Edit Distance for Pushdown Automata

Krishnendu Chatterjee[✉], Thomas A. Henzinger, Rasmus Ibsen-Jensen, and Jan Otop

IST Austria, Wien-Umgebung, Austria
krish.chat@gmail.com

Abstract. The edit distance between two words w_1, w_2 is the minimal number of word operations (letter insertions, deletions, and substitutions) necessary to transform w_1 to w_2. The edit distance generalizes to languages $\mathcal{L}_1, \mathcal{L}_2$, where the edit distance is the minimal number k such that for every word from $\mathcal{L}_1$ there exists a word in $\mathcal{L}_2$ with edit distance at most k. We study the edit distance computation problem between pushdown automata and their subclasses. The problem of computing edit distance to pushdown automata is undecidable, and in practice, the interesting question is to compute the edit distance from a pushdown automaton (the implementation, a standard model for programs with recursion) to a regular language (the specification). In this work, we present a complete picture of decidability and complexity for deciding whether, for a given threshold k, the edit distance from a pushdown automaton to a finite automaton is at most k.

1 Introduction

Edit distance. The edit distance [13] between two words is a well-studied metric, which is the minimum number of edit operations (insertion, deletion, or substitution of one letter by another) that transforms one word to another. The edit distance between a word w to a language $\mathcal{L}$ is the minimal edit distance between w and words in $\mathcal{L}$. The edit distance between two languages $\mathcal{L}_1$ and $\mathcal{L}_2$ is the supremum over all words w in $\mathcal{L}_1$ of the edit distance between w and $\mathcal{L}_2$.

Significance of edit distance. The notion of *edit distance* provides a quantitative measure of "how far apart" are (a) two words, (b) words from a language, and (c) two languages. It forms the basis for quantitatively comparing sequences, a problem that arises in many different areas, such as error-correcting codes, natural language processing, and computational biology. The notion of edit distance between languages forms the foundations of a quantitative approach to verification. The traditional qualitative verification (model checking) question is the *language inclusion* problem: given an implementation (source language)

This research was funded in part by the European Research Council (ERC) under grant agreement 267989 (QUAREM), by the Austrian Science Fund (FWF) projects S11402-N23 (RiSE) and Z211-N23 (Wittgenstein Award), FWF Grant No P23499-N23, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), and MSR faculty fellows award.

M.M. Halldórsson et al. (Eds.): ICALP 2015, Part II, LNCS 9135, pp. 121–133, 2015.
DOI: 10.1007/978-3-662-47666-6_10

defined by an automaton $\mathcal{A}_I$ and a specification (target language) defined by an automaton $\mathcal{A}_S$, decide whether the language $\mathcal{L}(\mathcal{A}_I)$ is included in the language $\mathcal{L}(\mathcal{A}_S)$ (i.e., $\mathcal{L}(\mathcal{A}_I) \subseteq \mathcal{L}(\mathcal{A}_S)$). The *threshold edit distance* (TED) problem is a generalization of the language inclusion problem, which for a given integer threshold $k \geq 0$ asks whether every word in the source language $\mathcal{L}(\mathcal{A}_I)$ has edit distance at most k to the target language $\mathcal{L}(\mathcal{A}_S)$ (with $k = 0$ we have the traditional language inclusion problem). For example, in simulation-based verification of an implementation against a specification, the measured trace may differ slightly from the specification due to inaccuracies in the implementation. Thus, a trace of the implementation may not be in the specification. However, instead of rejecting the implementation, one can quantify the distance between a measured trace and the specification. Among all implementations that violate a specification, the closer the implementation traces are to the specification, the better [5,7,10]. The edit distance problem is also the basis for *repairing* specifications [2,3].

