identical gene pattern into two different macronuclear genes according to our model? Since it was demonstrated experimentally, that templates guide the gene assembly process [49], probably we may tweak the gene assembly by designing our own templates [4].

A slightly different approach to solve computationally intractable problems was presented in [34, 35]. Computational methods to solve instances of *Boolean satisfiability problem* (SAT) were developed for both of the intramolecular and intermolecular models. Unlike the result with HPP from above, the formalism of contextual string rewriting rules was used.

7 Discussion

Initially, the research on the computational nature of gene assembly in stichotrichous ciliates was aiming at explaining and understanding this evolved biological DNA manipulation process. A considerable number of biologically related results were obtained. For instance, model- and strategy-independent invariant properties were discovered for gene assembly in [19, 55]. Theoretical models and the experimental evidence were presented for the short *pointer identification problem*. In particular, template-based recombination models were considered in [21, 7] and experimental work was presented in [49] describing the function of maternal RNA-templates on gene assembly. Virtual knot diagrams were presented in [7] as a physical representation of the homologous recombinations of molecule(s) during gene assembly.

Also, the research on computational properties of gene assembly has a great impact on Computer Science and Discrete Mathematics. In particular, new computational modeling techniques and computing paradigms, formal language theoretical results were obtained. Models for gene assembly and models motivated by gene assembly were introduced in terms of permutations, strings, graphs, formal languages and linear algebra. For instance, language generating systems based on gene assembly were introduced [14], non-contextual

string rewriting rules were presented in [9, 12, 13, 24] and used to study the closure properties and language equations. The template-guided recombination-based language operations were explored in [10]. Turing-completeness of both inter- and intramolecular operations was demonstrated in [25, 36, 38, 39]. The research related to the concept of parallel gene assembly was launched in [3, 5, 29, 28, 30]. For a recent detailed survey on the research topics on the computational nature of gene assembly we refer to [40]. Also, for a review on some recent results in this area we refer to [64].

Acknowledgments. The work of Vladimir Rogojin is supported by Science and Technology Center in Ukraine, project 4032. Vladimir Rogojin is on leave of absence from Institute of Mathematics and Computer Science of Academy of Sciences of Moldova, Chisinau MD-2028 Moldova.

References

[1] Adleman, L.M., Molecular computation of solutions to combinatorial problems. *Science* **226** (1994), 1021–1024.

[2] Alhazov, A., Ciliate Operations without Context in a Membrane Computing Framework. *Romanian Journal of Information Science and Technology,* **10**(4), (2007), pp. 315–322.

[3] Alhazov, A., Li, C., Petre, I., Computing the graph-based parallel complexity of gene assembly. *Theoretical Computer Science*, to appear (2008).

[4] Alhazov, A., Petre, I., Rogojin, V., Solutions to Computational Problems through Gene Assembly. *Natural Computing*, **7**(3), Springer, (2008), pp. 385–401.

[5] Alhazov, A., Petre, I., and Rogojin, V.. Computing the parallel complexity of signed graphs: an improved algorithm. *Theoretical Computer Science*, doi:10.1016/j.tcs.2009.02.028, (2009).

[6] Alhazov, A., Petre, I., Verlan, S., A sequence-based analysis of the pointer distribution of stichotrichous ciliates. *Biosystems* **101**, (2010), pp. 109–116.

[7] Angeleska, A., Jonoska, N., Saito, M., and Landweber, L.F., RNA-Template Guided DNA Assembly. *Journal of Theoretical Biology* **248**, Elsevier (2007), 706–720.

[8] Chang, W.J., Bryson, P.D., Liang, H., Shin, M.K., Landweber, L., The evolutionary origin of a complex scrambled gene. *Proceedings of the National Academy of Sciences of the US* **102**(42) (2005) 15149–15154.

[9] Daley, M., Ibarra, OH., Kari, L., Closure propeties and decision questions of some language classes under ciliate bio-operations. *Theoretical Computer Science*, **306**(1-3), (2003), pp. 19–38.

