

Planarity of Knots, Register Automata and LogSpace Computability

Alexei Lisitsa, Igor Potapov, and Rafiq Saleh

Department of Computer Science, University of Liverpool, Ashton Building,
Ashton St, Liverpool L69 3BX, U.K.
{Lisitsa,Potapov,R.A.Saleh}@liverpool.ac.uk

Abstract. In this paper we investigate the complexity of planarity of knot diagrams encoded by Gauss words, both in terms of recognition by automata over infinite alphabets and in terms of classical logarithmic space complexity. As the main result, we show that recognition of planarity of unsigned Gauss words can be done in deterministic logarithmic space and by deterministic register automata. We also demonstrate generic results on the mutual simulations between logspace bounded classical computations (over finite alphabets) and register automata working over infinite alphabets.

1 Introduction

Knot theory is the area of mathematics that studies mathematical knots and links. A knot (a link) is an embedding of a circle (several circles) in 3-dimensional Euclidean space, \mathbb{R}^3 , considered up to a smooth deformation of the ambient space. It is a well established and active area of research with strong connections to topology, algebra and combinatorics. Some major problems in the main focus of knot theory have algorithmic or computational nature: *equivalence problem* (how to recognise that two knots are equivalent), or *unknottedness problem* (how to recognise that a knot is a trivial one). Consideration of such problems led to fruitful interactions between knot theory and computer science. In particular, the questions of computational complexity of knot problems have been addressed in [9]. Examples of other interactions include works on formal language theory [11] and quantum computing [1, 7, 16].

One of the earliest questions of algorithmic nature related to knots was the question of characterisation of Gauss words [8]. With every knot one can associate a word, called a Gauss word, which is a sequence of labels for the crossings read off directly from the projection of the knot on a plane. Depending on whether the information on the orientation is present the word can be signed or unsigned. The simple property of any Gauss word is that every label (index) in it appears twice. The converse is not true, there are the words with every symbol appearing twice which do not correspond to any classical planar knot diagram. The question of characterisation of “true”, or planar Gauss words was posed by Gauss himself [8] and was eventually resolved by Nagy in [20]. Since then there has been proposed many criteria and algorithms both for recognition of signed [3, 13] and unsigned [17, 22, 24, 25, 12, 4–6, 24, 18, 26] Gauss words. The questions of computational

complexity of the proposed algorithms were rarely explicitly addressed with notable exceptions being [13] where linear time algorithm for the signed case is proposed, and in [25] where a linear time complexity for unsigned case is established and compared with earlier quadratic bounds in [22].

In [15] we proposed to evaluate the complexity of problems of recognising knot properties in terms of the computational power of devices needed to recognise the properties. Following the proposal we demonstrated lower and upper bounds for recognisability of knot properties in terms of various automata models over *infinite* alphabets (see Section 2.2 below). The infinite alphabet appeared naturally due to the fact that the number of crossings in knots is unbounded. The main property addressed was *planarity* of signed Gauss words [13] (see Section 3). We have shown that planarity of signed Gauss words can be recognised by deterministic register automata working over infinite alphabets. It follows that signed planarity problem is in \mathcal{L} (deterministic logspace).

In this paper we continue this line of research focussing mainly on the *unsigned* case. Our contribution is as follows:

- 1) We provide an analysis of Cairns-Elton algorithm for unsigned planarity and show that it is implementable by co-non-deterministic register automata.
- 2) We demonstrate generic results on the mutual simulations between logspace bounded classical computations (over finite alphabets) and register automata working over infinite alphabets. New characterisation of languages recognisable by register automata is more general than one proposed in [21].
- 3) We further show that Cairns-Elton algorithm is implementable in \mathcal{SL} (symmetric logspace) and therefore in \mathcal{L} . It follows that planarity of unsigned Gauss words is recognisable by deterministic register automata, refuting the conjecture from [15].

2 Preliminaries

We use standard notations from complexity theory and assume that the reader is familiar with the complexity classes \mathcal{L} (deterministic logspace), \mathcal{SL} (symmetric logspace), \mathcal{NL} (nondeterministic logspace), $\mathbf{co-NL}$ (co-non-deterministic logspace) and $SPACE(f(n))$ (space is bounded by a function $f(n)$).

2.1 Automata Over an Infinite Alphabet

Register automata are finite state machines equipped with a finite number of memory cells called registers which may hold values from an infinite alphabet. It is one of the weakest models of automata over infinite alphabets. It was initially introduced in [10] and then studied further in [21]. The model we consider is computationally equivalent to the original version in [21], which is modified by adding two extra rules (see Definition 1) [15].

