Relating L versus P to Reversal versus Access and Their Combinatorial Structures

Kenya UENO\(^{(a)}\), Student Member

SUMMARY  Reversal complexity has been studied as a fundamental computational resource along with time and space complexity. We revisit it by contrasting with access complexity which we introduce in this study. First, we study the structure of space bounded reversal and access complexity classes. We characterize the complexity classes \(L, P\) and \(PS\ PACE\) in terms of space bounded reversal and access complexity classes. We also show that the difference between polynomial space bounded reversal and access complexity is related with the \(L\ versus\ P\) problem. Moreover, we introduce a theory of memory access patterns, which is an abstracted structure of the order of memory accesses during a random access computation, and extend the notion of reversal and access complexity for general random access computational models. Then, we give probabilistic analyses of reversal and access complexity for almost all memory access patterns. In particular, we prove that almost all memory access patterns have \(\omega(\log n)\) reversal complexity while all languages in \(L\) are computable within \(O(\log n)\) reversal complexity.

**key words:** complexity classes, random combinatorial structures

1. Introduction

Several computational resources such as time [9], space [18] and reversal [8] are studied to see the hierarchical structures in the computable class and to clarify the notion of feasible computations. Time and space are always in the center of our attention because of their importance for realistic computational models. Fundamental complexity classes such as \(L, P\) and \(PS\ PACE\) are defined according to time and space complexity.


Besides time, space and reversal, we introduce a new notion of computational resource “access” and change the viewpoint of the existing studies on reversal complexity by revisiting it by contrasting with access complexity. Access complexity is defined as the maximum number of accesses in all space used during the computation.

Although our new notion “access complexity” seems to be natural and fundamental, there were no studies which treats “access” as a computational resource. This is because constant accesses bounded computations can recognize all computable languages (This can be understood immediately from the proof of Lemma 3.8). Thus, access bounded computations are too strong to make a hierarchy of computable classes and hence it seems to be not useful as a computational resource. On the contrary, in this paper, we show that similarities and differences between space bounded reversal and access complexity gives us a conceptually beautiful viewpoint for complexity classes such as \(L, P\) and \(PS\ PACE\).

Our introduction of access complexity is motivated by just a theoretical interest with beautiful resemblance between reversal and access complexity. It may be true that the notion of access complexity doesn’t give us an outstanding result for separating complexity classes. However, we obtain some merits to gain a better understanding of the structure of sequential computations.

In Sect. 2, we define \(DS\ PACE−REV(s(n), r(n))\) as the \(O(s(n))\) space and \(O(r(n))\) reversal bounded complexity class and \(DS\ PACE−ACCESS(s(n), a(n))\) as the \(O(s(n))\) space and \(O(a(n))\) access bounded complexity class for the multitape Turing machine model.

In Sect. 3, we study the structure of space bounded reversal and access complexity classes. First, we give characterizations of the complexity classes \(L, P\) and \(PS\ PACE\) in terms of space bounded reversal and access complexity classes by the same way as two figures described below. Then, we discuss differences between space bounded reversal and access complexity. We derive the inclusions \(DS\ PACE−REV(poly, 1) \subseteq L \subseteq DS\ PACE−REV(poly, \log)\) from the work of [4], [17], [20]. On the other hand, we prove \(P = DS\ PACE−ACCESS(poly, 1)\). As a consequence, the difference between space bounded reversal and access complexity is related with the \(L\ versus\ P\) problem. We also mention some further results on the structure of space bounded reversal and access complexity classes.

In Sect. 4, we introduce a notion of memory access patterns as an abstracted structure of the order of memory accesses during a random access computation. We consider a set of memory cells \(M = \{1, 2, \cdots, s\}\). A memory access pattern of \(t\) time and \(s\) space complexity is defined as \(p = (a_1, a_2, \cdots, a_t)\) where \(a_i \in M\) (1 \(\leq i \leq t\)). We as-

---

\(^{(a)}\)The author is with the Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, 113–8656 Japan.

a) E-mail: kenya@s.s.u-tokyo.ac.jp

DOI: 10.1093/ietisy/e91-d.12.2776
sume $s \leq t$. An interesting feature of reversal and access complexity is that their values are uniquely determined by a memory access pattern. Therefore, we can define a generalized notion of reversal and access complexity for each memory access pattern.