[10] Daley, M., Ibara, OH., Kari, L.,I.McQuillan,K.Nakano, The ld and dlad bio-operations on formal languages. *Autom.Lang.Comb.*, **8**(3), (2003), pp. 477–498.

[11] Daley, M., Kari, L., and McQuillan, I., Families of languages defined by ciliate bio-operations. *Theoretical Computer Science*, **320**(1), (2004), pp. 51–69.

[12] Dassow, J., and Holzer, M., Language families defined by a ciliate bio-operation: hierarchies and decidability problems. *International Journal of Foundations of Computer Science*, **16**(4), (2005), pp. 645–662.

[13] Dassow, J., Mitrana, V., and Salomaa, A., Operations and languages generating devices suggested by the genome evolution. *Theoretical Computer Science*, **270**(1-2), (2002), pp. 701–738.

[14] Dassow, J., and Vaszil, G., Ciliate bio-operations on finite string multisets. In: Ibarra OH, Dang Z (Eds.) *DLT 2006, LNCS* **4036**, (2006), pp. 168–179.

[15] Daley, M., and McQuillan, I., Template-guided DNA recombination. *Theoretical Computer Science*, **330**, (2005), pp. 237-250.

[16] Domaratzki, M., Equivalence in template-guided recombination. *Journal of Natural Computing*, **7**(3), (2007), pp. 439-449.

[17] Ehrenfeucht, A., Harju, T., Petre, I., Prescott, D. M., and Rozenberg, G., *Computation in Living Cells: Gene Assembly in Ciliates*, Springer (2003).

[18] Ehrenfeucht, A., Harju, T., Petre, I., Prescott, D. M., and Rozenberg, G., Formal systems for gene assembly in ciliates. *Theoret. Comput. Sci.* **292** (2003) 199–219.

[19] Ehrenfeucht, A., Petre, I., Prescott, D. M., and Rozenberg, G., Circularity and other invariants of gene assembly in ciliates. In: M. Ito, Gh. Păun and S. Yu (eds.) *Words, semigroups, and transductions*, World Scientific, Singapore, (2001) pp. 81–97.

[20] Ehrenfeucht, A., Prescott, D. M., and Rozenberg, G., Computational aspects of gene (un)scrambling in ciliates. In: L. F. Landweber, E. Winfree (eds.) *Evolution as Computation*, Springer, Berlin, Heidelberg, New York (2001) pp. 216–256.

[21] Prescott, D. M., Ehrenfeucht, A., and Rozenberg, G., Molecular operations for DNA processing in hypotrichous ciliates. *Europ. J. Protistology* **37** (2001) 241–260.

[22] Ehrenfeucht, A., Petre, I., Prescott, D. M., and Rozenberg, G., Universal and simple operations for gene assembly in ciliates. In: V. Mitrana and C. Martin-Vide (eds.) *Words, Sequences, Languages: Where Computer Science, Biology and Linguistics Meet*, Kluwer Academic, Dortrecht, (2001) pp. 329–342.

[23] Ehrenfeucht, A., Petre, I., Prescott, D. M., and Rozenberg, G., String and graph reduction systems for gene assembly in ciliates. *Math. Structures Comput. Sci.* **12** (2001) 113–134.

[24] Freund, R., Martin-Vide, C., and Mitrana, V., On some operations on strings suggested by gene assembly in ciliates. *New Generation Computing*, **20**(3), (2002), pp. 279–293.

[25] Hausmann, K., and Bradbury, P. C. (Eds.) *Ciliates: Cells As Organisms.* Vch Pub, (1997).

[26] Head, T., Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours. *Bulletin of Mathematical Biology*, **49**(6), (1987), pp. 737–759.

[27] Hogan, D.J., Hewitt, E.A, Orr, K.E., Prescott, D.M., Müller, K.M., Evolution of IESs and scrambling in the actin I gene in hypotrichous ciliates. *PNAS* 98 (26) (2001) 15101–15106.

