The Online Space Complexity of Probabilistic Languages

Nathanaël Fijalkow[1,2,3(✉)]

[1] LIAFA, Université Paris Diderot - Paris 7, Paris, France
[2] University of Warsaw, Warsaw, Poland,
[3] University of Oxford, Oxford, UK
nath@liafa.univ-paris-diderot.fr

Abstract. In this paper, we define the online space complexity of languages, as the size of the smallest abstract machine processing words sequentially and able to determine at every point whether the word read so far belongs to the language or not. The first part of this paper motivates this model and provides examples and preliminary results.

One source of inspiration for introducing the online space complexity of languages comes from a seminal paper of Rabin from 1963, introducing probabilistic automata, which suggests studying the online space complexity of probabilistic languages. This is the purpose of the second part of the current paper.

Keywords: Online complexity · Probabilistic languages · Automata · Online algorithms · Complexity theory

1 The Online Space Complexity

We introduce and study a new *complexity measure*, called *online space complexity*. The purpose of a complexity measure is to quantify the complexity of solving a given problem, focusing on a particular aspect. For instance, the most classical complexity measures are the *time and space complexity*, defined as the amount of time and space used by a Turing machine, while the *circuit complexity* counts the number of gates in a circuit; the *communication complexity* quantifies the amount of communication required when the input is spread among different agents.

The online complexity deals with the difficulty of observing the instance in an online fashion, *i.e.* one letter at a time. We consider deterministic abstract machines, which perform an action each time a letter is read. The task of the machine is to maintain enough information about the word read so far to answer boolean queries. The online space complexity focuses exclusively on space, *i.e.* the amount of information maintained.

A typical example is a machine presented with a sequence of a's and b's, which should at any point determine whether the sequence read so far contains

Work supported by the ANR STOCH-MC.

S. Artemov and A. Nerode (Eds.): LFCS 2016, LNCS 9537, pp. 106–116, 2016.
DOI: 10.1007/978-3-319-27683-0_8

exactly as many a's as b's. The canonical machine solving this problem uses as memory one counter taking integer values, the difference between the number of a's and the number of b's. This machine is of linear size, because after reading (up to) n letters it can be in at most $2n + 1$ different states; as we shall see, it is optimal, meaning that this problem has linear online space complexity.

Although, to the best of our knowledge, the following definition of the online space complexity is new, the concept of online computing is old and has been investigated in various scenarions and under various names. After giving the formal definitions, we will discuss further the relations between our framework and the existing ones.

1.1 Definitions

We fix an *alphabet* A, which is a finite set of letters. An instance of a problem is given by a *word*, which is a finite sequence of letters, often denoted $w = a_0 a_1 \cdots a_{n-1}$, where a_i's are letters from the alphabet A, *i.e.* $a_i \in A$. We say that w has length n. We denote A^* the set of all words and $A^{\leq n}$ the set of words of length at most n.

A computational problem is given by a set of words L, called a *language*; *i.e.* $L \subseteq A^*$. The online space complexity of a language measures the size of an abstract machine able to recognise the language in an online way: the machine processes words letter by letter, and must at any point be able to determine whether the word read so far belongs to the language or not.

The first definition that we give, that of a machine, is not new; it matches the classical definition of *deterministic automata*, except that the set of states is not assumed to be finite.

Definition 1 (Machine). *A machine is given by a (potentially infinite) set $\mathcal{C}$ of states, an initial state $c_0 \in \mathcal{C}$, a transition function $\delta : \mathcal{C} \times A \to \mathcal{C}$ and a set of accepting states $\mathcal{A} \subseteq \mathcal{C}$.*

When processing a word $w = a_0 a_1 \cdots a_{n-1}$, the machine assumes a sequence of states $c_0 c_1 \cdots c_n$, that we call the run on w, defined inductively by $c_{i+1} = \delta(c_i, a_i)$. It is unique, since we assume the machine to be deterministic and the transition function to be a total function. The last state of the run on w is denoted $c(w)$; the word w is accepted if $c(w)$ is accepting, *i.e.* if $c(w) \in \mathcal{A}$, and rejected otherwise. The language recognised by a machine is the set of words accepted by this machine.