If we clarify the structure of reversal and access complexity for memory access patterns in actual computation processes arising in the Turing machine or the random access memory model, it may follow a progress toward separating complexity classes. However, it seems to be as difficult as separating complexity classes. Thus, we focus on a relatively simpler setting, random memory access patterns rather than memory access patterns in actual computation processes. This is intended as a first step toward further development of the structure of actual memory access patterns, which seems to be quite difficult.

In the remaining part of Sect. 4, we give probabilistic analyses of memory access patterns. We show that almost all memory access patterns of $t$ time and $s$ space complexity have $\Theta(\sqrt{s/t})$ reversal complexity. We also show that almost all memory access patterns have $O(t + \log t)$ access complexity. We can achieve this by novel uses of the birthday problem and the occupancy problem, while a direct method for the analysis seems to be technically difficult.

As a consequence, we obtain a quantitative analysis of the difference between reversal and access complexity, which is related with the $L$ versus $P$ problem, in terms of random combinatorial structures. One of the difficulties of separating complexity classes arises from their intractable notions as sets of languages and hence combinatorial structures such as logical circuits, which is easier to handle than complexity classes, become strong candidates for this purpose. Speaking from this perspective, the reversal versus access problem, compared with the $L$ versus $P$ problem, is intuitively easy to grasp their combinatorial structures.

Furthermore, we have the following observation. In 1949, the pioneering work of Shannon [16] showed that almost all Boolean functions have exponential circuit complexity by a counting argument without any explicit constructions. On the other hand, it is well known that languages in $P$ have polynomial size circuit complexity. Thus, studying structures and properties of Boolean functions with polynomial size circuit complexity occupy important meanings for separating $P$ and other complexity classes such as $NP$. One of the interesting consequences of our results is that almost all memory access patterns have $\omega(\log n)$ reversal complexity when we take $t = \text{poly}(n)$. On the other hand, all languages in $L$ can be computed by a random access computation model within $O(\log n)$ reversal complexity because we have $L \subseteq \text{DSPACE} - \text{REV}(\text{poly}, \log)$. We can interpret these results as an analogue of Shannon’s theorem by considering the complexity class $L$ instead of $P$ and memory access patterns instead of Boolean functions.

In the last section, we will discuss future directions from the conclusions of this paper. We will describe some possible directions toward the structure of actual memory access patterns in actual computation processes.

2. Definitions of Complexity Classes

In this section, we define complexity classes by the offline multitape deterministic Turing machine model. We abbreviate an offline multitape deterministic Turing machine as a DTM. We assume that readers are familiar with the notion of the Turing machine. First, we define the following fundamental complexity classes.

**Definition 2.1.** $L$ is the class of languages recognized in logarithm space by a DTM. $P$ is the class of languages recognized in polynomial time by a DTM. $\text{PSPACE}$ is the class of languages recognized in polynomial space by a DTM. $\text{DSPACE}(s(n))$ is the class of languages recognized by a DTM using $s(n)$ space.

For these complexity classes, it is known that $L \neq$...
PS PACE and $L \subseteq P \subseteq PS PACE$, but whether these inclusions are proper or not is one of the most important open problems in theoretical computer science.

Then, we explain the notion of reversal complexity. For more details about reversal complexity, see [4] or chapter 2 of [20]. Reversals are direction changes made by tape heads of the Turing machine. Reversal complexity is the total number of reversals made by all tape heads including the input tape. We define the space and reversal bounded complexity class as follows. Here, $n$ means the input length.

Definition 2.2. $DS P A C E – R E V (s(n), r(n))$ is the class of languages recognized by a DTM using $O(s(n))$ space and $O(r(n))$ reversals. $DT I M E – R E V (\tau(n), r(n))$ is the class of languages recognized by a DTM using $O(\tau(n))$ time and $O(r(n))$ reversals. $D R E V (r(n))$ is the class of languages recognized by a DTM using $O(r(n))$ reversals.

We say that a function $f(n)$ is $r(n)$ reversal (and $s(n)$ space) constructible if there is a multitape Turing machine which produces the unary form of $f(n)$ on one of its work tapes within $O(r(n))$ reversals (and $O(s(n))$ space) from the input $n$ in unary form. We also say that a function $f(n)$ is reversal constructible if it is $f(n)$ reversal constructible. For the details of reversal constructibility, see [4], [20].

