Fundamenta Informaticae XX (2017) 1–13
DOI 10.3233/FI-2016-0000
IOS Press

On Tight Separation for Blum Measures Applied to Turing Machine Buffer Complexity

Jiří Šíma[*]
Institute of Computer Science
The Czech Academy of Sciences
sima@cs.cas.cz

Stanislav Žák[†]
Institute of Computer Science
The Czech Academy of Sciences
stan@cs.cas.cz

Abstract. We formulate a very general tight diagonalization method for the Blum complexity measures satisfying two additional axioms related to our diagonalizer machine. We apply this method to two new, mutually related, distance and buffer complexities of Turing machine computations which are important nontrivial examples of natural Blum complexity measures different from time and space. In particular, these measures capture how many times the worktape head needs to move a certain distance during the computation which corresponds to the number of necessary block uploads into a buffer cache memory. We start this study by proving a tight separation which shows that a very small increase in the distance or buffer complexity bound (roughly from $f(n)$ to $f(n+1)$) brings provably more computational power to both deterministic and nondeterministic Turing machines even for unary languages. We also obtain hierarchies of the distance and buffer complexity classes.

[*]Research was partially supported by project GA ČR P202/12/G061 and RVO: 67985807. The presentation of this paper benefited from valuable comments of an anonymous reviewer.
[†]Research was partially supported by project GA ČR P202/10/1333 and RVO: 67985807.
Address for correspondence: Jiří Šíma, Institute of Computer Science, The Czech Academy of Sciences, P.O. Box 5, 182 07 Prague 8, Czech Republic

1. Introduction

The theory of computational complexity is one of the major attempts to understand the phenomenon of computation. One of the key tasks of the theory is to find out how an increase or decrease of limits set on the computational resources can influence the computational power of different types of computational devices. In history, the efforts to answer questions of this type led to a long sequence of various separation and hierarchy results for particular computational devices and complexity measures, e.g. chronologically [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12].

The present paper follows this direction of research. We formulate a very general tight diagonalization method issuing from delayed diagonalization [9], which works for the Blum measures of computational complexity [13] satisfying two additional axioms related to our diagonalizer machine. We apply this method to two new nontrivial natural Blum measures, namely the so-called *distance* and *buffer* complexities, which are introduced for both deterministic and nondeterministic Turing machines (TM) with one worktape [14]. These measures capture how many times the worktape head needs to move a certain distance during the computation.

In particular, any computation by a TM is characterized, among others, by a sequence of worktape head positions $(h_t)_{t\geq 0}$ where h_t is the location of the head on the worktape just after t computational steps (i.e. at time instant t). The distance complexity d^δ is the minimum length of its so-called δ-distance subsequence $(h_{t_i})_{i\geq 0}$ which, starting with $t_0 = 0$, is defined inductively as follows. The next t_{i+1} is the first time instant at which the worktape head is exactly at distance $\delta(n)$ (measured in the number of tape cells) from any of its previous locations visited since time t_i. Note that parameter $\delta(n) > 0$ depends on the input length n where δ is a recursive function on the set of natural numbers.

The buffer complexity b^δ is defined similarly by using a so-called δ-buffer subsequence such that the distance in the definition of t_{i+1} is measured from the previous member h_{t_i} of this subsequence. The δ-buffer subsequence thus partitions the worktape into disjoint blocks of size $\delta(n)$. The buffer complexity measures how many times the worktape head crosses the border between two such neighboring blocks on the worktape when multiple immediately subsequent crossings through the same border are counted only once. This is a natural model of the number of times a buffer cache memory needs to be read/written during a computation.

In particular, consider the control unit has an imaginary cache buffer memory for the worktape whose capacity is two blocks of length $\delta(|u|)$ so that the worktape head location has to be within this buffer. This means that the block from the worktape next to the right is uploaded into the buffer memory when the head leaves the buffer to the right, and the buffer contents are shifted to the left so that the head finds itself in the midpoint of the buffer while the block on the left is stored to the worktape. This is reminiscent of a super-head reading the whole block as a super-cell of length $\delta(|u|)$ and moving to the right. Similarly, the buffer window moves to the left when the head is about to leave the buffer memory to the left. Within this framework, the buffer complexity measures the number of necessary block uploads into the cache buffer memory.

Furthermore, the buffer complexity proves to be related to the distance measure by $d^{2\delta} \leq b^\delta \leq d^\delta$. In this way, the buffer or distance measure can be used for investigating the buffering aspects of Turing computations. Thus, they represent important nontrivial examples of natural Blum complexity measures different from time and space. In fact, these new measures are bounded by space S and time T as $\lfloor S/\delta \rfloor \leq d^\delta \leq \lceil T/\delta \rceil$, $\lfloor S/\delta \rfloor \leq b^\delta \leq \lceil T/\delta \rceil$. For example, this means that for a constant function δ, the buffer measure coincides with the time complexity by the linear speedup theorem at the cost of multi-symbol worktape alphabet, or the space complexity is $O(n \log n)$ for linear buffer complexity with logarithmic δ etc. In addition, the worktape cells that are close to each other, can be reused during a computation without increasing the distance and buffer complexity. This is one of the two properties required in our diagonalizer machine, which is shared with the space measure but not with the time complexity, although the diagonalizer can also be adapted for measures that break this condition including the time complexity.

We start our study by separation and hierarchy results for the distance and buffer complexity which are surprisingly very tight. This indicates that these new measures are suitable tools for classifying the Turing computations with respect to their buffering complexity. The tightness in the results requires that the worktape alphabet is fixed and the measure is not applied to the TM computations directly but instead to their simulations on a fixed universal Turing machine, which rules out Blum's speedup theorem.