Definition 2 (Size of a Machine). *The size of a machine is the function $s : \mathbb{N} \to \mathbb{N}$ defined by $s(n)$ being the number of different states reached by all words of length at most n. Formally:*

$$s(n) = \operatorname{Card} \left\{ c(w) \mid w \in A^{\leq n} \right\}.$$

Note that in complexity theory, it is usual to count the size of an object by the size of its description. Here, it would be natural, instead of counting of many

different states are reached, how many bits are necessary to describe these states; this amounts to consider the logarithm of the number of states. We do not use this definition as it would erase the differences between, for instance, linearly many and quadratically many states.

For two functions $f, g : \mathbb{N} \to \mathbb{N}$, we say that f is smaller than g, denoted $f \leq g$, if it is true component wise: for all n, $f(n) \leq g(n)$.

Definition 3 (Online Space Complexity Class). *For a function $f : \mathbb{N} \to \mathbb{N}$, the class of languages* Online(f) *consists of all languages which are recognised by a machine of size smaller than f.*

1.2 Related Works

The definitions of online space complexity that we gave belong to the research area concerned with *online computing*. Unlike an *offline algorithm*, which has access to the whole input, an *online algorithm* is presented with its input in a restricted online way: it has to process it letter by letter. Various frameworks emerged from this versatile concept: we discuss three of them, *dynamic algorithms*, *streaming algorithms* and *competitive analysis of online algorithms*.

The field of dynamic algorithms, initiated by Patnaik and Immerman [6], focuses on the complexity of maintaining solutions to problems with online inputs. In this setting, the input can go through a series of changes, and the challenge is to store enough information to be able to solve the problem on the modified input. There are two differences between our approach and dynamic algorithms. The first is that whereas in our setting, the changes are only insertions, dynamic algorithms also consider deletions, and sometimes more complicated operations. The second is that the focus of dynamic algorithms is on the time complexity of the machines maintaining the information, whereas we consider instead the state space complexity of these machines, *i.e.* the number of different states they use.

The field of streaming algorithms, initiated by a series of papers (Munro and Paterson [5], then Flajolet and Martin [3], followed by the foundational paper of Alon, Matias and Szegedy [1]), focuses on algorithms having very limited available memory, much smaller than the input size. The challenge there is to use these constrained ressources to compute relevant information about the processed input, such as for instance statistics on frequency distributions. In this setting the input is a word, read letter by letter, and the focus is put on measuring the memory consumed by the machines processing the word, exactly is in our setting. The only difference is that streaming algorithms also have limited processing time per letter, whereas we abstract away this information, and only measure the state space complexity.

The field of competitive analysis of online algorithms, initiated by Sleator and Tarjan [8], and by Karp [4], compares the performances between offline and online algorithms. In this setting, each solution is assigned a real value, assessing its quality. An offline algorithm, having access to the whole input, can select the best solution. An online algorithm, however, has to make choices ignoring part

of the input that is still to be read. The question is then whether there exists online algorithms that can perform nearly as good as offline algorithms, up to a competitive ratio. In this setting, the complexity of the machines is ignored, and the only question is what is the cost of making online choices rather than processing the whole input.

2 Preliminary Results

2.1 Remarks and Examples

We begin with a few simple remarks. The first remark is that the size of a machine satisfies the following inequality, for all n:

$$s(n) \leq 1 + \mathrm{Card}(A) + \mathrm{Card}(A^2) + \cdots + \mathrm{Card}(A^n) = \frac{\mathrm{Card}(A)^{n+1} - 1}{\mathrm{Card}(A) - 1}.$$

It follows that the maximal complexity of a language is exponential, and the online space complexity classes are relevant for functions smaller than exponential.

Definition 4 (Usual Online Space Complexity Classes).

- $\mathrm{Online}(Const)$ *is the class of languages of constant online space complexity, defined as* $\bigcup_{K\in\mathbb{N}} \mathrm{Online}(n \mapsto K)$.
- $\mathrm{Online}(Lin)$ *is the class of languages of linear online space complexity, defined as* $\bigcup_{a\in\mathbb{N}} \mathrm{Online}(n \mapsto an)$.
- $\mathrm{Online}(Quad)$ *is the class of languages of quadratic online space complexity, defined as* $\bigcup_{a\in\mathbb{N}} \mathrm{Online}(n \mapsto an^2)$.
- $\mathrm{Online}(Poly)$ *is the class of languages of polynomial online space complexity, defined as* $\bigcup_{k\in\mathbb{N}} \mathrm{Online}(n \mapsto n^k)$.

We define completeness with respect to a online space complexity class as follows.

