

How Complex is to Solve a Hard Problem with Accepting Splicing Systems

Victor Mitrana^{1,3} ^a, Andrei Păun² ^b and Mihaela Păun³ ^c

¹Department of Information Systems, Polytechnic University of Madrid, Crta. de Valencia km. 7 - 28031 Madrid, Spain

²Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei 14, 010014 Bucharest, Romania

³National Institute for Research and Development of Biological Sciences, Independentei Bd. 296, Bucharest, Romania

Keywords: Splicing, Accepting Splicing System, Computational Complexity, Descriptive Complexity, 3-colorability Problem.

Abstract: We define a variant of accepting splicing system that can be used as a problem solver. A condition for halting the computation on a given input as well as a condition for making a decision as soon as the computation has stopped is considered. An algorithm based on this accepting splicing system that solves a well-known NP-complete problem, namely the 3-colorability problem is presented. We discuss an efficient solution in terms of running time and additional resources (axioms, supplementary symbols, number of splicing rules. More precisely, for a given graph with n vertices and m edges, our solution runs in $O(nm)$ time, and needs $O(mn^2)$ other resources. Two variants of this algorithm of a reduced time complexity at an exponential increase of the other resources are finally discussed.

1 INTRODUCTION

One of the basic phenomenon in genetic engineering is that by which genetic material is recombined. This phenomenon, called *splicing*, allows to genetically modify a biological entity for different purposes like: more resistant plants, organisms better adapted to weather changes, production of hormones, etc. The chemicals involved in the recombination of DNA sequences are two types of enzymes: restriction enzymes which cut the DNA at specific sites (called recognition sites) yielding two fragments with the so-called “sticky ends”, and ligase which rejoin fragments with sticky ends. A computational model based on an operation abstracted from the splicing operation described above has been defined in (Head, 1987). The model viewed as a language generating device is called *splicing system*. Roughly speaking, the two DNA molecules are represented by strings while the restriction enzymes are represented by quadruples of strings, called splicing rules, indicating the sites where the two strings are to be cut. The compatibility for rejoining is defined by the fact that two fragments

can be rejoined if they were obtained by applying the same splicing rule. The splicing operation is illustrated in Figure 1.

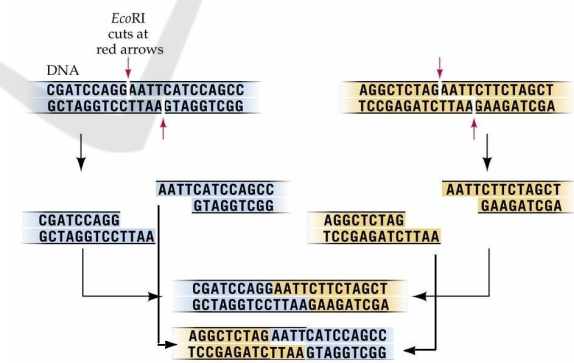





Figure 1: Splicing operation.

Most research in the area of splicing systems has been focused on defining different types of such systems and investigating their computational power from a language generating point of view. For several years, it was an open problem whether or not splicing systems are more powerful than finite automata, from a computational power point of view. The problem was first solved in (Culik and Harju, 1991), and

^a  <https://orcid.org/0000-0002-1457-8933>

^b  <https://orcid.org/0000-0002-1644-8198>

^c  <https://orcid.org/0000-0002-3342-9140>

later on, a constructive proof was proposed in (Pixton, 1996). In these papers it was proved that splicing systems are strictly weaker than finite automata. In order to increase the computational power of these systems, many variants have been defined and investigated; we mention here just a few of them: distributed splicing systems (Csuhaaj-Varjú et al. 1996), extended splicing systems (Păun et al. 1996), splicing systems with multisets (Denninghoff and Gatterdam, 1989), splicing systems with permitting and forbidding contexts (Freund et al. 1999), splicing systems with a regular set of rules (Păun, 1996), programmed and evolving splicing systems (Păun et al. 1997). Under certain circumstances, splicing systems are shown to be able to simulate Turing machines, see (Păun et al. 1998). This result suggests the possibility to consider splicing systems as theoretical models of programmable universal DNA computers based on the splicing operation. Two comprehensive surveys can be found in (Head et al. 1997) and (Păun et al. 1998).

Splicing systems working with circular strings are inspired by a recombinant behavior of circular DNA in plasmids. Two variants of circular splicing systems have been introduced in (Siromoney et al. 1992) and (Pixton, 1995)

Networks with nodes hosting splicing processors (NSP) have been considered in (Manea et al. 2007). The NSP model resembles some features of another distributed computing system, namely test tube distributed splicing systems introduced in (Csuhaaj-Varjú et al. 1996) and further investigated in (Păun, 1998). A characterization of the complexity class **NP** as the class of languages accepted by restricted NSPs in polynomial time was proposed. Furthermore, a similar characterization was proposed for the class **PSPACE** as the class of languages accepted by restricted NSPs with at most polynomial length of the strings used in the derivation. In (Manea et al. 2006) it was proved that NSPs (unrestricted, this time) of constant size accept all recursively enumerable languages, and can solve all problems in **NP** in polynomial time; also an universality result for NSPs was proposed. In both cases the number of nodes needed was 7. In (Loos et al. 2009) it is shown that computational completeness can be achieved by NSPs of 2 nodes. In the same paper a more involved construction, showing that NSPs of size 3 can simulate the computations of a non-deterministic Turing machine in parallel, is presented.