The results are of the form that an increase of only one in the argument of a given complexity bound (and of parameter δ plus an additive constant) leads to a strictly greater computational power. For example, an additive constant increase in the linear complexity bound is sufficient to gain more power. For the hierarchies of full languages[1], the increase in the complexity bound is required to be slightly larger. The main tool of the proof is a delayed diagonalization method derived from [9] which is newly formulated for suitable Blum computational complexity measures. The results are valid even for unary languages, which strengthens a preliminary version of this paper [15] containing the proofs only for a binary alphabet.

The paper is organized as follows. After a brief review of basic definitions regarding Turing machines and complexity measures in Section 2, we define a diagonalizer and prove a general diagonalization theorem for Blum complexity measures in Section 3. In Section 4, the distance and buffer complexity measures are introduced and related to each other. The separation for these measures is proven in Section 5 while the corresponding hierarchies are presented in Section 6. The results are summarized in Section 7 where possible further research directions are outlined.

2. Preliminaries

By a *language* L we mean any set of words over a fixed binary alphabet, that is, $L \subseteq \{0,1\}^*$. We will formulate our results for *unary languages* over a one-symbol alphabet for which we assume $L \subseteq \{1\}^*$. In addition, we say that two languages L and L' are equivalent, that is, $L \sim L'$ iff they differ only in a finite number of words. For a class C of languages we define $\mathcal{E}(C) =_{\mathrm{df}} \{L' \mid (\exists L \in C)\, L \sim L'\}$. Clearly, $L \notin \mathcal{E}(C)$ implies that L differs from any language of $\mathcal{E}(C)$ on infinitely many words.

By a *Turing machine* we mean any finite-control machine with two-way read-only input tape and with one semi-infinite worktape (infinite to the right) with alphabet $\{0,1,b\}$ (b stands for blank) and endmarker # at the left end side (at the 0th cell of the worktape), allowing for both the deterministic or nondeterministic versions [14].

[1] A language accepted by a TM is full if all its words are accepted within a given complexity bound.

The *programs* are words from $\{0,1\}^+$. If p is a program, then M_p is a corresponding machine which implements p. For any machine M, we denote by $L(M)$ the language accepted by M, and define $L_p = L(M_p)$. By p_M we mean a program of M. For any input word u, a *universal machine* starts its computation with some program p stored at the left end side of the worktape and it simulates machine M_p on u.

By a *complexity measure* we mean any (partial) mapping $c : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to \mathbb{N} \cup \{\infty\}$ where $\mathbb{N}$ denotes the set of natural numbers (including 0). Informally, $c(p,u,v)$ is intended to measure the complexity of computation by machine M_p on input word u starting with v stored at the left end side of the worktape. In general, we assume that c satisfies **Blum axioms** [13]: $c(p,u,v) < \infty$ iff M_p with initial worktape contents v halts on input u, and there is a Turing machine which, given $p,u,v \in \{0,1\}^*$ and $m \in \mathbb{N}$, decides whether $c(p,u,v) = m$. We also use notation $c(p,u) = c(p,u,\varepsilon)$ when ε is the empty word.

In Section 4, we will introduce the complexity measures whose definitions depend parametrically on the input length. This means that complexity measure c is in fact defined as a (uniform) sequence of complexity measures $(c^n)_{n=0}^{\infty}$ where c^n is employed for input words of length n, that is, $c(p,u) = c^{|u|}(p,u)$ where $|u|$ is the length of word u. For such c we denote by c' the complexity measure specified by a *shifted* sequence $(c^{n+1})_{n=0}^{\infty}$ so that $c'(p,u) = c^{|u|+1}(p,u)$ for any input word $u \in \{0,1\}^*$. For nonparametric measures, c' and c coincide.

Let $P \subseteq \{0,1\}^+$ be a recursively enumerable set of programs of the machines in question (e.g. nondeterministic Turing machines), that is, there exists a Turing machine which will enumerate all programs of P. For any $p \in P$, for any complexity measure c and for any complexity bound $f : \mathbb{N} \to \mathbb{N}$, we define language $L_p(c,f) =_{\rm df} \{u \in L_p \,|\, c(p,u) \le f(|u|)\}$ to be a set of words accepted by machine M_p within the complexity bound f measured by c. The *complexity class* $c(f) = \{L_p(c,f) \,|\, p \in P\}$ contains all such languages for $p \in P$, including those $L_p(c,f) \subsetneq L_p$ which are pruned by the complexity bound, while the respective complexity class of *full languages* satisfying $L_p(c,f) = L_p$, is denoted by a corresponding capital letter as $C(f) = \{L_p \,|\, L_p = L_p(c,f)\,,\, p \in P\}$. Obviously, $C(f) \subseteq c(f)$.

For the purpose of a tight separation of complexity classes we will measure the complexity of a computation by machine M as the complexity of simulating p_M on a fixed universal machine U. In particular, for any complexity measure c we define its *universal* version as $c_U(p,u) =_{\rm df} c(p_U,u,p)$.

In the notation of languages and complexity classes, $f(n+1)$ stands for the complexity bound $f' : \mathbb{N} \to \mathbb{N}$ such that $f'(n) = f(n+1)$ for every $n \in \mathbb{N}$. This means that $f(n+1)$ can denote a function on natural numbers (not just an image of $n+1$ under f) within the context where a function symbol is expected. Notice that the inclusion $c(f) \subseteq c(f(n+1))$ need not, in general, be true even for nondecreasing f since some of the languages $L_p(c,f)$ cut out of L_p by complexity bound f cannot be delimited by $f(n+1)$. Recall that complexity bound f is recursive if there is a Turing machine that computes f. Hereafter, $\log : \mathbb{N} \to \mathbb{N}$ will always mean the logarithmic function of base 2 restricted to natural numbers whose value $\log n = \lceil \log_2 n \rceil$ is rounded to the smallest following integer for any $n \in \mathbb{N}$.