Now, we introduce a new complexity resource. Access complexity is defined as the maximum number of accesses in all tape cells of the Turing machine. In other words, it is the maximum number of times any one cell is scanned by a head of the Turing machine. Accesses made by the input tape head are also counted. If a tape head stays in the same tape cell without any movements, we don’t count it. It means that stationary steps don’t increase the number of accesses. Then, we define the space and access bounded complexity class as follows.

Definition 2.3. $DS P A C E – A C C E S S (s(n), a(n))$ is the class of languages recognized by a DTM using $O(s(n))$ space and at most $O(a(n))$ accesses for each tape cell.

We write $\log$ as the logarithmic function and use conventions $poly(n) = \bigcup_{a \geq 1} n^a$ and $exp(n) = 2^{poly(n)}$.

3. Space Bounded Reversal and Access Complexity Classes

In this section, we discuss the reversal versus access problem from the viewpoint of complexity classes. First of all, the following lemma is essential.

Lemma 3.1.

$$DS P A C E – R E V (s(n), r(n)) \subseteq DS P A C E – A C C E S S (s(n), r(n)).$$

Proof. Let $M$ be a deterministic Turing machine which works within $c \cdot r(n)$ reversals for some constant $c$. Then, access complexity of $M$ is at most $c \cdot (r(n) + 1)$ because at least $k$ reversals are necessary to access some cell $k + 1$ times.

3.1 New Characterizations of L, P and PS PACE

We characterize the complexity classes $L$, $P$ and $PS P A C E$ in terms of space bounded reversal and access complexity classes. First, we show equalities between $P$ and polynomial space bounded reversal and access bounded complexity classes.

Proposition 3.2.


Proof. First, we have

$$DT I M E (\tau(n)) \subseteq DS P A C E – R E V (\tau(n), \tau(n))$$

because $\tau(n)$ time bounded computations use at most $\tau(n)$ space and $\tau(n)$ reversals. Thus, we have $P \subseteq DS P A C E – R E V (poly, poly)$. Next, we prove the converse inclusion by showing

$$DS P A C E – A C C E S S (s(n), a(n)) \subseteq DT I M E ((n + s(n)) \cdot a(n)).$$

Let $M$ be a deterministic Turing machine which works within $n + c_1 \cdot s(n)$ space, which means a total space of all tapes including the input tape, and $c_2 \cdot a(n)$ accesses for some constant $c_1$ and $c_2$. Because states, tapes and tape alphabets of $M$ are finite sets, there is some constant $k$ such that within $k$ steps some head should move. Stationary steps don’t continue more than $k$. The number of accesses of some cell increases within each $k$ steps. Thus, $M$ works within $k \cdot (n + c_1 \cdot s(n)) \cdot c_2 \cdot a(n)$ time. Combined with Lemma 3.1, we have shown the theorem.

Next, we show the following proposition. It means that space bounded upper reversal and access complexity tend to be similar.

Proposition 3.3. For any $s(n)$, there is some function $r'(n)$ which satisfies

$$DS P A C E – R E V (s(n), r(n)) = DS P A C E – A C C E S S (s(n), r(n))$$

for all $r(n) \geq r'(n)$.

Proof. Let $r(n) \geq r'(n) = 2^{c \cdot s(n)}$ for some constant $c$. From the definition, $DS P A C E – A C C E S S (s(n), r(n)) \subseteq DS P A C E (s(n))$. Moreover, we have $DS P A C E (s(n)) \subseteq DS P A C E – R E V (s(n), r(n))$ because $s(n)$ space bounded computations can work within $2^{c \cdot s(n)}$ time without loss of generality. This is clear from the proof of $L \subseteq P$. Thus, they can work within $r'(n)$ reversals. Combined with Lemma 3.1, we have shown the theorem.

As a corollary of this proposition, the complexity classes $L$ and $PS P A C E$ are characterized as follows.
3.2 Relating L versus P to Reversal versus Access

Corollary 3.4.

\[ L = \text{DSPACE} - \text{REV}(\log, \text{poly}) \]
\[ = \text{DSPACE} - \text{ACCESS}(\log, \text{poly}), \]
\[ \text{PSPACE} = \text{DSPACE} - \text{REV}(\text{poly}, \text{exp}) \]
\[ = \text{DSPACE} - \text{ACCESS}(\text{poly}, \text{exp}). \]

3.2 Relating L versus P to Reversal versus Access

Now, we discuss differences between space bounded reversal and access complexity classes. Yap [20] proved the following inclusion as an extension of \( \text{DTIME} - \text{REV}(t(n), r(n)) \subseteq \text{DSPACE}(r(n) \log t(n)) \) in [17].