Definition 5 (Completeness for Complexity Classes).
We say that L has linear online space complexity if:

- ***(upper bound)*** $L \in \mathrm{Online}(Lin)$*, i.e. there exists* $a \in \mathbb{N}$ *such that* $L \in \mathrm{Online}(n \mapsto an)$,
- ***(lower bound)*** *if* $L \in \mathrm{Online}(f)$*, then there exists* $a \in \mathbb{N}$ *such that for all* n, $f(n) \geq an$.

The definitions of L having constant, quadratic, polynomial or exponential complexity are similar.

We denote Reg the class of regular languages, *i.e.* those recognised by automata.

Theorem 1.

- Online($Const$) = Reg, *i.e. a language has constant online space complexity if, and only if, it is regular.*
- Online $\left(n \mapsto \frac{\mathrm{Card}(A)^{n+1}-1}{\mathrm{Card}(A)-1}\right)$ *contains all languages.*

The first item follows from the observation that deterministic automata are exactly machines with finitely many states. For the second item, consider a language L, we construct a machine recognising L of exponential size. Its set of states is A^*, the initial state is ε and the transition function is defined by $\delta(w, a) = wa$. The set of accepting states is simply L itself!

The languages defined in the following example will be studied later on to illustrate the lower bound techniques we give.

Example 1.

- Define $\textsc{Maj}_2 = \left\{w \in \{a, b\}^* \mid |w|_a > |w|_b\right\}$, the majority language over two letters. Here $|w|_a$ denotes the number of occurrences of the letter a in w. We construct a machine of linear size recognising $\textsc{Maj}_2$: its set of states is $\mathbb{Z}$, the integers, the letter a acts as $+1$ and the letter b as -1, and the set of accepting states is $\mathbb{N}$, the positive integers. After reading the word w, the state is $|w|_a - |w|_b$, implying that the machine has linear size. So $\textsc{Maj}_2 \in$ Online(Lin), and we will show in Subsect. 2.2 that this bound is tight: $\textsc{Maj}_2$ has linear online space complexity.
- Define $\textsc{Eq} = \left\{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c\right\}$. We construct a machine of quadratic size recognising Eq: its set of states is $\mathbb{Z}^2$, the letter a acts as $(+1, +1)$, the letter b as $(-1, 0)$, the letter c as $(0, -1)$, and the only accepting state is 0. After reading the word w, the state is $(|w|_a - |w|_b, |w|_a - |w|_c)$, implying that the machine has quadratic size. So $\textsc{Eq} \in$ Online($Quad$), and we will show in Subsect. 2.2 that this bound is tight: L_2 has quadratic online space complexity.
- Define $\textsc{Squares} = \{ww \mid w \in A^*\}$. We will show in Subsect. 2.3 that Squares has exponential online space complexity.

2.2 Lower Bounds Using Formal Language Theory

We present a first technique to give lower bounds on the online space complexity, relying on the notion of left quotients.

Let w be a finite word, define its left quotient with respect to L by

$$w^{-1}L = \{u \mid wu \in L\}.$$

A well known result from automata theory states that the existence of a minimal deterministic automaton, called the *syntactic automaton*, whose states of the left quotients.

This construction extends *mutatis mutandis* when dropping the assumption that the automaton has finitely many states, *i.e.* going from deterministic automata to machines as we defined them. The statement, however, is more involved, and gives precise lower bounds on the online space complexity of the language.

Formally, consider a language L, we define the syntactic machine of L, denoted $\mathcal{M}_L$, as follows. We define the set of states as the set of all left quotients: $\left\{w^{-1}L \mid w \in A^*\right\}$. The initial state is $\varepsilon^{-1}L$, and the transition function is defined by $\delta(w^{-1}L, a) = (wa)^{-1}L$. Finally, the set of accepting states is $\left\{w^{-1}L \mid w \in L\right\}$.

Denote s_L the size of the syntactic machine of L.

Theorem 2.

- *$\mathcal{M}_L$ recognises L, so $L \in \mathrm{Online}(s_L)$,*
- *for all f, if $L \in \mathrm{Online}(f)$, then $f \geq s_L$.*

Note that the implies the existence of a minimal function f such that $L \in \mathrm{Online}(f)$; in other words $L \in \mathrm{Online}(f)$ if, and only if, $f \geq s_L$. This was not clear a priori, because the order on functions is partial.