A new protocol of cooperation in networks of splicing processors, namely the polarization, has been introduced in (Bordihn et al. 2017) and further investigated in (Bordihn et al. 2018). The polarization protocol requires that each node has a polariza-

tion defined as a value in the set $\{+, 0, -\}$ (positive, neutral, negative) and the data have assigned a value in the same set which is computed by a valuation mapping. Now the communication is based on the compatibility between the polarization of nodes and the value of data. A quantitative generalization of these networks, called networks of splicing processors with evaluation sets have been introduced in (Gómez-Canaval et al. 2016). In a network of splicing processors with evaluation sets, unlike in all the aforementioned cases, the valuation mapping returns the exact value computed for data. The new model refines the communication protocol based on polarization discussed above, in which each polarization may be viewed as one of the intervals of integers $(-\infty, 0)$, $\{0\}$, and $(0, \infty)$, to a more complex polarization based on more intervals. The new model tries to resemble the biological concept of the concentration gradient in a solution. Now, the strategy of communication between two nodes follows the compatibility between their accepting values with respect to some predefined evaluation sets and the values of the data computed by a valuation mapping. More precisely, the values of data have to be in the set of accepting values with respect to some evaluation set of symbols. This new communication protocol might be interpreted as the movement of molecules or particles along a concentration gradient between two areas.

Other fundamental algorithmic problems regarding splicing systems, namely *recognition* and *synthesis* are considered in a series of papers starting with (Bonizzoni and Mauri, 2005). Recognition problem refers to the design of an algorithm able to decide whether or not a given regular language is a splicing language, while the synthesis problem refers to the possibility of effectively constructing a splicing system that generates a given regular language.

A surprising application of splicing systems which are used as an automatic music composer is proposed in (De Felice et al. 2017). This approach might be seen as a possible bio-inspired strategy for automatic music composition. The model in the aforementioned work is tailored on 4-voice chorale-like music. Along these lines, a new approach for both recognition and automatic composition of styles for musical collectives is designed in (De Prisco et al. 2017). Informally speaking, such a system exploits a machine learning recognizer, based on one-class support vector machines and neural networks for style recognition, and a splicing composer, for music composition (in the style of the whole collective).

Starting from some ideas considered in (Loos et al. 2006), a novel look at splicing systems is proposed in (Mitrana et al. 2010), namely these systems

are considered as accepting devices. More precisely, one defines the concept of *accepting splicing system*. The rough idea is to consider that an accepting splicing system receives as input a string which enters, together with a given finite set of axioms, into an iterated splicing process until a string from a specific finite sets of strings is produced. When such a string is produced, the computational process halts. In this paper, we consider the case when a decision can be made as soon as the computation halts. To this aim, we consider a two finite sets of strings, a set of halting strings such that when a halting string is produced the computation halts, and a set of accepting strings. For making a decision, it suffices to check whether or not at least one accepting string has been obtained.

The paper is organized as follows. In the next section we present the basic definitions and the main concept, that of an accepting splicing system viewed as a problem solver. We define one computational complexity measure, similar to the time complexity in most models, which measures the number of splicing steps necessary for a computation to halt. Then we define a few descriptive complexity measures like: the number of axioms, the number of supplementary symbols, the number of rules, the length of the longest splicing rule, and the number of halting or accepting strings. In the third section, we consider a very well-known NP-complete problem, that of 3-colorability, and propose an algorithm based on accepting splicing systems (an *AccSS*-algorithm) to solve this problem. We give both informal explanations as well as formal definitions on the way our algorithm works. Afterwards, we evaluate the computational and descriptive complexity of our algorithm. Thus, we prove that given a graph with n vertices and m edges, there is an *AccSS*-algorithm that decides the 3-colorability for this graph in $O(nm)$ time and $O(mn^2)$ other resources (axioms, symbols, splicing rules). Finally, we discuss two variants of this algorithm that need a reduced time complexity, but this is achieved at an exponential increase of the other resources.

2 MAIN CONCEPT

We start by summarizing the notions used throughout the paper. For all undefined notions the reader may consult (Rozenberg and Salomaa, 1997). An *alphabet* is a finite and nonempty set of symbols. The cardinality of a finite set A is written $\text{card}(A)$. Any finite sequence of symbols from an alphabet V is called *string* over V . The set of all strings over V is denoted by V^* and the empty string is denoted by λ . The length of a string x is denoted by $|x|$ while $\text{alph}(x)$ denotes the

minimal alphabet W such that $x \in W^*$.

A splicing rule over V is 4-tuple $[(u_1, u_2); (u_3, u_4)]$, with $u_1, u_2, u_3, u_4 \in V^*$. For a splicing rule $r = [(u_1, u_2); (u_3, u_4)]$ and a pair of strings $x, y \in V^*$, we write

$$\sigma_r(x, y) = \{y_1 u_3 u_2 x_2 \mid x = x_1 u_1 u_2 x_2, y = y_1 u_3 u_4 y_2\} \\ \cup \{x_1 u_1 u_4 y_2 \mid x = x_1 u_1 u_2 x_2, y = y_1 u_3 u_4 y_2\}$$

for some $x_1, x_2, y_1, y_2 \in V^*$. The analogy with the situation depicted in Figure 1 is immediate: the two strings represent the two double stranded DNA molecules, while the restriction enzyme is represented by the splicing rule. This definition is extended to a set of splicing rules R and a set of strings A by

$$\sigma_R(A) = \bigcup_{r \in R} \bigcup_{w_1, w_2 \in A} \sigma_r(w_1, w_2).$$

When the set of splicing rules is clear, we omit the subscript.

Note that this definition assumes that arbitrarily many copies of all strings in A are available for splicing. This is a natural assumption from the biological point of view because every such string, encoding a DNA molecule, can be duplicated sufficiently many times by the PCR procedure, a well defined technique for amplifying the genetic material (Rabinov, 1996).