Let D be an infinite set called an *alphabet*. A word, or a string over D , or shortly, D -word or D -string is a finite sequence d_1, \dots, d_n where $d_i \in D$, $i = 1, \dots, n$. A language over D (D -language) is a set of D -words. For a word w and a symbol d denote by $|w|_d$ the number of occurrences of d in w . As usual $|w|$ denotes the length of the word w .

Definition 1. [15, 21] *A non-deterministic two-way k -register automaton over an infinite alphabet D is a tuple (Q, q_0, F, τ_0, P) where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, $\tau_0 : \{1, \dots, k\} \rightarrow D \cup \{\triangleright, \triangleleft\}$ is the initial register assignment and P is a finite set of transitions :*

- 1) $(i, q) \rightarrow (q', d)$ (If a current state is q and the observed symbol on the tape equals to a value in the register i then enter the state q' and move along the string according to the specified direction d where $i \in \{1, \dots, k\}$, $q, q' \in Q$ and $d \in \{stay, left, right\}$)
- 2) $q \rightarrow (q', i, d)$ (If a current state is q and the observed symbol on the tape does not equal to any value held in registers then enter the state q' , copy the current symbol to a specified register i and move along the string according to the specified direction d , where $i \in \{1, \dots, k\}$, $q, q' \in Q$ and $d \in \{stay, left, right\}$).
- 3) $(i, q) \rightarrow (q', j, d)$ (If a current state is q and the observed symbol equals to a value in the register i then enter the state q' , copy the current symbol to a register j and move along the string according to the specified direction d where $i, j \in \{1, \dots, k\}$, $q, q' \in Q$ and $d \in \{stay, left, right\}$).
- 4) $q \rightarrow (q', d)$ (If a current state is q and the observed symbol does not equal to any value held in registers then enter the state q' and move along the string according to the specified direction d , where $q, q' \in Q$ and $d \in \{stay, left, right\}$).

Deterministic and co-non-deterministic register automata as well as the language accepted by automata are defined in the standard way. We denote by DRA, NRA, coNRA to be the classes of languages recognisable by deterministic, non-deterministic and co-non-deterministic two-way register automata, respectively.

Words and Data Words In previous works on the computational models on infinite alphabets it has been acknowledged that in many situations it is natural to consider infinite alphabets as the subsets of $\Sigma \times \Delta$ where Σ is a *finite* set and Δ is an infinite set. Thus, the symbol here is an ordered pair (a, b) . The words over such alphabets are called *data words* [2]. In particular Gauss words are introduced in Section 2.2 as natural instances of data words. In the definition of automata over data words, it is sensible to assume that when an automaton reads a symbol (a, b) it has a direct access to both components of the pair. For this purpose, the form of transition rules can be adapted to include one extra argument on the left-hand sides.

2.2 Knots

A knot is defined as a closed, non-self-intersecting curve that is embedded in \mathbb{R}^3 . Knots can be described in many ways, including various discrete representations. A common method of representing a knot is a knot diagram that is a projection of the knot to a plane in a general position involving only double crossings.

At each crossing we indicate which branch is “over” and which is “under”, which allows us to recreate the original knot. For oriented knots each crossing has a well-defined sign (+ or -). To determine the sign of each crossing, we label the crossing with a + if we can rotate its under-strand clockwise to make the

arrows line up and a $-$ if the rotation of the under-strand is counter-clockwise (see the picture in left-hand side of Figure 1).

A knot diagram can be encoded by a string of symbols O_i 's (over) and U_i 's (under) known as a *Gauss word*. The procedure of writing a Gauss word can be described as follows. Starting from a base point on a knot diagram, write down the labels of the crossings and their types of strand ordered according to the orientation of the knot. The oriented trefoil in Figure 1 is encoded by the *signed* Gauss word $O_1^+U_2^+O_3^+U_1^+O_2^+U_3^+$. Removing signs leads to the *unsigned* Gauss word. Indices of crossings can be arbitrarily permuted, so one knot diagram can be encoded by many Gauss words. A *shadow* Gauss word can be obtained from an unsigned Gauss word by removing the U s and O s from each label.

