A Turing Machine Distance Hierarchy

Stanislav Žák[*] and Jiří Šíma[**]

Institute of Computer Science, Academy of Sciences of the Czech Republic,
P. O. Box 5, 18207 Prague 8, Czech Republic, {stan|sima}@cs.cas.cz

Abstract. We introduce a new so-called distance complexity measure for Turing machine computations which is sensitive to long-distance transfers of information on the worktape. An important special case of this measure can be interpreted as a kind of buffering complexity which counts the number of necessary block uploads into a virtual buffer on top of the worktape. Thus, the distance measure can be used for investigating the buffering aspects of Turing computations. In this paper, we start this study by proving a tight separation and hierarchy result. In particular, we show that a very small increase in the distance complexity bound (roughly from $c(n)$ to $c(n+1) + \mathit{constant}$) brings provably more computational power to both deterministic and nondeterministic Turing machines. For this purpose, we formulate a very general diagonalization method for Blum-like complexity measures. We also obtain a hierarchy of the distance complexity classes.

Keywords: Turing machine, hierarchy, distance complexity, diagonalization

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 [3, 6–9, 1, 4, 5].

The present paper follows this direction of research. A new nontraditional complexity measure is introduced for both deterministic and nondeterministic Turing machines (TM) with one worktape (Section 2). This so-called *distance complexity* (Section 6) is sensitive to long transfers of information on the worktape of a Turing machine while the short transfers are not counted.

[*] S.Ž.'s research was partially supported by the projects GA ČR P202/10/1333 and RVO: 67985807.

[**] J.Š.'s research was partially supported by the projects GA ČR P202/12/G061 and RVO: 67985807.

In particular, the distance complexity of a given computation, which can be understood as a sequence of TM configurations, is the length of its certain subsequence defined as follows. The subsequence starts with the initial configuration c_0. The next element c_{i+1} is defined to be the first configuration c after c_i such that the distance (measured in the number of tape cells) between the worktape head position of c and the worktape head position of some configuration that precedes c and succeeds or is equal to c_i, reaches a given bound $d(n)$ where $d(n)$ is a parameter of the distance measure depending on the input length n. In other words, c_{i+1} is the first configuration along the computation such that its worktape head position is exactly at distance $d(n)$ from that of some preceding configuration taken from the segment of computation starting with c_i.

A special case of this distance complexity in which the distance is measured from the head position of previous-element configuration c_i (i.e. the worktape head position of c_{i+1} is at distance $d(n)$ from that of c_i), can be interpreted as a kind of *buffering complexity* of Turing computations. In particular, the worktape is divided into blocks of the same number $d(n)$ of cells and the control unit has a virtual buffer memory whose capacity is two blocks, the concatenated so-called left-hand and right-hand side buffers, respectively. The worktape head position has to be within this buffer and the buffering complexity measures the number of necessary block uploads into the buffer. Namely, if the worktape head finds itself at the right end of the right-hand side buffer containing the $(k+1)$st block of the worktape (while the left-hand side buffer stores the kth worktape block), and has to move further to the right worktape cell $(k+1)d(n)+1$ outside the buffer, then the content of the right-hand side buffer is copied to the left-hand side one and the $(k+2)$nd block of the worktape is uploaded to the right-hand side buffer while the worktape head ends up at the left end of the right-hand side buffer. In other words, the virtual buffer moves to the right by $d(n)$ cells which is reminiscent of a super-head reading the whole block as a super-cell of length $d(n)$. Similarly, for the worktape head at the left end of the left-hand side buffer which moves to the left. In this way, the distance measure can be used for investigating the buffering aspects of Turing computations.

We start our study by separation (Section 6) and hierarchy (Section 7) results for the distance complexity which are surprisingly very tight. This indicates that the new complexity measure is an appropriate tool for classifying the computations. The tightness in the results requires that the worktape alphabet is fixed and the measure is not applied to TM computations directly but instead to their simulations on a fixed universal Turing machine. The results are of the form that a shift by one in the argument of the complexity bound (and of parameter d plus an additive constant) leads to a strictly greater computational power. In the case of a linear complexity bound, the increase in the bound by an additive constant is sufficient to gain more power. For the hierarchy of complete languages the increase in the bound is slightly larger (Section 7). The main tool of the proof is the general diagonalization method introduced in [9] (Section 3) which is applied (Section 5) to arbitrary Blum-like complexity measures (Section 4).