We now recall the definition and terminology for accepting splicing systems, following (Mitrana et al. 2010). It is worth noting that the definition presented here differs from those in (Mitrana et al. 2010) and (Arroyo et al. 2013) because we use here accepting splicing systems as decision problem solvers. To this aim, we need a condition for halting the computation and a condition for making the decision. Furthermore, we need that all systems halt on every input. An *accepting splicing system* (*AccSS* for short) is a 8-tuple

$$\Gamma = (V, U, <, >, A, R, H, F),$$

where V is the input alphabet, U is the working alphabet, $V \subset U$, $<, >$ are two symbols in $U \setminus V$ called endmarkers, $A \subseteq U^*$ is the set of initial strings (axioms), while R is a set of splicing rules over U . Furthermore, H and F are finite sets of strings over U ; the elements of H are called *halting* strings and those of F are called *accepting* strings.

Let $\Gamma = (V, U, <, >, A, R, H, F)$ be an *AccSS* and a string $w \in V^*$; we define the following iterated splicing

$$\tau_R^1(A, w) = \sigma_R(A \cup \{w\}), \\ \tau_R^{i+1}(A, w) = \sigma_R(\tau_R^i(A, w) \cup A), i \geq 1.$$

Again, the subscript R may be omitted when it is self-understood. Note that if no splicing rule is applicable to a string different than an axiom at some step, it

disappears from the set of all available strings in the next splicing step. A computation of Γ on w is the sequence of sets $(\tau_R^i(A, w))_{i \geq 0}$. Such a computation is finite (we say that Γ halts on $w \in V^*$) if there exists $k \geq 0$ such that $\tau_R^k(A, w) \cap H \neq \emptyset$. An input string w is accepted by Γ if Γ halts on w as above and $\tau_R^k(A, w) \cap F \neq \emptyset$; otherwise, w is rejected. A string W is decided by Γ if Γ halts on w . A set of strings L is decided by Γ if Γ accepts all strings in L and rejects all strings that do not belong to L . As we want to consider *AccSS* as problem solvers, we assume that every *AccSS* in what follows halts on every input string.

We now define the following computational complexity measure for an *AccSS* $\Gamma = (V, U, <, >, A, R, F)$ that halts on every input:

$$\begin{aligned} \text{Time}_\Gamma(w) &= \min\{k \mid \tau_R^k(A, w) \cap H \neq \emptyset\}, \\ \text{Time}_\Gamma(n) &= \max\{\text{Time}_\Gamma(w) \mid |w| = n\}. \end{aligned}$$

It is worth noting that a similar measure with a different definition was introduced in (Loos and Ogihara, 2007) for generating splicing systems.

We also define the following descriptonal complexity measures for an *AccSS* $\Gamma = (V, U, <, >, A, R, F)$ that halts on every input:

$$\begin{aligned} Ax(\Gamma) &= \text{card}(A), \\ \text{Symb}(\Gamma) &= \text{card}(U \setminus V), \\ \text{NSplice}(\Gamma) &= \text{card}(R), \\ \text{LSplice}(\Gamma) &= \max\{|u_1u_2u_3u_4| \mid \\ &\quad [(u_1, u_2); (u_3, u_4)] \in R\}, \\ \text{Final}(\Gamma) &= \max(\text{card}(H), \text{card}(F)). \end{aligned}$$

We mention here another work considering some descriptonal complexity measures for generating splicing systems, namely (Loos et al. 2008). In this paper, the measures are the total length of the rules and the size of the initial language. These measures are related to the size of the minimal finite automata accepting the same language.

We discuss below the potential of *AccSS* to solve complex decidability problems. We now formally define what an *AccSS*-algorithm is. A decidability problem \mathcal{P} is said to be solved in time $O(t(n))$ by *AccSS*s if there exists a family \mathcal{G} of *AccSS*s, which can be constructed by an effective procedure, such that for each instance x of size $O(n)$ of the problem \mathcal{P} , encoded by a string w , one can effectively construct an *AccSS* $\Gamma(x) \in \mathcal{G}$ such that the computation of $\Gamma(x)$ on w halts in time $O(t(n))$ with at least an accepting string obtained during the computation if and only if the x is a true instance. This effective construction is called an $O(t(n))$ time *AccSS*-algorithm for the considered problem.

3 AN *AccSS*-ALGORITHM FOR THE 3-COLORABILITY PROBLEM

Given a connected undirected graph without loops, the 3-colorability problem is to decide whether or not is it possible to color each vertex by using three colors (say, red, blue, and green) such that no two adjacent vertices are of the same color. For instance, the graph in Figure 2 is correctly colored.

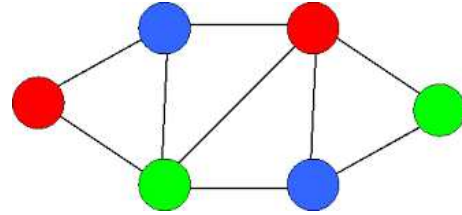


Figure 2: Graph correctly colored with three colors.

The problem is known to be **NP**-complete (Garey and Johnson, 1979). Let $Y = (B, Q)$ be a graph with set of vertices $B = \{1, 2, \dots, n\}$ and set of edges $Q = \{e_1, e_2, \dots, e_m\}$, where each e_k is given in the form $e_k = \{k_1, k_2\}$, for some $1 \leq k_1 \neq k_2 \leq n$.

We now present an *AccSS*-algorithm for the 3-colorability problem and analyze its computational and descriptonal complexity.

3.1 *AccSS*-algorithm

The input string is $\langle a^n e_1 e_2 \dots e_m \rangle$. The algorithm has two main phases. In the first phase all strings encoding graph colorings, namely strings of the form

$$\langle x_1 x_2 \dots x_n e_1 e_2 \dots e_m \rangle,$$

where $x \in \{r, b, g\}$ are produced in parallel by splicing. The meaning of this encoding is that the node i is colored by red, blue, green if r_i, b_i, g_i , respectively. In the second phase, the graph colorings obtained in the first phase are to be checked for correctness, that is they are correct colorings. All the strings encoding correct colorings will lead simultaneously to accepting strings. If the graph cannot be correctly colored, the computation will halt with no accepting string obtained.