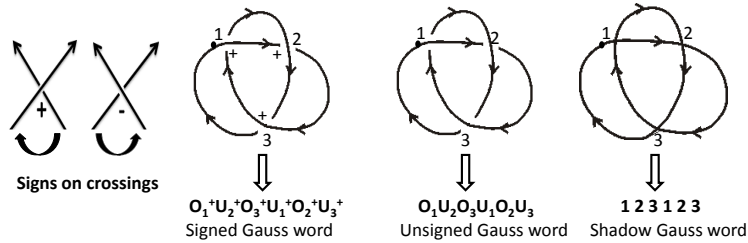


Fig. 1. Discrete representations of a knot

Definition 2. An *unsigned Gauss word* w is a data word over the alphabet $\Sigma \times \mathbb{N}$ where $\Sigma = \{U, O\}$, such that for every $n \in \mathbb{N}$ either

- $|w|_{(U,n)} = |w|_{(O,n)} = 0$, or
- $|w|_{(U,n)} = |w|_{(O,n)} = 1$.

Definition 3. A *signed Gauss word* w is a data word over the alphabet $\Sigma \times \mathbb{N}$ where $\Sigma = \{U^+, O^+, U^-, O^-\}$, such that for every n either

- $|w|_{(U^+,n)} = |w|_{(O^+,n)} = |w|_{(U^-,n)} = |w|_{(O^-,n)} = 0$, or
- $|w|_{(U^+,n)} = |w|_{(O^+,n)} = 1$ and $|w|_{(U^-,n)} = |w|_{(O^-,n)} = 0$, or
- $|w|_{(U^-,n)} = |w|_{(O^-,n)} = 1$ and $|w|_{(U^+,n)} = |w|_{(O^+,n)} = 0$.

Definition 4. A *shadow projection* of a (signed/unsigned) Gauss word $w = (A_1, i_1), \dots, (A_n, i_n)$ is a shadow Gauss word $sp(w) = i_1, \dots, i_n$ where $A_i \in \Sigma$ and $i_j \in \mathbb{N}$

Definition 5. A *signing* s is a mapping $s : \mathbb{N} \rightarrow \{+, -\}$.

2.3 Planarity of Knot Diagrams Represented by Gauss Words

Every knot diagram can be represented by a Gauss word but not every Gauss word represents a knot diagram. The fact that every knot can be represented by a Gauss word directly follows from the constructive definition of a Gauss word

and the second fact that not every Gauss word represents a classical knot in \mathbb{R}^3 is illustrated in Figure 2. For example, any attempt to reconstruct a knot diagram from the Gauss word $O_1O_2U_1O_3U_2U_3$ will lead to new (virtual) crossings which are not present in the Gauss word (virtual crossings are marked on the diagram in Figure 2 with a small circle around the crossing). Such an observation was one of the motivations for introducing virtual knot theory [12]. A Gauss word which represents a classical knot diagram, that is a diagram embeddable into a plane without virtual crossings, is called classical or planar. The question about

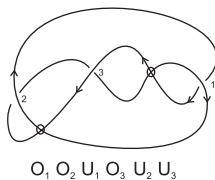


Fig. 2. Non-planar knot diagram

planarity of knot diagrams encoded by Gauss words is formulated separately for signed and unsigned cases. For Signed Gauss words the planarity question can be described as given a signed Gauss word w , does w represent a planar knot diagram? We will refer to this problem as SIGNED PLANARITY. For unsigned Gauss words, the question is stated in the following way: Given an unsigned Gauss word w , does there exist a choice of signings that can be assigned to w such that w represents a planar knot diagram? This problem will be referred to as UNSIGNED PLANARITY.

In [15], we showed that an algorithm for the decision of planarity of unsigned Gauss words proposed by Kauffman in [12] can be implemented by Linear Bounded Memory Automata over an infinite alphabet and conjectured that UNSIGNED PLANARITY is not recognisable by non-deterministic register automata. Here, we refute this conjecture and show that UNSIGNED PLANARITY can be recognised by deterministic register automata.

3 Main Results

In this section we present an upper bound for the planarity problem of unsigned Gauss words by first showing that it is recognisable by a co-non-deterministic register automata. Then we refine this result to show that the only part that requires non-determinism can be reduced to the search of cycles with simply checkable properties. We further show that the search is implementable in deterministic logspace. Finally, we show that deterministic logspace computations can be modelled by a deterministic register automata.

3.1 Cairns-Elton Algorithm

Cairns and Elton presented an algorithm in [4] which deals with UNSIGNED PLANARITY. We will first begin with definitions that will be used to describe the main steps of the algorithm.

Definition 6. For a Gauss word w , denote by $\alpha_i(w)$ the number of symbols that occur in w in cyclic order between the symbols U_i and O_i , taken modulo 2.