3. The Diagonalization Theorem

Our delayed diagonalization method issuing from [9] is based on a *diagonalizer* for Blum complexity measure c and its recursive bound $f : \mathbb{N} \to \mathbb{N}$, which is a Turing machine Δ working on unary inputs $u \in \{1\}^*$. We will below describe its program p_Δ (for sufficiently large $n = |u|$) by using a high-level pseudocode. The program is composed of two main parts: precomputation A and simulation S. The precomputation A is formally defined as an infinite process which is terminated when the worktape head of Δ leaves a delimited working space.

```
***Diagonalizer*** $\Delta$

input $u$

**Precomputation** $A$

    check whether the input $u$ is of the form $1^n$ and construct a working space $W_n$ of length
    $|W_n| = \log n$ on the worktape (e.g. using a binary counter)

    **for every** $i = 1, 2, 3, \ldots$ **until** space $W_n$ suffices **do**

      **Phase** $G$**:** on the untapped part of $W_n$, generate the next element $p_i$ of a sequence $(p_i)_{i=1}^{\infty}$
          which contains each program $p \in P$ infinitely many times

      **Phase** $T$**:** using only $W_n$, test whether $u_i =_{\text{df}} 1^{2^{s_i-1}+1} \overset{?}{\in} L_{p_i}(c_U, f)$ where $s_i \in \mathbb{N}$ is the
          length of the used worktape space after phase $G$

    **enddo**

**Simulation** $S$   $\{$ *last generated* $p_i$ *is on the worktape* $\}$

    **if** $A$ terminates during its phase $G$ without generating $p_{i+1}$ **then**
      accept $u$ iff $u_i \notin L_{p_i}(c_U, f)$
    **else** $\{$ *$A$ terminates during its phase $T$ without deciding* $u_i \overset{?}{\in} L_{p_i}(c_U, f)$ $\}$
      simulate machine $M_{p_i}$ on input $u1$ as universal machine $U$ would do
```

Note that the leaving condition which breaks the precomputation A when the logarithmic working space W_n on the worktape is consumed can, in fact, be replaced by any suitable condition which can be simply tested so that the longer the input word to Δ, the more computation of A is performed. In general, $u_i \in \{1\}^*$ must be the shortest input to Δ for which A generates p_i (within space $|W_{|u_i|}| = s_i$). Hence, our choice of $|W_n| = \log n$ leads to the definition of $u_i = 1^{2^{s_i-1}+1}$ since $2^{s_i-1} + 1$ is the minimal n_i such that $\log n_i = s_i$. On the other hand, there is no sublogarithmic increasing function that is fully space constructible by deterministic [16] and nondeterministic [17] Turing machines and could thus be exploited for delimiting working space W_n.

Furthermore, we define function $z : \mathbb{N} \to \mathbb{N}$ as $z(i)$ is the minimum $j \in \mathbb{N}$ such that A finishes its phase T on input $u_i 1^j$ and decides whether $u_i \overset{?}{\in} L_{p_i}(c_U, f)$. Thus, $z(i)$ corresponds to a sufficient delay when Δ diagonalizes against machine M_{p_i}. Denote $R_i = \{u_i 1^j \mid 0 \leq j < z(i)\}$ for any $i \in \mathbb{N}$, which are pairwise disjoint sets since $|u_i 1^{z(i)}| < |u_{i+1}|$ for every $i \in \mathbb{N}$, as the next p_{i+1} is generated on the untapped part of worktape space W_n during phase G. The union $R = \bigcup_{i \geq 1} R_i$ contains the inputs on which Δ performs the simulation in S, while the diagonalization is delayed. Now we are ready to introduce our diagonalization theorem which proves a tight separation for suitable Blum complexity measures that meet two additional axioms related to Δ:

Theorem 3.1. Let Δ be a diagonalizer for Blum complexity measure c and its recursive bound $f : \mathbb{N} \to \mathbb{N}$ which satisfy the following two conditions:

1. for any $u \in R_i$, $c'(p_\Delta, u) = c_U(p_i, u1)$
2. for any $u \notin R$, $c'(p_\Delta, u) \leq f(|u| + 1)$.

Then $L =_{\text{df}} L_{p_\Delta}(c', f(n+1)) \in c'(f(n+1)) \setminus \mathcal{E}(c_U(f))$.

Proof:
On the contrary, suppose that $L \in \mathcal{E}(c_U(f))$. Hence, $L \sim L_p(c_U, f)$ for some $p \in P$. By the definition of diagonalizer Δ we know that p occurs in the sequence $(p_i)_{i=1}^{\infty}$ generated during phase G of precomputation A infinitely many times. Thus, there is $i \in \mathbb{N}$ such that $L_{p_i}(c_U, f)$ differs from L only on words shorter than u_i.

For any input $u_i 1^j \in R_i$, the diagonalization delays so that $u_i 1^j \in L$ iff $u_i 1^j \in L_{p_\Delta}$ & $c'(p_\Delta, u_i 1^j) \leq f(|u_i 1^j| + 1)$ (by definition of L) iff $u_i 1^j \in L_{p_\Delta}$ & $c_U(p_i, u_i 1^{j+1}) \leq f(|u_i 1^{j+1}|)$ (by condition 1) iff $u_i 1^{j+1} \in L_{p_i}(c_U, f)$ (by definition of Δ) iff $u_i 1^{j+1} \in L$ since $L_{p_i}(c_U, f)$ differs from L only on words shorter than u_i. Hence, $u_i \in L$ iff $u_i 1^{z(i)} \in L$ iff $u_i 1^{z(i)} \in L_{p_\Delta}$ (as for $u = u_i 1^{z(i)} \notin R$ we know $c'(p_\Delta, u) \leq f(|u| + 1)$ by condition 2) iff $u_i \notin L_{p_i}(c_U, f)$ (by definition of Δ) iff $u_i \notin L$, which is a contradiction, and thus $L \notin \mathcal{E}(c_U(f))$. □