A complete formal solution is presented in the next two sections. The general idea is that used by most of the DNA-based algorithms: a first phase is dedicated to the generation of solution space, and a second phase which searches the correct solutions by filtering the whole solution space. The parameters of Γ are to be constructed dynamically as follows. We give the splicing rules for each phase, hence the set

of splicing rules R of Γ is the set containing all these rules. Similarly, the set of axioms A contains all the axioms presented for each phase. The working alphabet U is formed by all symbols that appear in the axioms and splicing rules listed for each phase.

3.1.1 First Phase

The first phase can be accomplished with the following splicing rules and axioms distributed in three groups, each of them having subgroups:

1. $[(\langle, a); (\langle C_1, \#)],$ and $\langle C_1\#$ is an axiom.
- 2.1. $[(\dagger, C_1a); (X_i, Y_i)], \dagger \in \{\langle\} \cup \{b_{i-1}, g_{i-1}, r_{i-1}\}$
and X_iY_i is an axiom, $1 \leq i \leq n$.
- 2.2. $[(X_iC_1a, \dagger_1); (X'_i\dagger_2C_{i+1}, \#)], \dagger_1 \in \{a, e_1\},$
 $\dagger_2 \in \{b_i, g_i, r_i\},$ and $X'_i\dagger_2C_{i+1}\#$ is an axiom,
 $1 \leq i \leq n-1$.
- 2.3. $[(X_nC_n a, e_1); (X'_n\dagger, \#)], \dagger \in \{b_n, g_n, r_n\},$
and $X'_n\dagger\#$ is an axiom.
- 2.4. $[(\langle, Y_i); (\#, Y'_i)],$ and $\#Y'_i$ is an axiom, $1 \leq i \leq n$.
- 3.1. $[(\langle, Y'_1); (X'_1, \dagger C_2a)], \dagger \in \{b_1, g_1, r_1\}$.
- 3.2. $[(\dagger_1, Y'_{i-1}); (X'_{i-1}, \dagger_2 C_ia)], \dagger_1 \in \{b_{i-2}, g_{i-2}, r_{i-2}\},$
 $\dagger_2 \in \{b_{i-1}, g_{i-1}, r_{i-1}\}, 3 \leq i \leq n$.
- 3.3. $[(\dagger_1, Y'_n); (X'_n, \dagger_2 e_1)], \dagger_1 \in \{b_{n-1}, g_{n-1}, r_{n-1}\},$
 $\dagger_2 \in \{b_n, g_n, r_n\}$.

We explain how an arbitrary string $\langle x_1x_2 \dots x_n e_1 e_2 \dots e_m \rangle$, with $x \in \{r, b, g\}$, is produced from the input string $w = \langle a^n e_1 e_2 \dots e_m \rangle$ by using the splicing rules listed above. The first splicing rule that can be applied is the rule in the group 1 which can be applied to the pair of strings formed by the input string and the axiom $\langle C_1\#$. This splicing step yields two new strings $\langle C_1 a^n e_1 e_2 \dots e_m \rangle$ and $\langle \#$. The string $\langle \#$ cannot be involved in further splicing steps, hence it will disappear in the next splicing step. Therefore, we can say that the role of the first splicing step is to introduce the symbol C_1 before the first occurrence of a in the input string. The rough idea of the next splicing steps is that symbols C_i , $1 \leq i \leq n$, scan the input string from left to right following the next procedure: each C_i scans the symbol a next to it, replaces this occurrence of a by a symbol in $\{b_i, g_i, r_i\}$, moves to the right of this symbol, and changes itself into either C_{i+1} , if $i < n$ or it is removed, if $i = n$. Inductively, this happens as follows. Assume that the current string is $\langle x_1x_2 \dots x_{i-1} C_i a^{n-i+1} e_1 e_2 \dots e_m \rangle$, with $x \in \{r, b, g\}$, and $i < n$. A rule in the subgroup 2.1 is applied to the current string and the axiom X_iY_i . Two strings are obtained $\langle x_1x_2 \dots x_{i-1} Y_i$ and $X_i C_i a^{n-i+1} e_1 e_2 \dots e_m \rangle$. Now, a rule in the subgroup 2.2 is applied to $X_i C_i a^{n-i+1} e_1 e_2 \dots e_m \rangle$ and to

one of the axioms $X'_i \dagger C_{i+1} \#$, where $\dagger \in \{b_i, g_i, r_i\}$. Thus three new words are obtained simultaneously: $X_i b_i C_{i+1} a^{n-i} e_1 e_2 \dots e_m \rangle$, $X_i g_i C_{i+1} a^{n-i} e_1 e_2 \dots e_m \rangle$, and $X_i r_i C_{i+1} a^{n-i} e_1 e_2 \dots e_m \rangle$. In the same splicing step, a rule in subgroup 2.4 is applied to an axiom $\#Y'_i$ and to $\langle x_1x_2 \dots x_{i-1} Y_i$. As a result of this splicing the string $\langle x_1x_2 \dots x_{i-1} Y'_i$ is obtained. Now the process of coloring the node i ends by applying a splicing rule in the subgroup 3.2 to each of the three strings aforementioned and to one of the strings $\langle x_1x_2 \dots x_{i-1} Y'_i$. All the strings $\langle x_1x_2 \dots x_i C_{i+1} a^{n-i} e_1 e_2 \dots e_m \rangle$ are now produced.

It is worth mentioning that the rules in subgroups 2.3 and 3.3 replace the rules in 2.2 and 3.2, respectively, when $i = n$ in the above discussion. Now the whole process described above is iterated until all strings $\langle x_1x_2 \dots x_n e_1 e_2 \dots e_m \rangle$, with $x \in \{r, b, g\}$ are produced.