Definition 7. For an unsigned Gauss word w and a signing s , a signed word w^s is obtained from w by replacing all symbols (U, i) with $(U^{s(i)}, i)$ and (O, i) with $(O^{s(i)}, i)$. Let S_i denote the subset of symbols that occur in w^s in cyclic order between, either the symbols U_i^+ and O_i^+ , or O_i^- and U_i^- . Let \bar{S}_i denote $\{U_i^{s(i)}, O_i^{s(i)}\} \cup S_i$ and S_i^{-1} denote the set S_i after swapping U s with O s, that is $S_i^{-1} = \{(U^{s(j)}, j) | (O^{s(j)}, j) \in S_i\} \cup \{(O^{s(j)}, j) | (U^{s(j)}, j) \in S_i\}$. Then $\beta_{ij}(w^s)$ is the number of elements in the intersection of \bar{S}_i and S_j^{-1} taken modulo 2 (i.e. $\beta_{ij}(w^s) = |\bar{S}_i \cap S_j^{-1}| \pmod{2}$).

Notice that $\beta_{ij}(w^s)$ depends on signs $s(i)$ and $s(j)$ but not on $s(k)$ for $k \neq i, j$.

Definition 8. Given an unsigned Gauss word w , the vertices of the interlacement graph $G(w)$ are labels in a shadow projection $sp(w)$ (natural numbers) and the edges of $G(w)$ are the pairs of labels (i, j) such that i and j are interlaced in $sp(w)$ (i occurs once between two occurrences of j and vice versa).

Example 1. Given $w = U_1O_3U_4U_2O_1U_5O_2U_3O_5O_4$, the interlacement graph $G(w)$ of w is shown in figure 4. Let $i = 1, j = 2, s(1) = +$ and $s(2) = +$. Then $\alpha_1(w) = |\{O_3, U_4, U_2\}| = 3 \equiv 1 \pmod{2}$, and $\beta_{12}(w^s) = |\bar{S}_1 \cap S_2^{-1}| = |\{U_1^{s(1)}, O_3^{s(3)}, U_4^{s(4)}, U_2^{s(2)}, O_1^{s(1)}\} \cap \{U_1^{s(1)}, O_5^{s(5)}\}| = |\{U_1^{s(1)}\}| \equiv 1 \pmod{2}$

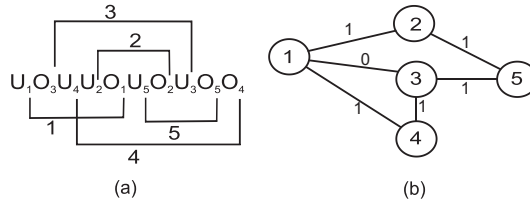


Fig. 4. Non-planar Gauss word w and its corresponding interlacement graph $G(w)$ with edges (i, j) labelled by $\beta_{ij}(w^s)$

For a signed word w^s and for each edge e_{ij} in $G(w)$, we assign the number $\beta_{ij}(w^s) \in \mathbb{Z}_2$. According to [4] that assignment defines a \mathbb{Z}_2 1-cochain $B(w^s)$ and the property that the Cairns-Elton algorithm checks is whether this co-chain is closed. For the purpose of this paper we need only characterisation of the closedness of $B(w^s)$ in terms of notions we have already introduced: $B(w^s)$ is closed if and only if for every closed path P in $G(w)$, the sum of the numbers $\beta_{ij}(w^s) \equiv 0 \pmod{2}$ for each edge $(ij) \in P$. The closed path is in fact a simple cycle with no repeated vertices other than starting and ending vertices.

The Propositions 1 and 2 provide with the properties crucial for the efficient implementation of the Cairns-Elton algorithm. Also as an easy consequence of Proposition 1, we formulate Lemma 1.

Proposition 1. [4, page 139] $\beta_{ij}(w^s)$ does not depend on s whenever i and j do not interlace.

Lemma 1. *Let $G(w)$ denote the interlacement graph of the Gauss word w and $N(v_i)$ denote the set of vertices connected to v_i , then $\beta_{ij}(w) = |N(v_i) \cap N(v_j)| \pmod{2}$ whenever i and j do not interlace in w .*

Proposition 2. [4, Lemma 1] *The condition $B(w^s)$ to be closed depends on w but not on s .*

For all positive signing s , that is $s(i) = +$ for all $i \in \mathbb{N}$, we denote $B(w^s)$ by $B(w)$ and $\beta_{ij}(w^s)$ by $\beta_{ij}(w)$.