Theorem 3.5 [17], [20]. If \( s(n) \geq \log n \),

\[ \text{DSPACE} - \text{REV}(s(n), r(n)) \]
\[ \subseteq \text{DSPACE}(r(n) \log s(n)). \]

Chen and Yap [4] proved the following theorem.

Theorem 3.6 [4]. \( \text{DSPACE}(s(n)) \subseteq \text{DREV}(s(n)) \) for any reversal constructible functions \( s(n) \geq \log n \).

The proof is achieved by a simulation method, which is done within polynomial space if \( s(n) = \log n \). Thus, we can conclude the following result as a corollary of two theorems.

Corollary 3.7 [4], [17], [20].

\[ \text{DSPACE} - \text{REV}(\text{poly}, 1) \subseteq L, \]
\[ L \subseteq \text{DSPACE} - \text{REV}(\text{poly}, \log). \]

On the other hand, we show the following lemma.

Lemma 3.8.

\[ \text{DTIME}(t(n)) \subseteq \text{DSPACE} - \text{ACCESS}(t(n)^2, 1). \]

Proof. The main idea is that we use new space 1 step by 1 step in order to bound the number of accesses within \( O(1) \). For this purpose, we multiply work tapes twice to simulate by copying contents of tapes to each other alternatively.

Let \( M \) be a deterministic Turing machine which works within \( t(n) \) time. We assume that \( M \) has \( k \) tapes including the input tape. We construct a deterministic Turing machine \( M' \) with \( 2k \) tapes not including the input tape. It will simulate \( M \) within \( t(n)^2 \) space and \( O(1) \) accesses. We add a special symbol \( \downarrow \) to the tape alphabet of \( M' \) to represent a position of a tape head of \( M \). We also add a special symbol $ to the tape alphabet of \( M' \) to distinguish the most right end of a tape of \( M \). All tapes of \( M' \) are divided into blocks by \$. Each block contains a representation of a configuration of \( M \).

We divide \( 2k \) tapes into the first \( k \) tapes and the second \( k \) tapes. At the beginning of the simulation, \( M' \) writes down the initial configuration of \( M \) on the first \( k \) tapes and the second \( k \) tapes, respectively. \( M' \) marks $ at the most right ends of all tapes. Here, the contents of the input tape of \( M \) are copied to the first tape of each \( k \) tapes of \( M' \), which corresponds to the input tape in each two kinds of \( k \) tapes.

\( M' \) simulates each step of \( M \) as follows. First, \( M' \) copies the last block of the first \( k \) tapes to the right most part of the second \( k \) tapes, which have not been used before. $'s are written on the right most cell of the second \( k \) tapes. Next, \( M' \) moves its heads to \( \downarrow \) in the last blocks of all tapes. Finally, \( M' \) simulates 1 step of \( M \). \( M' \) writes $'s and tape alphabets at appropriate positions according to the transition rules of \( M \). \( M' \) repeats this by alternating the roles of the first \( k \) tapes and the second \( k \) tapes. States of \( M \) are simulated by states of \( M' \) as products of two states.

\( M' \) works within \( O(1) \) accesses. Moreover, the size of all configurations of \( M \) are within \( t(n) \) and \( M' \) copies this \( t(n) \) times. Thus, the total space used by \( M' \) is at most \( t(n)^2 \). \( \square \)

Thus, we have the following theorem with Proposition 3.2.

Theorem 3.9. \( P = \text{DSPACE} - \text{ACCESS}(\text{poly}, 1) \).

This means that the complexity class \( P \) is equal to anything between \( \text{DSPACE} - \text{ACCESS}(\text{poly}, 1) \) and \( \text{DSPACE} - \text{ACCESS}(\text{poly}, \text{poly}) \). Thus, the difference between \( L \) and \( P \) is characterized by the gap between polynomial space bounded reversal and access complexity classes.

3.3 Further Results on Complexity Classes

In this subsection, we mention a part of further results on space bounded reversal and access complexity classes. Readers can skip to the next section without reading this subsection.

First, we show that space bounded reversal and access complexity tend to be different if we allow large space. Yap [20] proved \( \text{DREV}(r(n)) \subseteq \text{DTIME} - \text{REV}(n \cdot 2^{r(n)}, r(n)) \). By using this result, we can prove the following proposition.