A short discussion is necessary here. First, it is easy to note that all the other strings that are by-products of the splicing rules applied in the process discussed above will not be involved in further splicing steps because they cannot be cut by any splicing rule. Second, when applying a rule in the third group the two strings that are joined do not necessarily come from the same string cut with rules in the subgroup 2.1, but this fact does not introduce any illegal string, that is all the strings obtained after applying rules in the third group are still valid partial colorings of the graph. Note that by partial coloring we mean that every node of the graph is either colored with exactly one color or it is not colored yet.

3.1.2 Second Phase

The second phase starts when the first phase is finished, that is when all strings $\langle x_1x_2 \dots x_n e_1 e_2 \dots e_m \rangle$, with $x \in \{r, b, g\}$ are produced. Informally, the second phase checks whether every such string as above is a correct coloring. This means that for each $e_k = \{k_1, k_2\}$, $x_{k_1} \neq x_{k_2}$ holds. This is to be done by checking this condition for e_1, e_2, \dots, e_m .

For a better understanding of this checking process we need some preparatives. For every symbol e_k encoding the edge $\{k_1, k_2\}$ we write \bar{e}_k , \hat{e}_k , and \tilde{e}_k if one of the two nodes k_1, k_2 is colored blue, green, and red, respectively, while the other node is not colored yet. Now the checking process is to be accomplished as follows. The prefix $\langle x_1x_2 \dots x_n$ of each of the strings $\langle x_1x_2 \dots x_n e_1 e_2 \dots e_m \rangle$, with $x \in \{r, b, g\}$ will be scanned by every symbol e_k , $1 \leq k \leq m$, from right to left such that:

- (i) If e_k meets b_{k_1} or b_{k_2} to its left, then e_k is replaced

- by \bar{e}_k .
- (ii) If e_k meets g_{k_1} or g_{k_2} to its left, then e_k is replaced by \hat{e}_k .
 - (iii) If e_k meets r_{k_1} or r_{k_2} to its left, then e_k is replaced by \tilde{e}_k .
 - (iv) If \bar{e}_k meets a symbol in $\{g_{k_1}, r_{k_1}, g_{k_2}, r_{k_2}\}$ to its left, then \bar{e}_k is deleted.
 - (v) If \hat{e}_k meets a symbol in $\{b_{k_1}, r_{k_1}, b_{k_2}, r_{k_2}\}$ to its left, then \hat{e}_k is deleted.
 - (vi) If \tilde{e}_k meets a symbol in $\{g_{k_1}, b_{k_1}, g_{k_2}, b_{k_2}\}$ to its left, then \tilde{e}_k is deleted.
 - (vii) In all the other cases, \bar{e}_k , \hat{e}_k , and \tilde{e}_k are replaced by a new symbol Q , which will continue to scan the rest of the string until either the left endmarker or another occurrence of Q is met.
 - (viii) If a symbol x_r was scanned by all symbols e_k , $1 \leq k \leq m$, then it is deleted.

We give now the formal definitions of splicing rules, distributed in groups and subgroups, and those of axioms.

1. $[(\clubsuit, x_i Z_j); (\diamond_{i,j}, \heartsuit_{i,j})]$ and $\diamond_{i,j} \heartsuit_{i,j}$ is an axiom,
 $\clubsuit \in \{<, Q\} \cup \begin{cases} \{x_{i-1}, \text{ if } i > 1 \\ \emptyset, \text{ if } i = 1 \end{cases}$
 $Z_j \in \{e_j, \bar{e}_j, \hat{e}_j, \tilde{e}_j\}, 1 \leq j \leq m, 1 \leq i \leq n$.
- 2.1. $[(\clubsuit, \heartsuit_{i,j}); (\#, \heartsuit'_{i,j})]$ and $\# \heartsuit'_{i,j}$ is an axiom,
 $\clubsuit \in \{<, Q\} \cup \begin{cases} \{x_{i-1}, \text{ if } i > 1 \\ \emptyset, \text{ if } i = 1 \end{cases}$
 $1 \leq j \leq m, 1 \leq i \leq n$.
- 2.2. $[(\diamond_{i,j} x_i Z_j, x_{i+1}); (\diamond'_{i,j} Y_j x_i, \#)]$,
 and $\diamond'_{i,j} Y_j x_i \#$ is an axiom, $x \in \{b, g, r\}$,
 $Z_j \in \{e_j, \bar{e}_j, \hat{e}_j, \tilde{e}_j\}$ and
 - if $i \notin \{j_1, j_2\}$, then $Y_j = Z_j$,
 - if $i \in \{j_1, j_2\}$, and $Z_j = e_j$, then

$$Y_j = \begin{cases} \bar{Z}_j, \text{ if } x = b, \\ \hat{Z}_j, \text{ if } x = g, \\ \tilde{Z}_j, \text{ if } x = r, \end{cases}$$
 - if $i \in \{j_1, j_2\}$, then $Y_j = \lambda$ provided that
 - $x \in \{g, r\}$ and $Z_j = \bar{e}_j$,
 - $x \in \{b, r\}$ and $Z_j = \hat{e}_j$,
 - $x \in \{b, g\}$ and $Z_j = \tilde{e}_j$,
 - $Y_j = Q$, in all the other cases,
 $1 \leq j \leq m, 1 \leq i \leq n$.
3. $[(\clubsuit, \heartsuit'_{i,j}); (\diamond'_{i,j}, Z_j x_i P)]$, $Z_j \in \{Q, e_j, \bar{e}_j, \hat{e}_j, \tilde{e}_j, \lambda\}$,
 $\clubsuit \in \{<, Q\} \cup \begin{cases} \{x_{i-1}, \text{ if } i > 1 \\ \emptyset, \text{ if } i = 1 \end{cases}$
 $P \neq \#, 1 \leq j \leq m, 1 \leq i \leq n$.