Cairns-Elton Algorithm: Given an unsigned Gauss word w , the algorithm proceeds by checking that

1. For all $i \in w$, $\alpha_i(w) = 0$.
2. For all $i, j \in w$, $\beta_{i,j}(w) = 0$ whenever i and j do not interlace.
3. $B(w)$ is closed.

If conditions 1,2 and 3 are satisfied then the algorithm returns “the word w is planar”, otherwise if any of the conditions is not satisfied, the algorithm returns “the word w is non-planar”.

3.2 Implementation of Cairns-Elton Algorithm

We will show in this subsection that the checking of first two conditions of the above algorithm is implementable by a deterministic register automata whereas the checking of the third condition is implementable by a co-non-deterministic register automata.

First, we refine the above description of the algorithm and present it in more details. Given an unsigned Gauss word w , the algorithm proceeds in three stages.

1. The input word is checked on whether the number of neighbours of v_i is odd for some v_i in $G(w)$. If “yes”, the algorithm stops with the result “the input word is non-planar”, otherwise the algorithm proceeds to the second stage.
2. The input word is checked on whether $\beta_{ij}(w)$ is odd for some pair of vertices (v_i, v_j) in $G(w)$ that are not connected by an edge. If “yes”, the algorithm stops with the result “the input word is non-planar”, otherwise the algorithm proceeds to the third stage.
3. For all positive signing s , the input word is checked on whether there exists a cycle in $G(w)$ such that the sum of $\beta_{ij}(w^s)$ assigned to its edges e_{ij} is odd. If “yes”, the algorithm stops with the result “the input word is non-planar”, otherwise the algorithm stops with the result “the input word is planar”.

Theorem 1. *The language of non-planar unsigned Gauss words (UNSIGNED NONPLANARITY) can be recognised by a two-way non-deterministic register automaton.*

Proof. The proof is divided into two parts. In the first part we show that the first two conditions can be implemented by a deterministic register automata and in the second part we show the third condition can be implemented by a non-deterministic register automata. Let w be an unsigned Gauss word and $G(w)$ be the interlacement graph of w . Denote by $N(v_i)$ the set of all neighbours of a vertex $v_i \in G(w)$.

Part1 For the first condition, the automaton will check the number $|N(v_i)|$ of neighbours of each vertex $v_i \in G(w)$. It will store in the register the first occurrence of the label i in w which corresponds to vertex v_i and store the parity of $|N(v_i)|$ in finite state control of the automaton. To check the parity of $|N(v_i)|$, the automaton goes through each symbol j in between the pair of the labels i and i^{-1} in w (where i^{-1} represents the second occurrence of i in w) and on the first occurrence of j it moves first to an odd state and then alternates between odd and even states for any further occurrences. If i^{-1} is reached and the current state is odd then it moves to an accepting state. Otherwise if, for all vertices $v_i \in G(w)$, the parity of $|N(v_i)|$ is even then it checks condition (2).

For the second condition, we use Lemma 1 where the automaton is required to check the number of common neighbours ($|N(v_i) \cap N(v_j)|$) for any pair of vertices (v_i, v_j) that are not connected by an edge in $G(w)$. To verify that a vertex v_i is not connected to any vertex v_j , the automaton stores in the registers the first occurrence of i and the first occurrence of each j and then it checks whether there is an even number (either 2 or 0) of occurrences of each j in between i and i^{-1} . If the number of occurrences of j in between i and i^{-1} is even then it stores the symbol k in the register (which occurs in between i and i^{-1}) and compares it with the symbols in between j and j^{-1} . The parity of $|N(v_i) \cap N(v_j)|$ is stored in finite state control of the automaton. If there is a match, it will move first to an odd state and then alternate between odd and even states for any further matches. If i^{-1} is reached and current state is odd, the automaton moves to an accepting state. Otherwise if, for all pairs of vertices $v_i, v_j \in G(w)$ that are not connected by an edge, the parity of $|N(v_i) \cap N(v_j)|$ is even then it checks condition 3.