[28] Harju, T., Li, C., Petre, I., Parallel Complexity of Signed Graphs for Gene Assembly in Ciliates. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, **12**(8), (2008) 731–737.

[29] Harju, T., Li, C., Petre, I., Graph Theoretic Approach to Parallel Gene Assembly. *Discrete Applied Mathematics*, **156**(18), Elsevier, (2008), pp. 3416–3429.

[30] Harju, T., Li, C., Petre, I. and Rozenberg, G., Parallelism in gene assembly, *Natural Computing*, 5 (2) (2006) 203–223.

[31] Harju, T., Li, C., Petre, I. and Rozenberg, G., Complexity Measures for Gene Assembly. In: K. Tuyls (Eds.), Proceedings of the *Knowledge Discovery and Emergent Complexity in Bioninformatics* workshop. Springer, *Lecture Notes in Bioinformatics* **4366**, 2007.

[32] Harju, T., Petre, I., and Rozenberg, G., Modelling simple operations for gene assembly. In: J.Chen, N.Jonoska, G.Rozenberg (Eds.) *Nanotechnology: Science and Computation* (2006) 361–376.

[33] Harju, T., Petre, I., Rogojin, V. and Rozenberg, G., Patterns of Simple Gene Assembly in Ciliates, *Discrete Applied Mathematics*, **156**(14), Elsevier, (2008), pp. 2581–2597.

[34] Ishdorj, T.-O., Loos, R., and Petre, I., Computational Efficiency of Intermolecular Gene Assembly. *Fundamenta Informaticae*, **84**(3-4), (2008), pp. 363–373.

[35] Ishdorj, T.-O., and Petre, I., Computing Through Gene Assembly. *Unconventional Computation*, **4618**, (2007), pp. 91–105.

[36] Ishdorj, T.-O., Petre, I. and Rogojin, V, Computational Power of Intramolecular Gene Assembly. *International Journal of Foundations of Computer Science*, **18**(5), World Scientific, (2007), pp. 1123–1136.

[37] Jahn, C. L., and Klobutcher, L. A., Genome remodeling in ciliated protozoa. *Ann. Rev. Microbiol.* **56** (2000), 489–520.

[38] Kari, L., Kari, J., and Landweber, L. F., Reversible molecular computation in ciliates. In: J. Karhumäki, H. Maurer, G. Păun, and G. Rozenberg (eds.), *Jewels are Forever*, Springer, Berlin Heidelberg, New York, (1999), pp. 353–363.

[39] Kari, L., and Landweber, L. F., Computational power of gene rearrangement. In: E. Winfree and D. K. Gifford (eds.) *Proceedings of DNA Bases Computers, V* American Mathematical Society (1999) 207–216.

[40] Brijder, R., Daley, M., Harju, T., Jonoska, N., Petre, I., and Rozenberg, G., Computational nature of gene assembly in ciliates. In: L. Kari, G. Rozenberg (Eds.), *Handbook of Natural Computing*, Springer, (2009), to appear.

[41] Kari, L., and Thierrin, G., Contextual insertion/deletions and computability. *Information and Computation*, **131**, (1996), pp. 47–61.

[42] Landweber, L. F., and Kari, L., The evolution of cellular computing: Nature's solution to a computational problem. In: *Proceedings of the 4th DIMACS Meeting on DNA-Based Computers*, Philadelphia, PA (1998) pp. 3–15.

[43] Landweber, L. F., and Kari, L., Universal molecular computation in ciliates. In: L. F. Landweber and E. Winfree (eds.) *Evolution as Computation*, Springer, Berlin Heidelberg New York (2002).

[44] Langille, M., Petre, I., Simple gene assembly is deterministic. *Fundamenta Informaticae* **72**, IOS Press, (2006), pp. 1–12.

[45] Langille, M., Petre, I., Rogojin, V., Three models for gene assembly in ciliates: a comparison. *Proceedings of BIONETICS 2008*, to appear (2009).