Proposition 3.10. For any \( r(n) \), there is some function \( s'(n) \) which satisfies

\[ \text{DSPACE} - \text{REV}(s(n), r(n)) \]
\[ \subseteq \text{DSPACE} - \text{ACCESS}(s(n), r(n)) \]

for all \( s(n) \geq s'(n) \).

Proof. Because we have

\[ \text{DREV}(r(n)) \subseteq \text{DTIME} - \text{REV}(n \cdot 2^{r(n)}, r(n)) \]

from the work of Yap [20], we obtain

\[ \text{DSPACE} - \text{REV}(s(n), r(n)) \subseteq \text{DREV}(r(n)) \]
\[ \subseteq \text{DTIME} - \text{REV}(n \cdot 2^{r(n)}, r(n)) \]
\[ \subseteq \text{DTIME}(n \cdot 2^{r(n)}). \]

From Lemma 3.8, we have
for a sufficiently large \( (s'(n))^2 = s(n) \). Note that the first inclusion is proper from the time hierarchy theorem. As a consequence, we can conclude the proposition. □

From the same argument of this proposition, we can obtain the following corollary.

**Corollary 3.11.**

\[
\begin{align*}
\text{DSPACE} - \text{REV}(\text{poly}, 1) & \subseteq \text{DSPACE} - \text{ACCESS}(s(n), 1) \\
\text{DSPACE} - \text{REV}(\exp, \log) & \subseteq \text{DSPACE} - \text{ACCESS}(s(n), r(n)).
\end{align*}
\]

Next, we focus on the sublinear area of space bounded reversal and access complexity. Let \( M \) be an offline multitape deterministic Turing machine. For the Turing machine \( M \), let \( s(n) \) be the work space used and \( r(n) \) be the number of work tape reversals. Hong [11] proved \( s(n) \times r(n) \in o(n) \) implies \( s(n) \times r(n) \in O(1) \). We extend this result as follows.

**Proposition 3.12.** Let \( M \) be an offline multitape deterministic Turing machine. For the Turing machine \( M \), let \( s(n) \) be the work space used and \( a(n) \) be the maximum number of accesses in all work tape cells. Then, \( s(n) \times a(n) \in o(n) \) implies \( s(n) \times a(n) \in O(1) \).

**Proof.** We assume \( s(n) \times a(n) \in o(n) \). Then, we have \( t(n) \leq s(n) \times a(n) \in o(n) \) because the product of work tape space and the maximum number of work tape accesses is larger than or equal to \( t(n) \). Here, we denote \( t(n) \) as time complexity of work tapes.

We define \( L_t = \{ x \mid M \text{ makes exactly } t \text{ movements of the work tape heads on the input } x \} \). We assume \( L_t \neq \emptyset \) and define \( l(t) = \min \{ |x| \mid x \in L_t \} \). From this definition, the inequality \( l(t) < k \cdot (t + 1) \) contradicts \( t(n) \in o(n) \). Here, we take a sufficiently large constant \( k \), which depends only on the number of tapes and tape alphabets.

Then, we would like to derive a contradiction from the assumption of \( l(t) \geq k \cdot (t + 1) \). We call an interval \( I(i, j) \) as a range of the input string from \( i \)-th cell to \( j \)-th cell. If we assume \( l(t) \geq k \cdot (t + 1) \), then there is an interval \( I(i, i + k) \) such that all work tape heads do not move at all whenever the input head is inside \( I(i, i + k) \). Suppose the input is \( x = a_1a_2a_3 \cdots a_n \). Then, there are two indices \( i_1 \) and \( i_2 \) \((i_1 + 1 < i_2)\) between \( i \) and \( i + k \) such that the computations on inputs \( x \) and \( x' = a_1a_2 \cdots a_i a_{i+1} \cdots a_n \) have the same work tape movements from the definition of \( k \). Thus, we can shorten the input length. This contradicts the definition of \( l(t) \).

Therefore, there exists some constant \( t' > 0 \) such that \( L_t = \emptyset \) for any \( t \geq t' \). This implies \( s(n) \times a(n) \in O(1) \). □

Thus, there are no difference between space bounded reversal and access complexity classes on the sublinear area.