Part2 For checking Condition 3, we assume that s is all positive signing. First, we show how to check if two vertices v_i and v_j are connected by an edge and how to compute their $\beta_{ij}(w^s)$ value. Then we show how to sum up the $\beta_{ij}(w^s)$ values online during the traversal of a cycle. To verify that two vertices v_i and v_j are connected by an edge of $G(w)$, the automaton keeps a copy of i and j in the registers and checks that there is only one occurrence of j in between U_i and O_i . Now to calculate the value of $\beta_{ij}(w^s)$ for all positive signing s , the automaton moves its head to find the symbol U_j then compares the counterpart of each symbol k in between U_j and O_j (notice that all such counterparts form the set S_j^{-1}) with the symbols in the set $\tilde{S}_i(U_i, \dots, O_i)$. If there is a match it will move first to an odd state and then alternate between odd and even states for any further matches until O_j is reached. Finally to traverse a cycle in $G(w)$, the automaton non-deterministically chooses a vertex v_i and moves along chosen edge. During the traversal, it sums up the $\beta_{ij}(w^s)$ values of each visited edge by incrementing the counter by $1 \pmod{2}$ only if the value of $\beta_{ij}(w^s)$ is odd, and continue updating the counter until the same vertex v_i is met for the second time. If v_i is met for the second time and the value of the counter is odd then the automaton moves to an accepting state. \square

Corollary 1. *UNSIGNED PLANARITY can be recognised by two-way co-non-deterministic register automata.*

3.3 RA to LOGSPACE

We will show that if a language L over the infinite alphabet D is acceptable by two-way register automata then the encoding of L over a finite alphabet can be accepted by a Turing machine in log space memory. Let us first define the encoding of L over a finite alphabet as follows.

Let L be a language over the infinite alphabet D . For any symbol y of a word $w \in L$ we denote by $ord_w(y)$ (or $ord(y)$ if w is understood) the number of distinct symbols in w to the left of the first occurrence of y in w . So, for example, if $w = aacbca$ then $ord_w(a) = 0$, $ord_w(c) = 1$ and $ord_w(b) = 2$.

If ϕ is a mapping from natural numbers into their binary encoding, $\phi : \mathbb{N} \rightarrow \{0, 1\}^*$, then for any word $w = w_1, \dots, w_k \in L$ where $w_i \in D$ the mapping $\psi(w) : D^* \rightarrow \{0, 1, \#\}^*$ is an encoding of a word w , where each binary encoding of w_i is separated by a special symbol $\#$:

$$\psi(w) = \#\phi(ord(w_1))\#\phi(ord(w_2))\#, \dots, \#\phi(ord(w_k)).$$

Thus a language $L_{finite} = \{\psi(w) | w \in L\}$ is an encoding of L over the finite alphabet $\{0, 1, \#\}$ and L_{binary} is a natural encoding of L_{finite} over $\{0, 1\}$ alphabet, where $0 \rightarrow 00$, $1 \rightarrow 01$ and $\# \rightarrow 11$.

Proposition 3. *If L is recognisable by a finite register automata then L_{finite} is recognisable by a Turing machine in log space memory.*

Proof. Let L denote the language accepted by a finite register automaton A with r registers. Let us show that the language L_{finite} is recognisable by a Turing machine M that uses at most $O(\log n)$ space. Let $w = w_1, \dots, w_n \in L$ and $|w| = n$, then w consists of no more than n different symbols, so the length of the binary encoding of w_i is no more than $\log n$, i.e. $|\phi(ord(w_i))| \leq \log n$. So M can mimic all the computations of A by keeping the value of the registers of A on its work tape. The finite state control in A will correspond to the finite state control in M (including non-determinism) and the content of r registers in A will be stored on the work tape in M , which require $r \log n$ cells and r is a constant. The only two operations of register automata are: to store a symbol in a register and to compare the register value with a symbol on a tape will not require any extra space apart from $r \log n$ cells. \square

Corollary 2. *UNSIGNED PLANARITY is in $\mathbf{co-NL}$ and therefore is in \mathcal{NL}*

Next we refine the complexity bounds of planarity recognition by 1) demonstrating general result on simulation of Logspace bounded computations on register automata and 2) showing that UNSIGNED PLANARITY is in \mathcal{L} .

3.4 UNSIGNED PLANARITY in \mathcal{L}

We have shown in subsection 3.2 that planarity of Gauss words is recognisable by co-nondeterministic register automata and therefore belongs to the complexity class $\mathbf{co-NL}$ ($= \mathcal{NL}$). Using simple arguments one can show that in terms of classical complexity classes the result can be refined further, that is UNSIGNED

PLANARITY belongs to \mathcal{L} (deterministic logspace). Indeed, the only place when one needs nondeterminism in the proof of Theorem 1 is in PART 2 where the search for the cycles in the interlacement graph bearing odd sum of labels. It is routine to check that the rest of the algorithm can be easily implemented in \mathcal{L} . It follows then that UNSIGNED NONPLANARITY is logspace reducible to the problem from the following proposition and therefore is in \mathcal{L} .