2 Technical Preliminaries

We denote by N the set of natural numbers (including 0). By a complexity bound we mean any mapping c, $c : N \to N$. In the notation of complexity classes, $c(n+1)$ stands for complexity bound c' such that, for each $n \in N$, $c'(n) =_{df} c(n+1)$. By a language we mean any L, $L \subseteq \{0,1\}^*$. Moreover, ϵ denotes the empty word.

By a *Turing machine* we mean any machine with two-way read-only input tape and with one semi-infinite worktape (infinite to the right) with worktape alphabet $0, 1, b$ and with endmarker # at the left end side (at the 0-th cell of the tape), allowing for both the deterministic or nondeterministic versions. Any computation of a Turing machine on an input word can be understood as a sequence of its configurations.

The programs are words from $\{0,1\}^+$ and we suppose that there is a machine which, having any word p on its worktape, is able to decide whether p is a program without any use of the cells outside of p. If p is a program, then M_p is the corresponding machine. For any machine M, by $L(M)$ we denote the language accepted by M, and by p_M we mean the program of M.

On any input u, the universal machine starts its computation with some program p at the left end side of the worktape and it simulates machine M_p on u. We implicitly assume that the universal machine shifts program p along its worktape in order to follow the shifts of the head of the simulated machine M_p.

Let S be a set of programs and let $C = \{L_p \mid p \in S\}$ be a class of languages. We say that C is uniformly recursive iff there is a machine M such that for each $p \in S$ and for each $u \in \{0,1\}^*$, computing on the input pu, M decides whether $u \in L_p$ or not.

3 The Diagonalization Theorem

In the sequel we use the diagonalization method which is based on the theorem from [9]. This theorem is formulated without any notion concerning computability nor complexity. It is formulated only in terms of languages, functions and relations. Due to this property the method is largely applicable towards the world of computational complexity.

We say that two languages L, L' are equivalent $L \sim L'$ iff they differ only on a finite number of words. For a class C of languages we define $\mathcal{E}(C) =_{df} \{L' \mid \exists L \in C\ (L \sim L')\}$. Then the fact $L \notin \mathcal{E}(C)$ implies that L differs from any language of $\mathcal{E}(C)$ on infinitely many words.

Now we are ready to introduce our diagonalization theorem.

Theorem 1. *Let L be a language and let C be a class of languages indexed by a set S, that is $C = \{L_p \mid p \in S\}$. Let R be a language and let F be a mapping, $F : R \to S$, such that $(\forall\ p \in S)(\exists^\infty\ r \in R)(F(r) = p)$. Let z be a mapping, $z : R \to N$, such that for each $r \in R$, $z(r)$ satisfies the following two conditions:*
a) $r1^{z(r)} \in L \ \leftrightarrow\ r \notin L_{F(r)}$,

b) $(\forall j,\, 0 \leq j < z(r))(r1^j \in L \leftrightarrow\ r1^{j+1} \in L_{F(r)})$.
Then $L \notin \mathcal{E}(C)$.

The idea of the proof by contradiction is as follows. From the assumption $L \in \mathcal{E}(C)$ we derive an appropriate r such that $L = L_{F(r)}$. Then conditions (a), (b) produce a contradiction immediately.

The idea of the application to the complexity world is as follows. The decision whether $r \in L_{F(r)}$ or not is achieved during the computation on the word $r1^{z(r)}$ where $z(r)$ is a very large function. While from the point of length $|r|$ this decision requires very large amount of the source in question (e. g. space, time etc.), especially in the case of nondeterministic computations, from the point of length $|r1^{z(r)}|$ this amount is negligible. The main consumption of the sources is now concentrated in the simulation on the input of length $n+1$. Even in the case of nondeterministic computations the respective increase of complexity is moderate.