Note that condition 1 in Theorem 3.1 is naturally satisfied for Blum measures that are related to the space complexity. Clearly, the logarithmic space consumed within precomputation A is hidden in the space complexity of simulation S, which implies condition 1. For example, this is not true for the time complexity of Δ which is a sum of the times needed for performing A and S, respectively. In this case, the diagonalizer itself must control the time of simulation S [9]. Thus, for Blum complexity measures that do not meet condition 1 one can possibly modify Δ in part S so that Δ simulates M_{p_i} by U on input $u1$ and accepts iff U accepts within complexity bound $f(|u1|)$. In addition, the witness language L is defined as the full language L_{p_Δ}. Hence, for any $u_i 1^j \in R_i$, we obtain that $u_i 1^j \in L =_{\text{df}} L_{p_\Delta}$ iff $u_i 1^{j+1} \in L_{p_i}$ & $c_U(p_i, u_i 1^{j+1}) \leq f(|u_i 1^{j+1}|)$ by this modified definitions of Δ and L which compensate for condition 1 in the proof of Theorem 3.1 accordingly. On the other hand, Theorem 3.1 and its adaptation for the time measure yield exactly the known results when applied to the space [18] and time [9] complexity, respectively.

The diagonalization Theorem 3.1 provides the witness language L which is unary and is thus outside the respective complexity class of languages over any multi-symbol alphabet. This strengthens our previous simpler diagonalizer Δ' working on binary inputs which was presented in a preliminary version of this paper [15]. In the binary case, the unary code 1^k of program $p_i \in P$ where p_i is the binary expansion of $k > 0$, is a part of input $1^k 0^l 1^j$ to Δ', which is extended with the padding string 1^j that enlarges the working space W_n whose length $|W_n| = \log n$ grows with the input length $n = k + l + j$. The binary alphabet is here essential for separating the program code 1^k from the padding string 1^j by 0^l where $l > 0$. Then Δ' simply constructs program p_i from its code 1^k on the worktape in phase G instead of generating the sequence of programs up to p_i, which avoids the for-loop repeating the phases G and T in the unary diagonalizer Δ.

4. The Distance and Buffer Measures

We introduce two new nontrivial computational complexity measures which capture how many times the worktape head of a Turing machine needs to move a certain distance during the computation. For any computation by Turing machine M, denote by $h_t \in \mathbb{N}$ the head position on the worktape just after t computational steps by M (i.e. at time instant t) which equals the distance (in the number of tape cells) from the current worktape head location to the leftmost worktape cell with endmarker # whose position is thus zero (e.g. $h_0 = 1$). This defines a sequence $h_0, h_1, h_2, \ldots$ of the worktape head positions which is *finite* for halting computations and satisfies $|h_{t+1} - h_t| \leq 1$ for any $t \geq 0$. For any positive recursive function $\delta : \mathbb{N} \to \mathbb{N}$ (i.e. $\delta > 0$ which means $\delta(n) > 0$ for every $n \in \mathbb{N}$), we inductively define its so-called *δ-distance subsequence* $h_{t_0}, h_{t_1}, h_{t_2}, \ldots$ for a computation on input word u as follows:

1. $t_0 = 0$

2. $t_{i+1} = \min\{t \,|\, (\exists\, t')\ t_i \leq t' < t\ \ \&\ \ |h_t - h_{t'}| = \delta(|u|)\}$.

In other words, if we take into account only the worktape positions $h_{t_i}, h_{t_i+1}, h_{t_i+2}, \ldots$ visited by the head after t_i computational steps, then t_{i+1} is the first time instant at which the worktape head is exactly at distance $\delta(|u|)$ from some of these previous locations. Similarly, a so-called *δ-buffer subsequence* is defined when condition 2 above is replaced by

2'. $t_{i+1} = \min\{t \,|\, t > t_i\ \ \&\ \ |h_t - h_{t_i}| = \delta(|u|)\}$

in which t' is restricted to t_i, and hence $\delta(|u|)$ divides $h_{t_i} - 1$ for any $i \in \mathbb{N}$. This means that the δ-buffer subsequence partitions the worktape into blocks of length $\delta(|u|)$.

For any program $p \in P$, any input word u, and any initial worktape contents v, we define the *distance complexity* $d^\delta(p, u, v)$ to be the minimum length of δ-distance subsequence over all halting computations of M_p on input u, starting with v on the worktape, where δ is a parameter of the distance measure depending on the input length. In addition, we define formally $d^\delta(p, u, v) = \infty$ if M_p does not halt on u. Recall the short notation $d^\delta(p, u) = d^\delta(p, u, \varepsilon)$ for the empty word ε. Similarly, the *buffer complexity* measure b^δ is defined by using δ-buffer subsequences. Obviously, the distance and buffer complexity measures satisfy the Blum axioms and $\lfloor S/\delta \rfloor \leq d^\delta \leq \lceil T/\delta \rceil$, $\lfloor S/\delta \rfloor \leq b^\delta \leq \lceil T/\delta \rceil$ where S and T is the conventional space and time complexity, respectively. Moreover, they can mutually be related as follows.

Lemma 4.1. Let $\delta : \mathbb{N} \to \mathbb{N}$ be a positive recursive function. For any program $p \in P$ and any input word u, the inequality $d^{2\delta}(p,u) \leq b^{\delta}(p,u) \leq d^{\delta}(p,u)$ holds.