[46] Lynn, D.H., Small, E.B., A Revised Classification of the Phylum Ciliophora Doflein, 1901. *Revista de la Sociedad Mexicana de Historia Natural*, Mexico, 47, (1997), pp. 65–78.

[47] Marcus, S., Contextual Grammars. *Revue Roumaine de Matematique Pures et Appliquees,*, **14**, (1969), pp. 1525–1534.

[48] Martin-Vide, C., Păun, G., Pazos, J., and Rodriguez-Paton, A., Tissue P systems. *Theoretical Computer Science*, **296**(2), pp. 295–326. DOI: 10.1016/S0304-3975(02)00659-X.

[49] Nowacki, M., Vijayan, V., Zhou, Y., Schotanus, K., Doak, T.G., Landweber, L.F., RNA-mediated epigenetic programming of a genome-rearrangement pathway. *Nature* **451**, doi:10.1038/nature06452, (2008) 153–158.

[50] Papadimitriou, C.H., *Computational Complexity*. Addison-Wesley, (1994).

[51] Păun, G., *Marcus Contextual Grammars*, Kluwer, Dordrecht, (1997).

[52] Păun, G., Computing with Membranes, *TUCS Technical Report*, **208**, (1998).

[53] Păun, G., *Membrane Computing: An Introduction.* Springer, (2002).

[54] Păun, G., Rozenberg, G., Salomaa, A., *DNA Computing: New Computing Paradigms.* Springer, (1998).

[55] Petre, I., Invariants of gene assembly in stichotrichous ciliates. *IT*, Oldenbourg Wissenschftsverlag, **3** (2006), pp. 161–167.

[56] Petre, I., and Rogojin, V., Decision Problem for Shuffled Genes, *Information and Computation*, **206**(11), Elsevier, (2008), pp. 1346–1352.

[57] Prescott, D. M., DNA manipulations in ciliates. In: W. Brauer, H. Ehrig, J. Karhumäki, A. Salomaa (eds.) *Formal and Natural Computing: essays dedicated to Grzegorz Rozenberg*, LNCS 2300, Springer (2002) 394–417.

[58] Prescott, D. M., Genome gymnastics: unique modes of DNA evolution and processing in ciliates. *Nature Reviews, Genetics*, **1**(3), (2000), pp. 191–198.

[59] Prescott, D. M., The DNA of ciliated protozoa. *Microbiol. Rev.* **58**(2) (1994) 233–267.

[60] Prescott, D. M., Ehrenfeucht, A., and Rozenberg, G., Template-guided recombination for IES elimination and unscrambling of genes in stichotrichous ciliates. *Journal of Theoretical Biology*, **222**(3), (2003), pp. 323–330.

[61] Prescott, D. M., and Rozenberg, G., How ciliates manipulate their own DNA A splendid example of natural computing. *Natural Computing*, **1**, (2002), pp. 165-183.

[62] Păun, G., Membrane Computing. *Scholarpedia*, **5**(1):9259.

[63] Rogojin, V., Successful Elementary Gene Assembly Strategies. *International Journal of Foundations of Computer Science*, to appear, (2009).

[64] Rogojin, V., Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation. *TUCS Dissertations*, **117**, 2009.

[65] Wright, A.-D. G. and Lynn, D. H., Maximum ages of ciliate lineages estimated using a small subunit rRNA molecular clock: Crown eukaryotes date back to the Paleoproterozoic. *Arch Protistenkd*, **148**, (1997), pp. 329-341.

[66] Yao, M.C., Fuller, P., Xi, X., Programmed DNA Deletion As an RNA-Guided System of Genome Defense, *Science* 300 (2003) 1581–1584.

V. Rogojin

Received November 21, 2010

University of Helsinki
Faculty of Medicine
Genome-Scale Biology Research Program
Computational Systems Biology Laboratory
Biomedicum, Helsinki 00014, Finland
Phone: +358 919 125 407
E–mail:*vladimir.rogojin@helsinki.fi*, *vrogojin@vrogojin.net*

Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
5 Academiei str., Chişinău, MD-2028, Moldova