For the sake of completeness we add the complete proof of the theorem [9].

Proof. By contradiction. Let $L \in \mathcal{E}(C)$. Hence, $L \sim L_p$ for some $p \in S$. Moreover, there is $r \in R$ such that $F(r) = p$ and the languages L and $L_{F(r)}(= L_p)$ differ only on words shorter than r. In particular, for each $j \in N$, $r1^j \in L_{F(r)}$ iff $r1^j \in L$. Hence by condition (b), $r \in L \leftrightarrow r1^{z(r)} \in L$, and further by condition (a), $r1^{z(r)} \in L \leftrightarrow r \notin L$, which is a contradiction. □

4 Complexity Measures and Classes

Inspired by Blum axioms [2], by a complexity measure we mean any (partial) mapping $m : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to N$. Informally, the first argument is intended to be the input word, the second one corresponds to a program, and the third one represents the initial content of the worktape.

Let S, $S \subseteq \{0,1\}^+$, be the set of programs of the machines in question. For any $p \in S$ and for any complexity bound $c : N \to N$ we define $L_{m,p,c} =_{df} \{u \in \{0,1\}^* | m(u,p,\epsilon) \leq c(|u|)\}$ where ϵ is the empty word. We say that M_p accepts its language within m-complexity c iff $L(M_p) = L_{m,p,c}$.

Let m be a complexity measure, U be a universal machine, and p_U be the program of U. By the complexity measure m_U we mean the mapping $m_U : \{0,1\}^* \times \{0,1\}^* \to N$, $m_U(u,p) =_{df} m(u,p_U,p)$.

For any $p \in S$ and for any $c : N \to N$ we define language $L_{m_U,p,c} =_{df} \{u \in \{0,1\}^* \,|\, m_U(u,p) \leq c(|u|)\}$. We say that M_p accepts its language within m_U-complexity c iff $L(M_p) = L_{m_U,p,c}$. We also say that $L(M_p)$ is an (m_U, c)-complete language.

We define the complexity class $C_{m,c} =_{df} \{L_{m,p,c} \,|\, p \in S\}$. Similarly for $m = m_U$, $C_{m_U,c} =_{df} \{L_{m_U,p,c} \,|\, p \in S\}$. We say that M_p accepts its language within m-complexity c iff $L(M_p) = L_{m,p,c}$.

Let $C_{comp,m_U,c} =_{df} \{L \,|\, (\exists p \in S)\ L = L(M_p) = L_{m_U,p,c}\}$ be a class composed of all (m_U, c)-complete languages which we call an (m_U, c)-complete (or shortly complete) class.

5 The Diagonalization Result

The following definition forms the first step in the process of implementing our diagonalization theorem (Theorem 1) in the milieu of computations and complexity measures. The complexity measure is still not specified, so the constructed diagonalizer can possibly be applied to any Blum-like measure.

Definition 2. *Let S be a recursive set of programs (of machines in question). Let R be the set of program codes, $R =_{df} \{1^k0^l \,|\, k,l \in N;\, k,l > 0;\, bin(k) \in S\}$ (where $bin(k)$ is the binary code of k). Let F be a mapping, $F : R \to S$, $F(1^k0^l) = bin(k)$. For any $p \in S$, let L_p be a uniformly recursive part of $L(M_p)$.*

Then by diagonalizer M *we mean the machine that works on the input of length n as follows: M first checks whether the input is of the form $1^k0^l1^j$, $k,l,j \in N$, $k,l > 0$, and $j \geq 0$. Then M constructs the initial segment of its worktape of length $\log n$, $n = k+l+j$. Within this segment, M constructs $bin(k)$ and tries to verify that $bin(k) \in S$. If $bin(k) = p \in S$ then $1^k0^l = r \in R$ and M tries further to decide whether $1^k0^l \in L_p$ (i.e. whether $r \in L_{F(r)}$) using only the initial segment of the worktape of length $\log n$ constructed previously by M. If M accomplishes this decision, then M accepts iff $1^k0^l \notin L_p$ ($r \notin L_{F(r)}$). Otherwise, M simulates p on longer input $1^k0^l1^{j+1}$ in the same manner as universal machine U can do. (This simulation is not limited in the amount of used tape.)*

Moreover, for $r \in R$ we define $z(r)$ to be the minimal j such that working on $r1^j$, diagonalizer M decides whether $r \in L_{F(r)}$ or not.