**Corollary 3.13.** If \( s(n) \times r(n) \in o(n) \),

\[
\begin{align*}
\text{DSPACE} - \text{REV}(s(n), r(n)) & = \text{DSPACE} - \text{ACCESS}(s(n), r(n)).
\end{align*}
\]

### 4. A Theory of Memory Access Patterns

In this section, we introduce a theory of memory access patterns and generalize the notion of reversal and access complexity. Then, we give probabilistic analyses of reversal and access complexity for almost all memory access patterns. We denote \( Pr[X] \) as the probability of \( X \) which means the occurrence of the event \( X \). We say that some event \( X \) occurs with high probability when its probability \( Pr[X] \) is more than \( 1 - o(1) \). A property for almost all memory access patterns means that it occurs with high probability when we sequentially access memories uniformly at random.

#### 4.1 Introduction of Memory Access Patterns and Generalizations of Reversal and Access Complexity

We extend the notion of space bounded reversal and access complexity for general random access computational models by introducing a notion of memory access patterns. A memory access pattern is a structure which represents the order of memory accesses during a random access computation.

The same discrete structure has been studied in the theory of programming languages and compiler optimization to analyze the data locality [21]. The studies on online computation of paging algorithms also treats it as a page request [3]. Moreover, data stream algorithms handle it as data stream [1]. In this paper, we show that this structure is also meaningful from complexity theoretic perspective.

**Definition 4.1.** We consider a set of memory cells \( M = \{1, 2, \cdots, s \} \). Then, a memory access pattern of \( t \) time and \( s \) space complexity is defined as \( p = (a_1, a_2, \cdots, a_t) \) where \( a_i \in M \) \((1 \leq i \leq t)\). Because \( t \) time bounded computation cannot use more than \( t \) space, we assume \( s \leq t \).

An interesting feature of reversal and access complexity is that their values are uniquely determined by each memory access pattern. Thus, we can define reversal and access complexity from each memory access pattern.

**Definition 4.2.** Access complexity of a memory access pattern \( p = (a_1, a_2, \cdots, a_t) \) is defined as \( \max_{m \in M} \alpha(m) \) where \( \alpha(m) \) is the number of appearances of \( m \) in \( p \).

An appearance of \( m \) in \( p \) corresponds to a memory access for the memory cell \( m \). Thus, the defined value corresponds to the maximum number of accesses in general computational models.

On the other hand, we can also define reversal complexity for each memory access pattern. A phase is a period of the computation during which information written on the work space in the same phase will not be read. This means
that computational operations of each phase can be carried out independently. So, we can see that reversal complexity is related with the parallel time complexity. The total number of phases during the whole computation corresponds to reversal complexity. Formally, this can be stated as follows.

Definition 4.3. Reversal complexity of a memory access pattern \( p = (a_1, a_2, \ldots, a_i) \) is defined as follows. We assume that each memory cell \( m \in M \) has two states, marked and unmarked. At the beginning, all memory cells are initialized to unmarked. We have a counter initialized to 0. Then, we sequentially check elements of \( p \) from \( a_1 \) to \( a_i \). If the element \( a_i \) is unmarked, then we change the state of \( a_i \) into marked. If the element \( a_i \) is marked, then we change states of all memory cells except \( a_i \) into unmarked and increment the counter. Then, we go to the next element \( a_{i+1} \). After we check all the elements of \( p \), the value of the counter is reversal complexity of \( p \).

Note that this definition of reversal complexity for memory access patterns can be viewed as a generalization of that defined for the multitape Turing machine because we can assume that tape heads of the multitape Turing machine move in turn without loss of generality. We also remark that we can ignore successively multiple accesses to the same memory cell without loss of generality because they can be simulated by one access to it.

4.2 Probabilistic Analysis of Access Complexity

First, we consider access complexity for almost all memory access patterns. For the analysis, we consider random allocation processes of balls into bins. The balls into bins game has been studied in the theory of hashing [7] and dynamic resource allocation [2]. The occupancy problem is concerned with the maximum number of balls in any bin after we sequentially throws \( m \) balls into \( n \) bins uniformly at random. Its upper bound has been studied in the literature [7] and a tight analysis was given in [15] as follows.