The first item is routinely proved, as in the case of automata. For the second item, assume towards contradiction that there exists f such that $L \in \mathrm{Online}(f)$ and $f \not\geq s_L$, *i.e.* there exists n such that $f(n) < s_L(n)$. Consider a machine $\mathcal{M}$ recognising L of size f. Since $f(n) < s_L(n)$, there exists two words u and v of length n such that $u^{-1}L \neq v^{-1}L$ but $c(u) = c(v)$, *i.e.* in $\mathcal{M}$ the words u and v lead to the same state. The left quotients $u^{-1}L \neq v^{-1}L$ being different, there exists a word w such that $uw \in L$ and $vw \notin L$, or the other way around. But $c(uw) = c(vw)$ since $\mathcal{M}$ is deterministic, so this state must be both accepting and rejecting, contradiction.

The view using left quotients is very powerful, as it gives the exact online space complexity of a language. However, it is sometimes hard to deal with, as it may involve complicated word combinatorics. We illustrate it on some of the examples introduced in Subsect. 2.1.

To prove for instance that L has an online space complexity at least linear using this technique, one has to exhibit, for infinitely many n, a family of linearly many words (*i.e.*, at least an words, for some constant a) of length at most n that induce pairwise distinct left quotients.

Example 2.

- Fix n. The words $a^{n+k}b^{n-k}$, for $0 \leq k \leq n$, have length $2n$ and all induce pairwise distinct left quotients, since $a^{n+k}b^{n-k} \cdot b^p \in \textsc{Maj}_2$ if, and only if, $p < 2k$. It follows from Theorem 2 that $\textsc{Maj}_2$ has linear online space complexity.
- Fix n. The words $a^{2n-p-q}b^pc^q$, for $0 \leq p, q \leq n$, have length $2n$ and all induce pairwise distinct left quotients, since $a^{2n-p-q}b^pc^q \cdot a^{k+\ell}b^{2n-k}c^{2n-\ell} \in \textsc{Eq}$ if, and only if, $k = p$ and $\ell = q$. It follows from Theorem 2 that $\textsc{Eq}$ has quadratic online space complexity.

2.3 Lower Bounds Using Communication Complexity

We present a second technique to give lower bounds on the online space complexity. This is inspired by the field of streaming algorithms, which made several use of this idea.

Rather than giving a generic lower bound technique, we illustrate the ideas by giving an exponential lower bound for SQUARES, the language of squares, *i.e.* words of the form ww for some word w. Note that we are here using a very big hammer for a very simple result; the first technique using left quotients would very easily give an exponential lower bound.

We consider the following communication problem: Alice receives a word u of length n, and Bob another word v of length n. The goal for Bob is to determine whether $u = v$, with the least amount of communication between them. It is well known that no protocol can solve this problem using less than n bits of communication; the optimal protocol is simply for Alice to send her whole input u to Bob.

The idea now is to use a machine recognising SQUARES to construct a problem solving the above communication problem, thereby obtaining lower bounds on the size of such a machine. Denote s the size of the machine, and fix n. Consideer an input of length $2n$, which we denote $w = uv$ where u and v have length n. We construct the following protocol: Alice runs the machine on her input u, and communicates to Bob the state reached. Bob takes over from there, running the machine on his input, starting from the state sent by Alice. The last state he obtains determines whether the whole input, *i.e.* w, belongs to SQUARES, or equivalently whether $u = v$.

Now, to communicate the state reached after reading u, Alice only needs $\log(s(n))$ (indeed, if there are $s(n)$ different states, then they can be all described using $\log(s(n))$ bits). The lower bound on the communication problem implies than $\log(s(n)) \geq n$, *i.e.* $s(n) \geq 2^n$. It follows that SQUARES has exponential online space complexity.

2.4 Comparison to Circuit Complexity

We conclude this section by observing that the examples studied above show that online space complexity and circuit complexity are orthogonal. Indeed:

- The language MAJ has linear online space complexity (*small*), but does not belong to AC^0, *i.e.* it has a rather big circuit complexity. Another example of a language having a small online space complexity and a big circuit complexity is Parity: it is regular, and recognised by a machine of size 2, but does not belong to AC^0.
- The language SQUARES has exponential online space complexity (*large*), but it has a very small circuit complexity: it is recognised by a family of circuits of linear size of constant depth.

3 The Online Space Complexity of Probabilistic Automata

In his seminal paper introducing probabilistic automata [7], Rabin devotes a section to "approximate calculation of matrix products", which is related to, and inspired, online space complexity. In the end of this section, Rabin states a result, without proof; the aim of this section is to substantiate this claim, *i.e.* formalising and proving the result.