Proof:
Let $(h_{t_i^1})_{i\geq 0}$, $(h_{t_i^2})_{i\geq 0}$, and $(h_{t_i'})_{i\geq 0}$ be the δ-distance, 2δ-distance, and δ-buffer subsequences, respectively, for a computation of p on input word u. It suffices to prove $t_i^1 \leq t_i' \leq t_i^2$ for any meaningful i (note that the subsequences are typically of different length). By definition we know $t_0^1 = t_0' = t_0^2 = 0$. On the contrary, let $j \geq 1$ be the minimum index such that $t_j^1 \leq t_j' < t_{j+1}' < t_{j+1}^1$ or $t_j' \leq t_j^2 < t_{j+1}^2 < t_{j+1}'$. Suppose first that $t_j^1 \leq t_j' < t_{j+1}' < t_{j+1}^1$. According to the definition of δ-distance subsequence, t_{j+1}^1 is the first time instant such that there is t' satisfying $t_j^1 \leq t' < t_{j+1}^1$ and $|h_{t_{j+1}^1} - h_{t'}| = \delta(|u|)$, but for $t' = t_j'$ we know $t_j^1 \leq t_j' < t_{j+1}^1$ and $|h_{t_{j+1}'} - h_{t_j'}| = \delta(|u|)$, which contradicts $t_{j+1}' < t_{j+1}^1$. Thus, assume $t_j' \leq t_j^2 < t_{j+1}^2 < t_{j+1}'$. According to the definition of δ-buffer subsequence, t_{j+1}' is the first time instant such that $t_{j+1}' > t_j'$ and $|h_{t_{j+1}'} - h_{t_j'}| = \delta(|u|)$, which implies $|h_{t_{j+1}^2} - h_{t_j'}| < \delta(|u|)$ and $|h_{t_j'} - h_{t_j^2}| < \delta(|u|)$. Hence, $2\delta(|u|) > |h_{t_{j+1}^2} - h_{t_j'}| + |h_{t_j'} - h_{t_j^2}| \geq |h_{t_{j+1}^2} - h_{t_j^2}| = 2\delta(|u|)$, which is a contradiction completing the argument. □

5. The Separation Result

We first show a lemma concerning the distance complexity of simulating a machine on the universal Turing machine. Recall that $\delta + k$ for a constant $k \in \mathbb{N}$, denotes the function $\delta' : \mathbb{N} \to \mathbb{N}$ defined as $\delta'(n) = \delta(n) + k$ for any $n \in \mathbb{N}$.

Lemma 5.1. Let U be a fixed universal machine, p_M be a program of machine M, and $\delta : \mathbb{N} \to \mathbb{N}$ be a positive recursive function. Then $d^{\delta}(p_M, u) = d_U^{\delta+|p_M|}(p_M, u)$ for any input word u.

Proof:
(Sketch) Machine M is simulated by universal Turing machine U so that its program p_M is being shifted along the worktape following the shifts of the head of M. The distance δ on the worktape of M is thus transformed to the distance $\delta + |p_M|$ on the worktape of U. □

A tight separation for the distance complexity measure is formulated in the following theorem. We point out that this is a quite strong result since a very small increase in the distance complexity bound f to $f(n+1)$ and in the distance δ to $\delta(n+1)$ plus a constant brings provably more computational power to both deterministic and nondeterministic Turing machines working on unary inputs. Recall from Section 2 that for a fixed universal machine U, d_U^{δ} denotes the universal version of the distance complexity measure d^{δ} (i.e. $d_U^{\delta}(p,u) = d^{\delta}(p_U, u, p)$ for any program $p \in P$ running on input u). In addition, remember that $\delta(n+1)$ can denote a function on natural numbers, and as such, it can be added or compared to other functions (e.g. $\delta(n+1) \geq \log$ means $\delta(n+1) \geq \log n$ for every $n \in \mathbb{N}$).

Theorem 5.2. Let U be a fixed universal machine. Assume $\delta : \mathbb{N} \to \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{N}$ are positive recursive functions, and $\delta(n+1) \geq \log$. Then $L = L_{p_\Delta}(d^{\delta(n+1)}, f(n+1)) \in d_U^{\delta(n+1)+|p_\Delta|}(f(n+1)) \setminus \mathcal{E}(d_U^{\delta}(f))$ where Δ is a diagonalizer for complexity measure d^{δ} and its bound f.

Proof:
We will employ Theorem 3.1 for the distance complexity d^δ and its bound f for which the two conditions have to be verified. For any input word $u = 1^n$ to Δ, the distance complexity $d^{\delta(n+1)}(p_A, u)$ of precomputation A equals 1 by the assumption of $\delta(n+1) \geq \log$ since the capacity of working space W_n is $\log n$. Thus, for input word $u \in R_i$, the distance complexity $d^{\delta(n+1)}(p_\Delta, u) = d^\delta_U(p_i, u1)$ is dominated by the complexity of simulation S, which secures condition 1 of Theorem 3.1. Condition 2 which has now the form $1 = d^{\delta(n+1)}(p_\Delta, u) \leq f(|u| + 1)$ for any $u \notin R$ follows from the assumption of $f > 0$. By applying Theorem 3.1, we obtain language $L = L_{p_\Delta}(d^{\delta(n+1)}, f(n+1)) \in d^{\delta(n+1)}(f(n+1))$ which satisfies $L \notin \mathcal{E}(d^\delta_U(f))$. Moreover, $L \in d_U^{\delta(n+1)+|p_\Delta|}(f(n+1))$ by Lemma 5.1, which completes the proof. □

Note that in order to secure condition 1 of Theorem 3.1, the assumption $\delta(n+1) \geq \log$ in Theorem 5.2 cannot be substantially weakened as we know the length of working space W_n for precomputation A in diagonalizer Δ cannot be delimited by a sublogarithmic function. In addition, it is obvious that Lemma 5.1 and Theorem 5.2 are also valid for the buffer complexity b^δ, which can be verified simply by replacing d with b in their statements and proofs.