Theorem 4.4 [15]. Let \( M \) be the random variable that counts the maximum number of balls in any bin when we sequentially throw \( m \) balls into \( n \) bins uniformly at random. Then, \( \Pr[M > k_0] = o(1) \) if \( \alpha > 1 \) and \( \Pr[M > k_0] = 1 - o(1) \) if \( 0 < \alpha < 1 \), where

\[
k_0 = \begin{cases} \log n \left(1 + \frac{\log^2 \left(\frac{n \log n}{m}\right)}{\log \frac{n \log n}{m}}\right) - \frac{n}{\text{polylog}(n)} \leq m \ll n \log n \leq (d_c - 1 + \alpha) \log n \leq (m = c \cdot n \log n \text{ for some constant } c) \leq \frac{m}{n} + \alpha \sqrt{\frac{2}{n} \log n} \leq (n \log n \ll m \leq n \cdot \text{polylog}(n)) \leq \frac{m}{n} + \sqrt{\frac{2m \log n}{n} \left(1 - \frac{1}{\alpha \cdot 2 \log n}\right)} \leq (n \cdot (\log n)^3 \ll m).
\end{cases}
\]

Here, \( d_c \) denotes a suitable constant depending only on \( c \).

From the analysis of the occupancy problem, we can analyze access complexity for almost all memory access patterns.

Theorem 4.5. Almost all memory access patterns of \( t \) time and \( s \) space complexity have \( O\left(\frac{t}{s} + \log t\right) \) access complexity.

Proof. We consider the balls into bins game as a computational process. We regard a series of movements of balls into bins as a memory access pattern of the computation. Then, the number of balls and bins correspond to time and space complexity, respectively. In the case of the balls into bins game, a notion corresponding to access complexity is the maximum number of balls in any bins. Thus, we can derive access complexity of almost all memory access patterns because we can reinterpret a random structure with high probability as a property for almost all structures. \( \square \)

4.3 Probabilistic Analysis of Reversal Complexity

Then, we discuss reversal complexity for almost all memory access patterns. Similar to the case of access complexity, we analyze this by considering random allocation processes of balls into bins. A proof is done by a novel use of the result of the classical problem in probability theory, known as the birthday problem [5], [6], [10], [12], [19]. The birthday problem is a problem which treats the probability that at least two persons have the same birthday when there are \( m \) people in the room.

We can formalize the birthday problem in terms of the balls into bins game. We consider a process in which we sequentially throw \( m \) balls into \( n \) bins uniformly at random. In the case of the birthday problem, \( m \) is the number of people in the room and \( n \) is the number of days in a year. It is concerned with the probability that at least two balls are in the same bin after we throw \( m \) balls into \( n \) bins. In the ball into bins game, reversal complexity corresponds to the number of phases where a phase is a period of ball-throws during which no two balls are thrown in the same bin.
We denote $Φ(n)$ as the random variable which denotes the number of balls in a phase where $n$ is the number of bins. Note that the probability $Φ(n) ≥ k$ is equal to the probability that all $k$ balls are in distinct bins after we throw $k$ balls into $n$ bins:

$$Pr[Φ(n) ≥ k] = 1 - \left(1 - \frac{1}{n}\right) \cdots \left(1 - \frac{k-1}{n}\right) ≤ e^{-\frac{k^2}{2n}}.$$ 

If $k$ is much larger than $\sqrt{n}$, this value goes to 0.

Klamanik and Newman [12] estimated the expected value of the birthday problem. We can interpret this result in terms of the number of balls in a phase as follows. Here, $Γ(x)$ is the Gamma function.

**Theorem 4.6** [12]. \(\lim_{n→∞} \frac{E[Φ(n)]}{\sqrt{n}} = \sqrt{2} \cdot Γ(1.5).\)

Dwass [5] analyzed the limit distribution of the birthday problem. Another method for the analysis was given by Henze [10]. We can also interpret this result in terms of $Φ(n)$ as follows.

**Theorem 4.7** [5], [10].

$$\lim_{n→∞} Pr\left[\frac{Φ(n)}{\sqrt{n}} ≤ t\right] = 1 - \exp\left(-\frac{t^2}{2}\right).$$

It means that the limit distribution of $\frac{Φ(n)}{\sqrt{n}}$ is given by a Weibull distribution. In general, the Weibull cumulative distribution function is given by $F(X) = 1 - \exp(-\frac{X}{β})$. Its expected value and variance are given by $E(X) = a \Gamma\left(1 + \frac{1}{β}\right)$ and $V(X) = a^2 \Gamma\left(1 + \frac{2}{β}\right) - \Gamma\left(1 + \frac{1}{β}\right)$ respectively. Thus, the variance of $Φ(n)$ is given as follows.