Proposition 4. *The following problem is in \mathcal{L} (deterministic logspace)*

GIVEN: *An undirected graph G with every edge labelled by 0 or 1*

QUESTION: *Is there any cycle C in G that the sum of labels of all edges in C is odd?*

Proof. The search for the required cycle can be done nondeterministically in logspace by guessing next edge in the cycle and computing the parity of the sum of labels online. Since the graph is undirected, the search can be implemented in symmetric logspace [14]. By [23] $\mathcal{SL} = \mathcal{L}$. Since \mathcal{L} is closed under complementation, it follows that UNSIGNED PLANARITY is in \mathcal{L} . \square

3.5 LOGSPACE to DRA

For a word w we define its variability $v(w)$ as a number of distinct symbols in w . For a language L and an integer function $f(n)$ we say that variability of L is of the order $f(n)$ iff $\min_{w \in L, |w|=n} v(w) \geq f(n)$, i.e. $f(n)$ is a lower bound of variabilities of words of length n in L .

Lemma 2. *Computations of a Turing Machine on a work tape T of size $c \cdot \log(k)$ over a binary alphabet can be simulated by Register Automata model on an input string S , where $v(S)=k$.*

Proof. Assume that a head H is on a work tape T at the position $T(i)$. First all operations under the tape head such as rewriting, checking current symbol on a tape and head moves can be simulated by operations on two pushdown stacks, where first stack keeps the front part of T : $T(0) \dots T(i)$ and the second part of T , is stored in the second pushdown stack with $T(i+1)$ on the top [19]. Also it is easy to see that a binary word on a stack can be represented by an integer x , where empty stack corresponds to 1 value, push a 0 onto the top of a stack corresponds to $x \mapsto 2x$ and pushing a 1 corresponds to $x \mapsto 2x + 1$; checking the top symbol can be done by checking divisibility by 2 and popping of a 1 or 0 can be done by operations $x \mapsto (x-1)/2$ or $x \mapsto (x-1)/2$ respectively. Further we notice that the register automata on an input with k distinct symbols can simulate any finite number of counters of size k^c for any constant c with the following main operations: increment, decrement, multiplication by 2, division by 2 and zero testing. Storing a symbol x in a register represents a value $ord(x)+1$ of a counter. The implementation of all required operations is straightforward albeit tedious. The operation of increment (decrement) by one can be implemented by moving forward to the first occurrence of the next (previous) distinct symbol on an input string [15]. Testing of a counter to be equal to 1 corresponds to

checking whether a stored symbol appear as a first symbol of an input. Also it is well known that operation of multiplication (or division) by 2 and divisibility by 2 can be implemented by a finite number of extra counters with increment and decrement operations and zero testing (or testing to be equal to 1) [19]. Thus, all operations for integer representations of pushdown stacks can be implemented by a register automaton. In this case since the virtual counters can store a value of size k^c and simulate two pushdown stacks of size $\log(k^c)$ we have that the length of a work tape T which is simulated by register automaton on an input with k distinct symbols is bounded by $\log(k^c)$. \square

Theorem 2. *Given a language L with a variability of the order $f(n)$, if a finite projection L_{binary} of L belongs to $\text{SPACE}(\log(f(n)))$ then L can be recognised by a register automata.*

Proof. The binary code of each symbol in $w \in L$ is of a logarithmic size from the number of distinct symbols in w . So for any word of size n over an infinite alphabet, the length of the binary code for each symbol is no more than $\log(n)$. Since the number of distinct symbols in any word $w \in L$ is at least $f(n)$, thus by Lemma 2 we can simulate virtual tape over a binary alphabet of a length $c \cdot \log(f(n))$ for any integer constant c . Therefore register automaton can take any symbol y from an infinite alphabet on an input string and convert it into a finite binary representation in the virtual work tape over a binary alphabet, by counting the number of previously appeared distinct symbols from the beginning of an input string and storing it on a virtual tape, i.e. converting y into a binary representation of $\text{ord}(y)$. Also by the same Lemma 2 we have that any extra memory of size $O(\log(f(n)))$ can be implemented by a finite number of registers on a language L with a variability of the order $f(n)$. So any computations over language L_{binary} that requires $\text{SPACE}(\log(f(n)))$ can be implemented by a register automaton on a language L with a variability of the order $f(n)$. \square

Corollary 3. *UNSIGNED PLANARITY is in DRA.*

The above corollary follows from Proposition 4 and Theorem 2.