This definition has introduced a diagonalizer which is appropriate for language separation tasks. A similar diagonalizer can be defined which is appropriate for proving separation results for unary languages. The following theorem translates Theorem 1 into the world of computations and Blum-like complexity measures.

Theorem 3. *Let m be a measure and c be a complexity bound. Let S, R, F, L_p, M, z be as in Definition 2, and $C =_{df} \{L_p \,|\, p \in S\}$. For each $r \in R$, let the following two conditions hold:*
a) $r1^{z(r)} \in L_{m,p_M,c(n+1)} \leftrightarrow r \notin L_{F(r)}$,
b) for each $j < z(r)$ ($r1^j \in L_{m,p_M,c(n+1)} \leftrightarrow r1^{j+1} \in L_{F(r)}$).
Then $L_{m,p_M,c(n+1)} \notin \mathcal{E}(C)$.

Proof. S, R, F, C satisfy the assumptions of Theorem 1. It is clear that also z and $L = L_{m,p_M,c(n+1)}$ satisfy the assumptions of Theorem 1 and especially its conditions (a) and (b). The statement follows immediately. □

6 The Separation Result for the Distance Measure

We introduce a new complexity measure which we call the *distance complexity*. Let d, $d : N \to N$, be a positive function. For any machine, we define the d-subsequence $\{c_i\}$ of its computation on a word u in question as follows:

1. c_0 is the initial configuration.
2. Given c_i, the next configuration c_{i+1} is defined to be the first configuration c after c_i such that there is a configuration c' in between c_i and c (along the computation) or $c' = c_i$, and the distance between the worktape head positions of c and c' equals $d(|u|)$.

The distance complexity measure m^d is defined as follows. For $u \in \{0,1\}^*$ and for $p \in S$, define $m^d(u,p,\epsilon)$ to be the minimum length of the d-subsequences over all accepting computations of M_p on u. Clearly, the *buffering complexity* is obtained as a special case of the distance measure when we demand $c' = c_i$ in part 2 of the definition of d-subsequence above.

Note that by means of the length of subsequences one can also define the classical time or space complexity measures. The subsequence is simply the whole computation for the time complexity while the space complexity is obtained when the subsequence contains exactly each configuration c_k such that the worktape head position of any previous configuration c_i, $i < k$, is to the left of the worktape head position of c_k.

Lemma 4. *Let d be a function, U be the universal machine, and u be an input word. Then $m^d(u,p_M,\epsilon) = m_U^{d+|p_M|}(u,p_M)$.*

Proof. Hint. On the worktape of the universal machine U the key property of the simulation of M is that the program p_M is being shifted along the tape following the shifts of the head of M. The distance $d(n)$ on the worktape of M is transformed to the distance $d(n) + |p_M|$ on the worktape of U. □

Now we prove the separation result for the distance complexity measure.

Theorem 5. *Let U be a fixed universal machine, c be a recursive complexity bound, and d, $d : N \to N$, be a recursive nondecreasing function satisfying $d > \log_2$. Then there is a language $L \in C_{m_U^{d(n+1)+K},c(n+1)}$ (where K is a constant) which is not in the class $\mathcal{E}(C_{m_U^d,c})$.*

Proof. Let S, R, F, L_p, M, z be as in Definition 2 and $L_p =_{df} L_{m_U^d,p,c}$. Let C be also as in Definition 2, $C = C_{m_U^d,c}$, and $m =_{df} m^{d(n+1)}$. We define $L =_{df} L_{m^{d(n+1)},p_M,c(n+1)}$.