**Corollary 4.8** [5], [10].

$$\lim_{n→∞} \frac{V[Φ(n)]}{\sqrt{n}} = 2 \cdot (Γ(2) - Γ(1.5)).$$

Now, we give an analysis of reversal complexity for almost all memory access patterns. Although a direct analysis is technically difficult, we can achieve this by utilizing the result of the birthday problem.

**Theorem 4.9.** Almost all memory access patterns of $t$ time and $s$ space complexity have $Θ(\frac{1}{\sqrt{n}})$ reversal complexity.

**Proof.** Intuitively speaking, the number of memory accesses in a phase is $Θ(\sqrt{n})$ with high probability and hence reversal complexity is $Θ(\frac{1}{\sqrt{n}})$ with high probability when we access $s$ memory cells $t$ times uniformly at random. This can be seen by the following argument.

We consider $r$ independent balls into bins processes in which we sequentially throw balls into $n$ bins uniformly at random until a ball collision. We denote $Φ_1(n), Φ_2(n), \cdots, Φ_r(n)$ as the number of balls thrown before a ball collision. Let $Φ(n) = Φ_1(n) + Φ_2(n) + \cdots + Φ_r(n)$.

From Chebyshev’s inequality, for any $ε > 0$,

$$Pr\left[\frac{Φ(n)}{r} - c \sqrt{\frac{n}{r}} < ε \sqrt{\frac{n}{r}}\right] > 1 - \frac{V[Φ(n)]}{r \cdot (ε \sqrt{\frac{n}{r}})^2},$$

where $c = \sqrt{2} \cdot Γ(1.5)$. Since $r > 1$ and the limit value of $\frac{V[Φ(n)]}{\sqrt{n}}$ goes to a constant, we have the following equation:

$$\lim_{n→∞} Pr\left[(c - ε) \cdot r \cdot \sqrt{n} < Φ(n) < (c + ε) \cdot r \cdot \sqrt{n}\right] = 1.$$

Suppose that the total number of balls is $m = c' \cdot r \cdot \sqrt{n}$ for some constant $c' > c + ε$. We define a notion “round” as a balls-and-bins game that we keep throwing balls until a ball collision occurs. Since $Φ(n)$ is at most $(c + ε) \cdot r \cdot \sqrt{n}$ with high probability, we play at least $r$ rounds before we use all the $m$ balls with high probability. This implies that a ball collides at least $r = m/(c' \sqrt{n})$ times before $m$ balls are thrown.

Similarly to the case of access complexity, we consider the balls into bins game as a computational process. We regard a series of movements of balls into bins as a memory access pattern. Then, reversal complexity corresponds to $Θ(\frac{1}{\sqrt{n}})$ with high probability when we regard bins as memory cells and ball-throws as memory accesses.

Since we assume $s ≤ t$ in the definition, we have the following corollary.

**Corollary 4.10.** If we take $t = poly(n)$, almost all memory access patterns have $ω(\log n)$ reversal complexity.

5. **Concluding Remarks and Future Directions**

In this paper, we have studied the reversal versus access problem from the viewpoint of complexity classes and random combinatorial structures. From this result, we give the following conjecture: Some language in $P$ (or $NP$) is not computable by any random access computational models within $O(\log n)$ reversal complexity.

Thus, we consider that it is interesting to deepen the knowledge of structures on memory access patterns which have $O(\log n)$ reversal complexity as a future directions from this study.

One of the possible candidates for this purpose is to utilize the probabilistic method, which shows properties on extremal combinatorics from the probabilistic argument. There are many technical tools to show the existence of combinatorial properties. They will be useful to show the properties related with the theory of memory access patterns as extremal combinatorics.

Another candidate for the future direction is to introduce the network structure on the theory of memory access patterns. What we need to know for separating complexity classes is the structure of memory access patterns dominated by computation rules while we considered memory access patterns in the case in which we sequentially access memories uniformly at random in this study. For this purpose,
the network structure inside memory access patterns will be useful for the further development of the theory of memory access patterns.

Acknowledgments

The author is grateful to Hiroshi Imai for his valuable advice. The author thanks anonymous referees for their helpful comments. This work is supported by Research Fellowship for Young Scientists from Japan Society for the Promotion of Science (JSPS) and Grant-in-Aid for JSPS Fellows.

References