6. The Hierarchies

In order to achieve a fine-grained hierarchy for the distance (or buffer) complexity we will employ the classes $D^\delta_U(f)$ of full languages (see Section 2 for the definition). Unlike $d^\delta_U(f)$, these classes $D^\delta_U(f)$ prove to be linearly ordered with respect to the inclusion for hierarchies of functions δ and f.

Lemma 6.1. Let U be a fixed universal machine. Assume $\delta_1 : \mathbb{N} \to \mathbb{N}$, $\delta_2 : \mathbb{N} \to \mathbb{N}$ are positive recursive functions, and $f_1 : \mathbb{N} \to \mathbb{N}$, $f_2 : \mathbb{N} \to \mathbb{N}$ are recursive complexity bounds. If $\delta_1 \leq \delta_2$ and $f_1 \leq f_2$, then $D^{\delta_1}_U(f_1) \subseteq D^{\delta_2}_U(f_2)$.

Proof:
Let $L_p \in D^{\delta_1}_U(f_1)$, that is, $L_p = L(M_p) = L_p(d^{\delta_1}_U, f_1)$ for some $p \in P$. We will prove that $L_p(d^{\delta_1}_U, f_1) \subseteq L_p(d^{\delta_2}_U, f_2)$ which implies $L_p = L_p(d^{\delta_2}_U, f_2)$, and consequently $L_p \in D^{\delta_2}_U(f_2)$.

Suppose that $u \in L_p(d^{\delta_1}_U, f_1)$. For $k \in \{1, 2\}$, let $h_{t^k_0}, h_{t^k_1}, h_{t^k_2}, \ldots$ be the δ_k-distance subsequence for a computation of U on the input word u starting with p on the worktape. We will prove that $t^1_i \leq t^2_i$ for any meaningful i. We know $t^1_0 = t^2_0 = 0$. On the contrary, let $j \geq 1$ be the minimum index such that $t^1_j \leq t^2_j < t^2_{j+1} < t^1_{j+1}$. This means that there is t' such that $t^2_j \leq t' < t^2_{j+1}$ and $|h_{t^2_{j+1}} - h_{t'}| = \delta_2(|u|) \geq \delta_1(|u|)$ by the definition of δ_2-distance subsequence, which contradicts the definition of δ_1-distance subsequence since $t^2_{j+1} < t^1_{j+1}$ was assumed. Thus, $t^1_i \leq t^2_i$, which implies $d^{\delta_2}_U(p, u) \leq d^{\delta_1}_U(p, u) \leq f_1(|u|) \leq f_2(|u|)$. Hence, $u \in L_p(d^{\delta_2}_U, f_2)$, which completes the argument for $D^{\delta_1}_U(f_1) \subseteq D^{\delta_2}_U(f_2)$. □

We will show that any language from the complexity class $d^\delta(f)$ is a full language for a slightly larger distance parameter δ' and complexity bound f'.

Lemma 6.2. Let $\delta : \mathbb{N} \to \mathbb{N}$ be a positive recursive function and $f : \mathbb{N} \to \mathbb{N}$ be a recursive complexity bound such that for any input word u, the binary representation of function values $\delta(|u|)$ and $f(|u|)$ can be computed by a Turing machine Γ so that $d^{\delta'}(p_\Gamma, u) \leq g_f^\delta(|u|)$ where $\delta' = \delta + 8\log\delta + 2\log f$, for some complexity bound $g_f^\delta : \mathbb{N} \to \mathbb{N}$. Then $L_p(d^\delta, f) \in D^{\delta'}(f + g_f^\delta)$ for any $p \in P$.

Proof:
Denote $L = L_p(d^\delta, f)$ and $M = M_p$. We will define machine M' that simulates M on any input word u and halts immediately before $d^\delta(p_M, u) > f(|u|)$, which implies $L(M') = L$. In addition, we will ensure that $L(M') \in D^{\delta'}(f + g_f^\delta)$, which gives desired $L \in D^{\delta'}(f + g_f^\delta)$. The main ideas of how M' computes are as follows. At the beginning, M' constructs on its worktape two segments of length $8\log\delta(|u|)$ and $2\log f(|u|)$, respectively, by using Γ. Then M' simulates M so that M' shifts the two segments along the worktape following the worktape head of M.

The first segment of length $8\log\delta(|u|)$ serves for identifying the time instant t_{i+1} corresponding to the next worktape head position $h_{t_{i+1}}$ from the δ-distance subsequence $h_{t_0}, h_{t_1}, h_{t_2}, \ldots$ for a computation of M on u. Recall that t_{i+1} is the first time instant t since t_i for which there is t' such that $t_i \leq t' < t$ and $|h_t - h_{t'}| = \delta(|u|)$. For this purpose, it suffices to keep and update the current head location h_τ, the current minimum and maximum head positions $h_{\min} = \min\{h_t \,|\, t_i \leq t \leq \tau\}$ and $h_{\max} = \max\{h_t \,|\, t_i \leq t \leq \tau\}$ since time instant t_i as the differences $h_\tau - h_{t_i}$, $|h_{\min} - h_{t_i}|$, and $|h_{\max} - h_{t_i}|$, respectively, which consumes $3\log\delta(|u|)$ worktape cells of the segment. Moreover, a test whether the current maximum distance $|h_{\min} - h_{t_i}| + |h_{\max} - h_{t_i}|$ equals $\delta(|u|)$ requires the value of $\delta(|u|)$ occupying additional $\log\delta(|u|)$ cells to be precomputed and shifted with the first segment. Similarly, the second segment of length $2\log f(|u|)$ serves for halting the computation after $f(|u|)$ members of the δ-distance subsequence. In particular, the value of $f(|u|)$ occupying $\log f(|u|)$ cells is computed at the beginning and decremented after each head position of the δ-distance subsequence is reached.