We start by defining probabilistic automata, state Rabin's claim, and prove that it holds true.

3.1 Probabilistic Automata

Let Q be a finite set of states. A distribution over Q is a function $\delta : Q \to [0, 1]$ such that $\sum_{q \in Q} \delta(q) = 1$. We denote $\mathcal{D}(Q)$ the set of distributions over Q.

Definition 6 (Probabilistic Automaton). *A probabilistic automaton $\mathcal{A}$ is given by a finite set of states Q, a transition function $\phi : A \to (Q \to \mathcal{D}(Q))$, an initial state $q_0 \in Q$, and a set of final states $F \subseteq Q$.*

In a transition function ϕ, the quantity $\phi(a)(s, t)$ is the probability to go from the state $s \in Q$ to the state $t \in Q$ reading the letter a. A transition function naturally induces a morphism $\phi : A^* \to (Q \to \mathcal{D}(Q))$. We denote $\mathbb{P}_{\mathcal{A}}(s \xrightarrow{w} t)$ the probability to go from a state s to a state t reading w on the automaton $\mathcal{A}$, *i.e.* $\phi(w)(s, t)$.

The *acceptance probability* of a word $w \in A^*$ by $\mathcal{A}$ is $\sum_{t \in F} \phi(w)(q_0, t)$, which we denote $\mathbb{P}_{\mathcal{A}}(w)$.

The following threshold semantics was introduced by Rabin [7].

Definition 7 (Probabilistic Language). *Let $\mathcal{A}$ be a probabilistic automaton, it induces the probabilistic language*

$$L^{>\frac{1}{2}}(\mathcal{A}) = \left\{ w \in A^* \mid \mathbb{P}_{\mathcal{A}}(w) > \frac{1}{2} \right\}.$$

3.2 Substantiating the Claim

In a section called "approximate calculation of matrix products" in the paper introducing probabilistic automata [7], Rabin asks the following question: is it possible, given a probabilistic automaton, to construct an algorithm which reads words and compute the acceptance probability in an online fashion? He then shows that this is possible under some restrictions on the probabilistic automaton, and concludes the section by stating that "*an example due to R. E. Stearns shows that without assumptions, a computational procedure need not exist*". The example is not given, and to the best of the author's knowledge, has never been published anywhere.

In this section, we substantiate this claim, in the framework of online space complexity as we defined it. Note that the formalisation of Rabin's claim is subject to discussions, as for instance Rabin asks whether the acceptance probability can be computed up to a given precision; in our setting, the acceptance probability is not actually computed, but only compared to a fixed threshold, following Rabin's definition of probabilistic languages.

The following result shows that there exists a probabilistic automaton defining a language of maximal (exponential) online space complexity.

Theorem 3. *There exists a probabilistic automaton $\mathcal{A}$ such that $L^{>\frac{1}{2}}(\mathcal{A})$ has exponential online space complexity.*

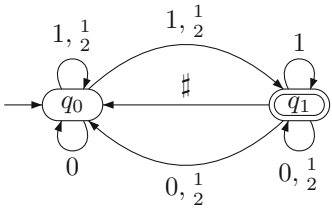

Fig. 1. The initial state is marked by an ingoing arrow and the accepting state by an outgoing arrow. The first symbol over a transition is a letter (either 0, 1, or ♯). The second symbol (if given) is the probability of this transition. If there is only one symbol, then the probability of the transition is 1.

In the original paper introducing probabilistic automata, Rabin [7] gave an example of a probabilistic automaton $\mathcal{A}$ computing the binary decomposition function (over the alphabet $\{0,1\}$), denoted bin, *i.e.* $\mathbb{P}_{\mathcal{A}}(u) = \mathrm{bin}(u)$, defined by

$$\mathrm{bin}(a_1 \dots a_n) = \frac{a_1}{2^n} + \dots + \frac{a_n}{2^1}$$

(*i.e.* $0.a_n \dots a_1$ in binary). We show that adding one letter and one transition to this probabilistic automaton gives an automaton with exponential online space complexity. This example appeared in [2].

The automaton $\mathcal{A}$ is represented in Fig. 1. The alphabet is $A = \{0, 1, \sharp\}$. The only difference between the automaton proposed by Rabin [7] and this one is the transition over $\sharp$ from q_1 to q_0. As observed by Rabin, a simple induction shows that for u in $\{0,1\}^*$, we have $\mathbb{P}_{\mathcal{A}}(u) = \mathrm{bin}(u)$.