We will prove that $L \notin \mathcal{E}(C)$. It suffices to verify conditions (a) and (b) of Theorem 3. For $r \in R$, machine M, working on $r1^{z(r)}$, uses only $\log n < d(n)$ cells of its worktape. Hence, M decides, whether $r \in L_{F(r)}$ or not, within the complexity bound 1 corresponding to the initial configuration in the d-subsequence ($\log n < d(n)$) under the measure $m = m^{d(n+1)}$. Thus, condition (a) is satisfied.

For $j < z(r)$, let us verify that $r1^j \in L_{m^{d(n+1)},p_M,c(n+1)} \leftrightarrow r1^{j+1} \in L_{m_U^d,F(r),c}$ $= L_{F(r)}$. This is true since from the definition of M, it follows that, in this case, M works as U does. Hence, condition (b) of Theorem 3 holds. According to Theorem 3, we have $L \notin \mathcal{E}(C)$.

Furthermore,

$$\begin{aligned} L &= L_{m^{d(n+1)},p_M,c(n+1)} \\ &= \left\{ u \in \{0,1\}^* \,\middle|\, m^{d(n+1)}(u,p_M,\epsilon) \le c(|u|+1) \right\} \\ &= \left\{ u \in \{0,1\}^* \,\middle|\, m_U^{d(n+1)+|p_M|}(u,p_M) \le c(|u|+1) \right\} \quad \text{(according to Lemma 4)} \\ &= L_{m_U^{d(n+1)+|p_M|},p_M,c(n+1)} \in C_{m_U^{d(n+1)+|p_M|},c(n+1)} . \quad \square \end{aligned}$$

7 The Hierarchy

In order to prove the hierarchy theorem, we first show the folowing lemma.

Lemma 6. *Let c_1, c_2 be complexity bounds and d_1, d_2 be functions, $d_1 : N \to N$, $d_2 : N \to N$. If $c_1 \le c_2$ and $d_1 \le d_2$, then $C_{comp,m_U^{d_1},c_1} \subseteq C_{comp,m_U^{d_2},c_2}$.*

Proof. Let $L \in C_{comp,m_U^{d_1},c_1}$. Then there is a program $p \in S$ such that $L = L_{m_U^{d_1},p,c_1} = L(M_p)$. It suffices to prove that $L_{m_U^{d_1},p,c_1} \subseteq L_{m_U^{d_2},p,c_2}$ which implies $L = L_{m_U^{d_2},p,c_2} = L(M_p)$, and consequently $L \in C_{comp,m_U^{d_2},c_2}$. Suppose $u \in L_{m_U^{d_1},p,c_1}$.

For $i = 1, 2$, let c_j^i be the j-th configuration of d_i-subsequence of a computation of U on the input u with p given on the worktape of the starting configuration. For our purposes, it suffices to prove that for each j, c_j^1 is not later in the computation than c_j^2. We know that $c_0^1 = c_0^2$. We continue by contradiction. Let j be the first number that the configuration c_j^1 is not after c_j^2 but c_{j+1}^1 is after c_{j+1}^2. Thus in the sequence of configurations between c_j^2 and c_{j+1}^2 we find a configuration whose worktape head position is within the distance of $d_2(n) \ge d_1(n)$ from the worktape head position of c_{j+1}^2 which contradicts the assumption that c_{j+1}^1 is after c_{j+1}^2. Hence, $u \in L_{m_U^{d_2},p,c_2}$. $\square$

We see that for the language L from Theorem 5, $L \notin \mathcal{E}(C_{m_U^d,c}) \supseteq C =_{df} \mathcal{E}(C_{comp,m_U^d,c})$ holds. In order to obtain a hierarchy we search for $d_1 \ge d$ and $c_1 \ge c$ such that $L \in C_1 =_{df} C_{comp,m_U^{d_1},c_1}$. According to Lemma 6, this will give $C \subseteq C_1$ for complete classes C and C_1, which together with $L \in C_1 \setminus C$ will provide the desired hierarchy.