In order to clarify the roles of these splicing rules we show how they are used for fulfilling the conditions (i)-(vi) and partly (vii) defined above. Let

$< \alpha x_i Z_j \beta$, for some strings α, β , be an arbitrary string available at this splicing step. We take the axiom $\diamond_{i,j} \heartsuit_{i,j}$ and apply a rule in group 1 to this pair of strings. The two new strings are: $\alpha \heartsuit_{i,j}$ and $\diamond_{i,j} x_i Z_j \beta$. Now, in the same splicing step, $\alpha \heartsuit_{i,j}$ is transformed into $\alpha \heartsuit'_{i,j}$ by using a rule in subgroup 2.1, while $\diamond_{i,j} x_i Z_j \beta$ is spliced by a rule in subgroup 2.2. As one can easily see, all the conditions (i)-(vi) are considered in the definition of rules in subgroup 2.2. Moreover, the condition for introducing the new symbol Q is also considered. The meaning for introducing Q is that the coloring encoded by the string $< \alpha x_i Z_j \beta$ is illegal for the edge e_j .

For the second part of condition (vii) and condition (viii) we give only some informal explanations without formally defining the rules, which can be easily defined by the reader. The condition (viii) assumes that a symbol x_i reached the right endmarker $>$. This symbol can be easily deleted now by using one splicing rule that cuts the string just before $x_i >$ and replaces this segment by $>$ only. As soon as a symbol Q has still to scan symbols from right to left, this can be easily done in three simple splicing steps using rules similar to those in subgroups 2.1 (first splicing step), 2.2 and 2.4 (second splicing step), and 3.2 (third splicing step).

When no splicing rule can be applied anymore, only strings of the form $< Q^s >$, for some $1 \leq s \leq m$ and possibly one string more, namely $< >$, have been produced. By the aforementioned explanations, we conclude that each string obtained in the first phase, which does not encode a legal coloring, will eventually lead, at the end of the second phase, to a string $< Q^s >$, for some $1 \leq s \leq m$. On the other hand, each string obtained in the first phase, which does encode a legal coloring, will eventually lead, at the end of the second phase, to $< >$. It is clear that, if $< >$ is obtained, it is produced before any of the strings $< Q^s >$. Now, it suffices to reduce, by splicing, all the strings $< Q^s >$ to just one, namely $< Q >$ and consider the set of halting strings $H = \{< >, < Q >\}$ and $F = \{< >\}$, and the construction of the AccSS-algorithm is complete.

3.1.3 Evaluating Time Complexity

We first evaluate the time complexity of the AccSS constructed in the Section 3. Insertion of C_1 takes one step, each coloring of a node takes 3 steps, hence the first phase takes a total number of $3n + 1$ steps. Note that each splicing step may be formed by a number of individual splicings that are done in parallel. Again we make use of the assumption that the each string appears in a sufficient number of copies such that if

a splicing rule is applicable to a certain string, it can effectively be used.

The second phase requires, in the worst case, that the prefix $\langle x_1x_2 \dots x_n \rangle$ to be scanned by every symbol e_i , $1 \leq i \leq m$. Each such symbol can scan the prefix in $3n$ splicing steps. As there are m such symbols, the total number of steps is $3nm$. Furthermore, the number of steps necessary to delete the symbols x_i as required by the condition (viii) is n . Finally, the number of steps to reduce $\langle Q^s \rangle$ to $\langle Q \rangle$ is at most m .

We now conclude that the time complexity of our algorithm is $O(nm)$

3.1.4 Evaluating Descriptive Complexity

We now evaluate the descriptive complexity measures defined in Section 2. In the first phase we have the followings:

- The set of axioms used in the first phase is

$$A_1 = \{C_1\#, X_nb_n\#, X_ng_n\#, X_nr_n\#\} \cup \{X_iY_i \mid 1 \leq i \leq n\} \cup \{X_i' \dagger C_{i+1}\# \mid \dagger \in \{b_i, g_i, r_i\}, 1 \leq i \leq n-1\} \cup \{\#Y_i' \mid 1 \leq i \leq n\}.$$

Therefore, the number of axioms is $5n + 1$.

- We use the following set of symbols:

$$S_1 = \{\langle, \rangle, \#\} \cup \{C_i, X_i, Y_i, X_i', Y_i' \mid 1 \leq i \leq n\} \cup \{b_i, g_i, r_i \mid 1 \leq i \leq n\}.$$

Therefore, the number of new symbols is $8n + 3$.

- The number of splicing rules in each subgroup is given in Table 1. Consequently, there are $16n - 5$

Table 1: The number of splicing rules.

Subgroup	Number of splicing rules
1	1
2.1	$3n$
2.2	$3(n-1)$
2.3	3
2.4	n
3.1	3
3.2	$9(n-2)$
3.3	9

splicing rules.

- The maximal length of a splicing rule is 8.

For the second phase an exact evaluation of all measures is rather cumbersome, but we can evaluate the order of these measures.

- The number of axioms used is at most $14n$.
- The number of symbols used is $6nm + 4m + 2$.
- The number of splicing rules is at most $5mn(n+1) + 12mn + n + m$
- The maximal length of a splicing rule is 8.

Finally, $Final(\Gamma) = 2$.

We conclude that

$$\begin{aligned} Ax(\Gamma) &= O(n), \\ Symb(\Gamma) &= O(mn), \\ NSplice(\Gamma) &= O(mn^2), \\ LSplice(\Gamma) &= 8, \\ Final(\Gamma) &= 2. \end{aligned}$$

3.2 Two Variants

It is worth noting that the algorithm presented above can be accelerated to $O(n)$ time at an exponential cost of rules and axioms ($O(5^m n)$), and a linear cost of the length of splicing rules. It can also be accelerated to $O(n)$ time at the same cost of rules, axioms and number of symbols but a constant length of the splicing rules. We briefly discuss these variants in the sequel.