Let $w \in A^*$, it decomposes uniquely into $w = u_1 \sharp u_2 \sharp \cdots \sharp u_k$, where $u_i \in \{0,1\}^*$. Observe that $\mathbb{P}_{\mathcal{A}}(w) = \mathrm{bin}(u_1) \cdot \mathrm{bin}(u_2) \cdots \mathrm{bin}(u_k)$.

Consider a machine recognising $L^{>\frac{1}{2}}(\mathcal{A})$ and fix n. The binary decomposition function maps words of length n to rationals of the form $\frac{a}{2^n}$, for $0 \leq a < 2^n$. Consider two words u and v in $\{0,1\}^*$ of length n, we show that $(u1)^{-1}L^{>\frac{1}{2}}(\mathcal{A}) \neq (v1)^{-1}L^{>\frac{1}{2}}(\mathcal{A})$.

Without loss of generality assume $\mathrm{bin}(u1) < \mathrm{bin}(v1)$; observe that $\frac{1}{2} \leq \mathrm{bin}(u1) < \mathrm{bin}(v1)$. There exists w in $\{0,1\}^*$ such that $\mathrm{bin}(u1) \cdot \mathrm{bin}(w) < \frac{1}{2}$ and $\mathrm{bin}(v1) \cdot \mathrm{bin}(w) > \frac{1}{2}$: it suffices to choose w such that $\mathrm{bin}(w)$ is in $\left(\frac{1}{2\mathrm{bin}(v1)}, \frac{1}{2\mathrm{bin}(u1)}\right)$, which exists by density of the dyadic numbers in $(0,1)$. Thus, $(u1)^{-1}L^{>\frac{1}{2}}(\mathcal{A}) \neq (v1)^{-1}L^{>\frac{1}{2}}(\mathcal{A})$, and we exhibited exponential many words having pairwise distinct left quotients. It follows from Theorem 2 that $L^{>\frac{1}{2}}(\mathcal{A})$ has exponential online space complexity.

4 Conclusion

We introduced a new complexity measure called the online space complexity, quantifying how hard it is to solve a problem when the input is given in an online fashion, focusing on the space consumption.

We considered the online space complexity of probabilistic automata, as hinted by Rabin in [7], and showed that probabilistic automata give rise to languages of high (maximal) online space complexity.

We mention some directions for future research about online space complexity.

The first is to give characterisations of the natural online space complexity classes (linear, quadratic, polynomial). Such characterisations could be in terms of logics, as it is done in descriptive complexity, or algebraic, as it is done in the automata theory. The canonical example is languages of constant online space complexity, which are exactly regular languages, defined by monadic second-order logic.

A second direction would be to extend the framework of online space complexity to quantitative queries. Indeed, we defined here the online space complexity of a language, *i.e.* of qualitative queries: a word is either inside, or outside the language.

A third intriguing question is the existence of a dichotomy for the online space complexity of probabilistic automata. Is the following conjecture true: for every probabilistic language, its online space complexity is either polynomial or exponential?

References

1. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: STOC 1996, pp. 20–29 (1996). http://doi.acm.org/10.1145/237814.237823
2. Fijalkow, N., Skrzypczak, M.: Irregular behaviours for probabilistic automata. In: RP 2015, pp. 33–36 (2015)
3. Flajolet, P., Martin, G.N.: Probabilistic counting algorithms for data base applications. J. Comput. Syst. Sci. **31**(2), 182–209 (1985). http://dx.doi.org/10.1016/0022-0000(85)90041-8
4. Karp, R.M.: On-line algorithms versus off-line algorithms: how much is it worth to know the future? In: IFIP 1992, pp. 416–429 (1992)

5. Munro, J.I., Paterson, M.: Selection and sorting with limited storage. Theor. Comput. Sci. **12**, 315–323 (1980). http://dx.doi.org/10.1016/0304-3975(80)90061-4
6. Patnaik, S., Immerman, N.: Dyn-FO: A parallel, dynamic complexity class. In: PODS 1994, pp. 210–221 (1994). http://doi.acm.org/10.1145/182591.182614
7. Rabin, M.O.: Probabilistic automata. Inf. Control **6**(3), 230–245 (1963). http://dx.doi.org/10.1016/S0019-9958(63)90290-0
8. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985). http://doi.acm.org/10.1145/2786.2793