4 Conclusion

In this paper we have investigated the complexity of planarity of knot diagrams represented by Gauss words. We have shown that planarity of unsigned Gauss words can be recognised in deterministic logarithmic space on classical computational models and by deterministic register automata over infinite alphabets. To demonstrate these results we have used generic mutual simulation between both computations models for the languages of bounded variability. Notice that unlike the case of signed Gauss words [15] we do not provide explicit deterministic LOGSPACE bounded decision procedure for planarity of unsigned Gauss words and rather refer to the general reduction of \mathcal{SL} to \mathcal{L} [23]. An explicit deterministic LOGSPACE algorithm for the later case as well as the comparison of its complexity with the algorithm(s) for signed case is a topic for further work.

References

1. Abramsky, S.: Temperley-Lieb Algebra: From Knot Theory to Logic and Computation via Quantum Mechanics. *Mathematics of Quantum Computation and Quantum Technology* (2007)
2. Bjorklund, H., Schwentick, T.: On Notions of Regularity for Data Languages. *Lectures Notes in Computer Science* **4639** (2007) 88
3. Cairns, G., Elton, D.: The planarity problem for signed Gauss words. *Journal of Knot Theory and its Ramifications* **2**(4) (1993) 359–367
4. Cairns, G., Elton, D.: The Planarity Problem II. *Journal of Knot Theory and its Ramifications* **5** (1996) 137–144
5. De Fraysseix, H., Ossona de Mendez, P.: On a characterization of Gauss codes. *Discrete and Computational Geometry* **22**(2) (1999) 287–295
6. Dehn, M.: Über kombinatorische topologie. *Acta Math* **67** (1936) 123–168
7. Freivalds, R.: Knot Theory, Jones Polynomial and Quantum Computing. *Lectures Notes in Computer Science* **3618** (2005) 15
8. Gauss, C.: *Werke*. Band 8. Teubner (1900)
9. Hass, J., Lagarias, J., Pippenger, N.: The computational complexity of knot and link problems. *Journal of the ACM (JACM)* **46**(2) (1999) 185–211
10. Kaminski, M., Francez, N.: Finite-memory automata. *Theoretical Computer Science* **134**(2) (1994) 329–363
11. Kari, J., Niemi, V.: Morphic Images of Gauss Codes. In: *Developments in Language Theory*. (1993) 144–156
12. Kauffman, L.: Virtual knot theory. *Arxiv Preprint Math. GT/* **9811028** (1998)
13. Kurlin, V.: Gauss paragraphs of classical links and a characterization of virtual link groups. *Mathematical Proceedings Cambridge Phil. Soc.* **145** (2008) 129–140
14. Lewis, H., Papadimitriou, C.: Symmetric space-bounded computation (extended abstract). *Automata, Languages and Programming* (1980) 374–384
15. Lisitsa, A., Potapov, I., Saleh, R.: Automata on Gauss Words. In: *Proceedings of the 3rd International Conference on LATA*, Springer (2009) 505–517
16. Lomonaco Jr, S., Kauffman, L.: Topological Quantum Computing and the Jones Polynomial. *Arxiv Preprint Quant-ph/* **0605004** (2006)
17. Lovász, L., Marx, M.: A forbidden substructure characterization of Gauss codes. *Bull. Amer. Math. Soc.* **82**(1) (1976)
18. Manturov, V.: A proof of Vassiliev’s conjecture on the planarity of singular links. *Izvestiya: Mathematics* **69**(5) (2005) 1025–1033
19. Minsky, M.: *Computation: finite and infinite machines*. (1967)
20. Nagy, J.: Über ein topologisches Problem von Gauss. *Mathematische Zeitschrift* **26**(1) (1927) 579–592
21. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic* **5**(3) (2004) 403–435
22. Read, R., Rosenstiehl, P.: On the Gauss crossing problem. In: *Colloq. Math. Soc. Janos Bolyai*. Volume 18 (1976) 843–876
23. Reingold, O.: Undirected connectivity in log-space. *Journal of the ACM (JACM)* **55**(4) (2008) 17
24. Rosenstiehl, P.: Solution algebrique du probleme de Gauss sur la permutation des points d’intersection d’une ou plusieurs courbes fermees du plan. *CR Acad. Sci. Paris Ser. AB* **283** (1976)
25. Rosenstiehl, P., Tarjan, R.: Gauss codes, planar Hamiltonian graphs, and stack-sortable permutations. *Journal of algorithms* **5**(3) (1984) 375–390
26. Shtylla, B., Traldi, L., Zulli, L.: On the realization of double occurrence words. *Discrete Mathematics* **309**(6) (2009) 1769–1773