Recall from the proof of Theorem 5 that $L = L_{m^{d(n+1)},p_M,c(n+1)}$. We will define machine M' such that $L(M') = L$. We will first describe the main ideas of how M' computes. At the beginning of its computation, M' constructs on its worktape two segments of lengths $4\log(d(n+1))$ and $\log(c(n+1))$, respectively. Then M' simulates M so that M' shifts the two segments along the worktape together with the worktape head of M. The first segment of length $4\log(d(n+1))$ serves for identifying the time instant at which the current configuration of M belongs to the $d(n+1)$-subsequence. For this purpose, it suffices to keep and update the current head position, the minimum and maximum head positions—all these three positions measured as a (possibly oriented) distance from the head position of the previous-element configuration from the $d(n+1)$-subsequence. In addition, this includes a test whether a current maximum distance between two head positions equals $d(n+1)$ which requires the value $d(n+1)$ to be precomputed and shifted within this first segment. Hence, the length of the first segment follows. Similarly, the second segment of length $\log(c(n+1))$ serves for halting the computation after $c(n+1)$ configurations of the $d(n+1)$-subsequence.

In fact, the implementation of the ideas above requires a slightly longer segments due to the fact that the worktape alphabet of M' is restricted to $\{0,1\}$ according to our definition of Turing machine. 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. So, the introduced segments must be twice as long as above.

Obviously, $L(M') = L$. Moreover, $L(M') \in C_{comp,m_U^{d'},c'}$ for

$$d' = d(n+1) + 8\log(d(n+1)) + 2\log(c(n+1)) + K\,, \tag{1}$$
$$c' = c(n+1) + D \tag{2}$$

where $K = |p_{M'}|$ is a constant and $D : N \to N$ is a function of n which compensates for the consumption of the source for constructing the segments and computing the values of $d(n+1)$ and $c(n+1)$.

The hierarchy result is summarized in the following theorem.

Theorem 7. *Let U be a fixed universal machine, c be a recursive complexity bound, and d, $d : N \to N$, be a recursive nondecreasing function satisfying $d > \log_2$. Let functions $c' : N \to N$ and $d' : N \to N$ be defined by formula (2) and (1), respectively. Then $C_{comp,m_U^d,c} \subsetneqq C_{comp,m_U^{d'},c'}$.*

8 Conclusions

In this paper we have introduced a new distance complexity measure for computations on Turing machines with one worktape. This measure can be used for investigating the buffering aspects of Turing computations. As a first step along this direction, we have proven quite strong separation and hierarchy results. The presented theorems can even be strengthened to unary languages (by modifications in Definition 2 of diagonalizer M). Many questions concerning e.g. the comparison to other complexity measures, reductions, completeness and complexity classes remain open for further research.

We have also formulated our diagonalization method for general Blum-like complexity measures which is interesting by its own. Analogous theorems can possibly be proven for other types of machines such as those with auxiliary pushdown or counter, or with oracle etc.

References

1. Allender, E., Beigel, R., Hertrampf, U., Homer, S.: Almost-everywhere complexity hierarchies for nondeterministic time. Theoretical Computer Science 115(2), 225–241 (1993)
2. Blum, M.: A machine-independent theory of the complexity of recursive functions. Journal of the ACM 14(2), 322–336 (1967)
3. Cook, S.A.: A hierarchy for nondeterministic time complexity. Journal of Computer and System Sciences 7(4), 343–353 (1973)
4. Geffert, V.: Space hierarchy theorem revised. In: Proceedings of the MFCS 2001 Twenty-Sixth Symposium on Mathematical Foundations of Computer Science. LNCS, vol. 2136, pp. 387–397 (2001)
5. 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. pp. 433–444 (2008)
6. Seiferas, J.I.: Relating refined space complexity classes. Journal of Computer and System Sciences 14(1), 100–129 (1977)
7. Seiferas, J.I., Fischer, M.J., Meyer, A.R.: Separating nondeterministic time complexity classes. Journal of the ACM 25(1), 146–167 (1978)
8. Sudborough, I.H.: Separating tape bounded auxiliary pushdown automata classes. In: Proceedings of the STOC'77 Ninth Annual ACM Symposium on Theory of Computing. pp. 208–217 (1977)
9. Žák, S.: A turing machine time hierarchy. Theoretical Computer Science 26, 327–333 (1983)