The second phase can be changed as follows. Instead of scanning the prefix from right to left with each symbol e_i , we can do this scanning process with the whole segment $e_1e_2 \dots e_m$. The splicing rules for doing this are:

$$[(\clubsuit, x_i Z_1 Z_2 \dots Z_m); (\#, Y_1 Y_2 \dots Y_m x_i)],$$

$$\text{where } \clubsuit = \begin{cases} <, & \text{if } i = 1 \\ x_{i-1}, & \text{if } i > 1, \end{cases}$$

$\#Y_1 Y_2 \dots Y_m x_i$ is an axiom,

$Z_j \in \{Q, e_j, \bar{e}_j, \hat{e}_j, \tilde{e}_j\}$, and

- if $i \notin \{j_1, j_2\}$, then $Y_j = Z_j$,

- if $i \in \{j_1, j_2\}$, and $Z_j = e_j$, then

$$Y_j = \begin{cases} \bar{Z}_j, & \text{if } x = b, \\ \hat{Z}_j, & \text{if } x = g, \\ \tilde{Z}_j, & \text{if } x = r, \end{cases}$$

- if $i \in \{j_1, j_2\}$, then $Y_j = \lambda$ provided that

$x \in \{g, r\}$ and $Z_j = \bar{e}_j$,

$x \in \{b, r\}$ and $Z_j = \hat{e}_j$,

$x \in \{b, g\}$ and $Z_j = \tilde{e}_j$,

- $Y_j = Q$, in all the other cases,

$1 \leq j \leq m, 1 \leq i \leq n$.

As one can easily see, at every splicing step the segment $Z_1 Z_2 \dots Z_m$ is updated with respect to the value of x_i and moves to the left. Therefore, after n splicing steps either a segment consisting of some occurrences of Q reaches the left endmarker, or the string is $\langle \rangle$.

As far as the descriptive complexity of this variant, we note that the number of splicing rules as well

as that of axioms is $O(5^m n)$. Furthermore, the maximal length of a splicing rule is $2m + 4$.

Another variant could consider symbols for encoding the segments, hence the number of symbols explodes exponentially but the maximal length of a splicing rule remains constant.

As a consequence of the constructions and complexity evaluations above, we can conclude

Theorem 1. *Let G be a graph with n vertices and m edges, without loops.*

1. *There is an AccSS-algorithm that decides the 3-colorability for G in $O(nm)$ time and $O(mn^2)$ other resources (axioms, symbols, splicing rules).*
2. *There is an AccSS-algorithm that decides the 3-colorability for G in $O(n)$ time, $O(5^m n)$ axioms and splicing rules, and $O(m)$ maximal length of the splicing rules.*
3. *There is an AccSS-algorithm that decides the 3-colorability for G in $O(n)$ time, $O(5^m n)$ axioms, symbols, and splicing rules, and a constant maximal length of splicing rules.*

We also want to stress that the technique used in the construction of the above AccSS-algorithm can be employed for constructing similar algorithms for other intractable problems.

4 CONCLUSIONS AND FURTHER WORK

A new variant of accepting splicing system that can be used as a problem solver was introduced. This model may be used as a problem solver because the condition for halting the computation on a given input is accompanied by a condition for making a decision as soon as the computation has stopped. An algorithm based on this accepting splicing system that solves a well-known NP-complete problem, namely the 3-colorability problem is presented. The time efficiency of this solution is analyzed together with some descriptive complexity measures for axioms, supplementary symbols, number of splicing rules. More precisely, for a given graph with n vertices and m edges, our solution runs in $O(nm)$ time, and needs $O(mn^2)$ other resources. Two variants of this algorithm of a reduced time complexity at an exponential increase of the other resources are also discussed.

We discuss here a few possible ways, that appear attractive to us, of continuing the work started here. From a theoretical point of view, a complete investigation of the computational power of this model seems to be of interest. This investigation could contribute to a more complete picture of the study started

in (Mitrana et al. 2010). From a more practical point of view, would like to consider possible software implementations of this model that could be similar to those reported in the literature for other bio-inspired models. Along these lines, software simulators for this model using massively parallel platforms for multi-core desktop computers, clusters of computers and cloud resources similarly to the approached proposed in (Gómez-Canaval et al. 2015) and (Gómez-Canaval et al. 2015) might lead to relevant results. In (Gómez-Canaval et al. 2015), it is shown that massively distributed platforms for big data scenarios makes them potential candidates for the development of ultra-scalable simulators able to simulate different variants of bio-inspired models. In particular, computing platforms by developing an engine that uses *Apache Giraph* on top of the *Hadoop* platform might be useful. The results of some experiments with such simulators suggest that they might be amenable to minimize the growth of processing data, hence to be adapted for models like that discussed here.

ACKNOWLEDGEMENTS

Work supported by a grant of the Romanian National Authority for Scientific Research and Innovation, project number POC P-37-257.