In fact, the implementation of the ideas above requires the full double length of the two segments since the worktape alphabet of M' is (besides blank) restricted to 0 and 1 according to our assumption on Turing machines. In particular, it suffices to replace each bit by a pair of bits. The first bit of this pair indicates "marked/non-marked", which is used e.g. for comparing two parts of segments, and the second one represents the value of the original bit. Hence, the length of the two segments follows, which guarantees $L(M') \in D^{\delta'}(f + g_f^\delta)$. $\square$

Now we are ready to prove the hierarchy theorem for the distance complexity classes of full languages.

Theorem 6.3. Let U be a fixed universal machine. Assume $\delta : \mathbb{N} \to \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{N}$ are positive nondecreasing recursive functions, and $\delta(n+1) \geq \log$. Define recursive functions $\delta' : \mathbb{N} \to \mathbb{N}$ and $f' : \mathbb{N} \to \mathbb{N}$ as

$$\delta' = \delta(n+1) + 8\log\delta(n+1) + 2\log f(n+1) \tag{1}$$

$$f' = f(n+1) + g_{f(n+1)}^{\delta(n+1)} \tag{2}$$

where $g_{f(n+1)}^{\delta(n+1)} : \mathbb{N} \to \mathbb{N}$ is a nondecreasing recursive complexity bound such that for any input word u, the binary representation of function values $\delta(|u|+1)$ and $f(|u|+1)$ can be computed by a Turing machine Γ so that $d^{\delta'}(p_\Gamma, u) \leq g_{f(n+1)}^{\delta(n+1)}(|u|)$. Then $D_U^\delta(f) \subsetneq D_U^{\delta'+|p_\Delta|}(f')$ where Δ is the diagonalizer for the distance complexity measure d^δ and its bound f.

Proof:
Since δ, f, and $g_{f(n+1)}^{\delta(n+1)}$ are nondecreasing functions we know that $\delta \leq \delta' + |p_\Delta|$ and $f \leq f'$ according to (1) and (2), respectively. Hence, $D_U^\delta(f) \subseteq D_U^{\delta'+|p_\Delta|}(f')$ follows from Lemma 6.1. Define $L = L_{p_\Delta}(d^{\delta(n+1)}, f(n+1))$. According to Theorem 5.2, we know $L \notin \mathcal{E}(d_U^\delta(f)) \supseteq \mathcal{E}(D_U^\delta(f))$ which gives $L \notin D_U^\delta(f)$. On the other hand, $L \in D^{\delta'}(f')$ by Lemma 6.2, which implies $L \in D_U^{\delta'+|p_\Delta|}(f')$ according to Lemma 5.1. This completes the proof of the theorem. □

For example, consider the case of $\delta(n) = \log n$ and $f(n) = n$ which gives $\delta' = 3\log(n+1) + 8\log\log(n+1)$. One can easily construct a Turing machine Γ which, given a unary input $u = 1^n$, computes the binary representation of $\log(n+1)$ and $n+1$ within the distance complexity $d^{\delta'}(p_\Gamma, u) = 1$, by using a binary counter. Hence, $f'(n) = n+2$, and by Theorem 6.3 we obtain $D_U^{\log}(n) \subsetneq D_U^{3\log(n+1)+8\log\log(n+1)+|p_\Delta|}(n+2)$ which can be rewritten as $D_U^{\log}(n) \subsetneq D_U^{4\log}(n+2)$ according to Lemma 6.1.

The argument is similar for the hierarchy of buffer complexity classes $B_U^\delta(f)$ of full languages, which is formulated in the following theorem.

Theorem 6.4. Let U be a fixed universal machine. Assume $\delta : \mathbb{N} \to \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{N}$ are positive nondecreasing recursive functions, and $\delta(n+1) \geq \log$. Define recursive functions $\delta' : \mathbb{N} \to \mathbb{N}$ and $f' : \mathbb{N} \to \mathbb{N}$ as

$$\delta' = \delta(n+1) + 4\log\delta(n+1) + 2\log f(n+1) \tag{3}$$

$$f' = f(n+1) + g_{f(n+1)}^{\delta(n+1)} \tag{4}$$

where $g_{f(n+1)}^{\delta(n+1)} : \mathbb{N} \to \mathbb{N}$ is a nondecreasing recursive complexity bound such that for any input word u, the binary representation of function values $\delta(|u|+1)$ and $f(|u|+1)$ can be computed by a Turing machine Γ so that $b^{\delta'}(p_\Gamma, u) \leq g_{f(n+1)}^{\delta(n+1)}(|u|)$. Then $B_U^\delta(f) \subsetneq B_U^{\delta'+|p_\Delta|}(f')$ where Δ is the diagonalizer for the buffer complexity measure b^δ and its bound f.