REFERENCES

- Arroyo, F., Castellanos, J., Dassow, J., Mitrana, V., and Sánchez-Couso, J.R. (2013). Accepting splicing systems with permitting and forbidding words. *Acta Inf.* 50 pp.1–14.
- Bonizzoni, P. and Mauri, G. (2005). Regular splicing languages and subclasses. *Theoret. Comput. Sci.* 340 pp. 349–363.
- Bordihn, H., Mitrana, V., Păun, A., Păun, M. (2017). Networks of polarized splicing processors. In *Theory and Practice of Natural Computing, TPNC 2017*, LNCS 10687, Springer, Berlin, Heidelberg, pp. 165–177.
- Bordihn, H., Mitrana, V., Negru, M.C., Păun, A., Păun, M. (2018). Small networks of polarized splicing processors are universal. *Natural Computing*, 17 pp. 799–809.
- Csuhaj-Varjú, E., Kari, L. and Păun, Gh. (1996). Test tube distributed systems based on splicing. *Computers and AI* 15 pp. 211–232.
- Culik, K. and Harju, T. (1991). Splicing semigroups of dominoes and DNA. *Discrete Appl. Math.* 31 pp. 261–277.
- Denninghoff, K.L. and Gatterdam, R.W. (1989). On the undecidability of splicing systems. *Intern. J. Computer Math.* 27 pp. 133–145.

- De Felice, C., De Prisco, R., Malandrino, D., Zaccagnino, G., Zaccagnino, R., and Zizza, R. (2017). Splicing music composition. *Inf. Sci.* 385 pp. 196–212.
- Freund, R., Kari, L., and Păun, Gh. (1999). DNA computing based on splicing. The existence of universal computers. *Theory of Computing Syst.* 32 pp. 69–112.
- Garey, M., and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co. New York.
- Gómez-Canaval, S., Mitrana, V., Sánchez-Couso, J.S. (2016). Networks of splicing processors with evaluation sets as optimization problems solvers. *Information Sciences* 369, 457–466.
- Gómez-Canaval, S., Ortega, A., Orgaz, P. (2015). Distributed simulation of NEPs based on-demand cloud elastic computation. In *Advances in Computational Intelligence*, LNCS 9094, Springer, Berlin, Heidelberg, pp. 40–54.
- Gómez-Canaval, S., Ordozgoiti, B., Mozo, A. (2015). NPEPE: Massive natural computing engine for optimally solving NP-complete problems in Big Data scenarios. In *Communications in Computer and Information Science* 539, Springer, Berlin, Heidelberg, pp. 207–217.
- Head, T. (1987). Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours. *Bull. Math. Biology* 49 pp. 737–759.
- Head, T., Păun, Gh., and Pixton, D. (1997). Language theory and molecular genetics. Chapter 7, vol. 2, in (Rozenberg and Salomaa, 1997).
- Loos, R., Malcher, A., and Wotschke, D. (2008). Descriptive complexity of splicing systems. *Intern. J. Found. Comp. Sci.* 19 pp. 813–826.
- Loos, R. and Ogihara, M. (2007). Complexity theory for splicing systems. *Theor. Comput. Sci.* 386 pp. 132–150.
- Loos, R., Martín-Vide, C., and Mitrana, V. (2006). Solving SAT and HPP with accepting splicing systems. In *Proc. 9th Parallel Problem Solving from Nature (PPSN IX)*, LNCS 4193, Springer-Verlag, Berlin, pp. 771–777.
- Loos, R., Manea, F., Mitrana, V. (2009). On small, reduced, and fast universal accepting networks of splicing processors. *Theoretical Computer Science* 410, pp. 406–416.
- Manea, F., Martín-Vide, C., Mitrana, V. (2006). All NP-problems can be solved in polynomial time by accepting networks of splicing processors of constant size. In: *DNA Computing*. LNCS, vol. 4287, Springer, Berlin, Heidelberg, pp. 47–57.
- Manea, F., Martín-Vide, C., Mitrana, V. (2007). Accepting networks of splicing processors: Complexity results. *Theoretical Computer Science* 371, pp. 72–82.
- Mitrana, V., Petre, I., and Rogojin, V. (2010) Accepting splicing systems. *Theor. Comput. Sci.* 411 pp. 2414–2422.
- Păun, Gh. (1996). Regular extended H systems are computationally universal. *J. Automata, Languages, Combinatorics*, 1 pp. 27–36.
- Păun, G. (1998). Distributed architectures in DNA computing based on splicing: Limiting the size of components. In *Unconventional Models of Computation*, Springer, Berlin, Heidelberg, pp 323–335.
- Păun, Gh., Rozenberg, G., and Salomaa, A. (1996). Computing by splicing. *Theoret. Comput. Sci.* 168 pp. 321–336.
- Paun, Gh., Rozenberg, G., and Salomaa, A. (1997). Computing by splicing. Programmed and evolving splicing systems. *IEEE Intern. Conf. on Evolutionary Computing*, Indianapolis, pp. 273–277.
- Paun, Gh., Rozenberg, G., and Salomaa, A. (1998). *DNA Computing - New Computing Paradigms*, Springer-Verlag, Berlin.
- Pixton, D. (1996). Regularity of Splicing Languages. *Discrete Appl. Math.*, 69 pp. 101–124.
- Pixton, D. (1995). Linear and circular splicing systems. In *First International Symposium on Intelligence in Neural and Biological Systems. INBS'95*, IEEE Herndon, VA, USA, pp. 181–188.
- De Prisco, R., Malandrino, D., Zaccagnino, G., Zaccagnino, R., and Zizza, R. (2017). Splicing-inspired recognition and composition of musical collectives styles. In *Theory and Practice of Natural Computing - 6th International Conference, TPNC 2017*, LNCS 10687, Springer-Verlag, Berlin, pp. 219–231.
- Rabinow, P. (1996). *Making PCR: A Story of Biotechnology*. University of Chicago Press.
- Rozenberg, G. and Salomaa, A. (eds.) (1997). *Handbook of Formal Languages*, vol. I-III, Springer-Verlag, Berlin.
- Siromoney, R., Suhrmanian, K.G., and Rajkumar Dare, V. (1992). Circular DNA and splicing systems. In *International Conference on Parallel Image Analysis, ICPIA 1992*, LNCS 654, Springer-Verlag, Berlin, pp. 260–273.