Proof:
The proof proceeds in the same way as in the case of the distance complexity. In particular, the proof of Theorem 6.3 is based on Theorem 5.2 and on Lemmas 5.1–6.2. We already know that the statements of Theorem 5.2 and Lemma 5.1, in which d is replaced with b, are also valid for the buffer complexity. The same applies to Lemma 6.1 in whose proof the time instant t' coincides with t_j^2. The only slight change appears in the proof of the buffer complexity version of Lemma 6.2. In the definition of machine M', the first segment serves for identifying the time instant t_{i+1} corresponding to the next worktape head position $h_{t_{i+1}}$ from the δ-buffer subsequence $h_{t_0}, h_{t_1}, h_{t_2}, \ldots$, which is the first time instant t such that $t > t_i$ and $|h_t - h_{t'}| = \delta(|u|)$. Thus, it suffices to keep and update only the current head location h_τ since time instant t_i as the difference $h_\tau - h_{t_i}$ which consumes $\log\delta(|u|)$ worktape cells of the segment. Hence, the first segment is of half length $4\log\delta(|u|)$ as compared to the distance complexity version, which appears in formula (3). □

7. Conclusions

In this paper we have introduced the new distance and buffer complexity measures for computations on Turing machines with one worktape. These measures can be used for investigating the buffering aspects of Turing computations and they thus represent important nontrivial examples of natural Blum complexity measures different from time and space. As a first step along this direction, we have proven quite strong separation and hierarchy results which are valid even for unary languages. Many questions concerning e.g. reductions, completeness, complexity classes, and relations to the time and space hierarchies remain open for further research. To name just one simple example: Is the buffer complexity class $B^{\sqrt{n}}(\sqrt{n})$ *strictly* included in SPACE(n)?

We have also formulated our diagonalization method for the Blum complexity measures satisfying two additional axioms related to our diagonalizer, which is interesting on its own. This general method yields the known separation results for the usual space [18] and time [9] complexity. Moreover, the present framework is also applicable to oracle Turing machines employing other complexity measures such as the number of oracle queries or the maximum/total length of queries [19]. Analogous theorems can possibly be proven for other types of machines including those with auxiliary pushdown or counter and/or for other common computational resources such as randomness, interaction etc., which certainly deserves deeper study.

References

[1] Trakhtenbrot BA. Turing computations with logarithmic delay. *Algebra i Logika*, 1964. **3**(4):33–48.

[2] Stearns RE, Hartmanis J, Lewis II PM. Hierarchies of Memory Limited Computations. In: Proceedings of the SWCT 1965 Sixth Annual Symposium on Switching Circuit Theory and Logical Design. 1965 pp. 179–190. doi:10.1109/FOCS.1965.11.

[3] Borodin A. Computational Complexity and the Existence of Complexity Gaps. *Journal of the ACM*, 1972. **19**(1):158–174. doi:10.1145/321679.321691.

[4] Cook SA. A Hierarchy for Nondeterministic Time Complexity. *Journal of Computer and System Sciences*, 1973. **7**(4):343–353. doi:10.1016/S0022-0000(73)80028-5.

[5] Ibarra OH. A Hierarchy Theorem for Polynomial-Space Recognition. *SIAM Journal on Computing*, 1974. **3**(3):184–187. doi:10.1137/0203014.

[6] Seiferas JI. Relating Refined Space Complexity Classes. *Journal of Computer and System Sciences*, 1977. **14**(1):100–129. doi:10.1016/S0022-0000(77)80042-1.

[7] Seiferas JI, Fischer MJ, Meyer AR. Separating Nondeterministic Time Complexity Classes. *Journal of the ACM*, 1978. **25**(1):146–167. doi:10.1145/322047.322061.

[8] Sudborough IH. Separating Tape Bounded Auxiliary Pushdown Automata Classes. In: Proceedings of the STOC'77 Ninth Annual ACM Symposium on Theory of Computing. 1977 pp. 208–217. doi: 10.1145/800105.803410.

[9] Žák S. A Turing Machine Time Hierarchy. *Theoretical Computer Science*, 1983. **26**:327–333. doi: 10.1016/0304-3975(83)90015-4.

[10] Allender E, Beigel R, Hertrampf U, Homer S. Almost-Everywhere Complexity Hierarchies for Nondeterministic Time. *Theoretical Computer Science*, 1993. **115**(2):225–241. doi:10.1016/0304-3975(93)90117-C.

[11] Geffert V. Space Hierarchy Theorem Revised. In: Proceedings of the MFCS 2001 Twenty-Sixth Symposium on Mathematical Foundations of Computer Science, volume 2136 of *LNCS*. 2001 pp. 387–397. doi:10.1007/3-540-44683-4_34.

[12] Kinne J, van Melkebeek D. Space Hierarchy Results for Randomized Models. In: Proceedings of the STACS 2008 Twenty-Fifth Annual Symposium on Theoretical Aspects of Computer Science. 2008 pp. 433–444. doi:10.4230/LIPIcs.STACS.2008.1363.

[13] Blum M. A Machine-Independent Theory of the Complexity of Recursive Functions. *Journal of the ACM*, 1967. **14**(2):322–336. doi:10.1145/321386.321395.

[14] Sipser M. Introduction to the Theory of Computation. International Thomson Publishing, 1st edition, 1996. ISBN 053494728X. doi:10.1145/230514.571645.

[15] Žák S, Šíma J. A Turing Machine Distance Hierarchy. In: Proceedings of the LATA 2013 Seventh International Conference on Language and Automata Theory and Applications, volume 7810 of *LNCS*. 2013 pp. 570–578. doi:10.1007/978-3-642-37064-9_50.

[16] Freedman AR, Ladner RE. Space Bounds for Processing Contentless Inputs. *Journal of Computer and System Sciences*, 1975. **11**(1):118–128. doi:10.1016/S0022-0000(75)80052-3.

[17] Geffert V. Nondeterministic Computations in Sublogarithmic Space and Space Constructibility. *SIAM Journal on Computing*, 1991. **20**(3):484–498. doi:10.1137/0220031.

[18] Žák S. A Turing Machine Space Hierarchy. *Kybernetika*, 1979. **15**(2):100–121.

[19] Žák S. A Turing Machine Oracle Hierarchy, Parts I and II. *Commentationes Mathematicae Universitatis Carolinae*, 1980. **21**(1):11–39.