Our models. In this work we consider the edit distance computation problem between two automata $\mathcal{A}_1$ and $\mathcal{A}_2$, where $\mathcal{A}_1$ and $\mathcal{A}_2$ can be (non)deterministic finite automata or pushdown automata. Pushdown automata are the standard models for programs with recursion, and regular languages are canonical to express the basic properties of systems that arise in verification. We denote by DPDA (resp., PDA) deterministic (resp., nondeterministic) pushdown automata, and DFA (resp., NFA) deterministic (resp., nondeterministic) finite automata. We consider source and target languages defined by DFA, NFA, DPDA, and PDA. We first present the known results and then our contributions.

Previous results. The main results for the classical language inclusion problem are as follows [11]: (i) if the target language is a DFA, then it can be solved in polynomial time; (ii) if either the target language is a PDA or both source and target languages are DPDA, then it is undecidable; (iii) if the target language is an NFA, then (a) if the source language is a DFA or NFA, then it is PSpace-complete, and (b) if the source language is a DPDA or PDA, then it is PSpace-hard and can be solved in ExpTime (to the best of our knowledge, there is a complexity gap where the upper bound is ExpTime and the lower bound is PSpace). The TED problem was studied for DFA and NFA, and it is PSpace-complete, when the source and target languages are given by DFA or NFA [2,3].

Our contributions. Our main contributions are as follows.

1. We show that the TED problem is ExpTime-complete, when the source language is given by a DPDA or a PDA, and the target language is given by a DFA or NFA. We present a hardness result which shows that the TED problem is ExpTime-hard for source languages given as DPDA and target languages given as DFA. We present a matching upper bound by showing that for source languages given as PDA and target languages given as NFA the problem can be solved in ExpTime. As a consequence of our lower bound we obtain that the language inclusion problem for source languages given by DPDA (or PDA) and target languages given by NFA is ExpTime-complete. Thus we present a complete picture of the complexity of the TED problem,

Table 1. Complexity of the language inclusion problem from $\mathcal{C}_1$ to $\mathcal{C}_2$. Our results are boldfaced.

	$\mathcal{C}_2 = \mathsf{DFA}$	$\mathcal{C}_2 = \mathsf{NFA}$	$\mathcal{C}_2 = \mathsf{DPDA}$	$\mathcal{C}_2 = \mathsf{PDA}$
$\mathcal{C}_1 \in \{\mathsf{DFA}, \mathsf{NFA}\}$	PTime	PSpace-c	PTime	
$\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$		**ExpTime-c (Th. 2)**	undecidable	

Table 2. Complexity of $\mathsf{FED}(\mathcal{C}_1, \mathcal{C}_2)$. Our results are boldfaced. See Conjecture 14 for the open complexity problem of $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$ and $\mathcal{C}_2 = \mathsf{DFA}$.

	$\mathcal{C}_2 = \mathsf{DFA}$	$\mathcal{C}_2 = \mathsf{NFA}$	$\mathcal{C}_2 = \mathsf{DPDA}$	$\mathcal{C}_2 = \mathsf{PDA}$
$\mathcal{C}_1 \in \{\mathsf{DFA}, \mathsf{NFA}\}$	coNP-c [3]	PSpace-c [3]	open (Conj. 18)	
$\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$	coNP-hard [3] **in ExpTime (Th. 8)**	**ExpTime-c (Th. 8)**	**undecidable (Prop. 15)**	

Table 3. Complexity of $\mathsf{TED}(\mathcal{C}_1, \mathcal{C}_2)$. Our results are boldfaced.

	$\mathcal{C}_2 = \mathsf{DFA}$	$\mathcal{C}_2 = \mathsf{NFA}$	$\mathcal{C}_2 = \mathsf{DPDA}$	$\mathcal{C}_2 = \mathsf{PDA}$
$\mathcal{C}_1 \in \{\mathsf{DFA}, \mathsf{NFA}\}$	PSpace-c [2]		**undecidable (Prop. 17)**	
$\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$	**ExpTime-c (Th. 2 (1))**		undecidable	

and in addition we close a complexity gap in the classical language inclusion problem. In contrast, if the target language is given by a DPDA, then the TED problem is undecidable even for source languages given as DFA. Note that the interesting verification question is when the implementation (source language) is a DPDA (or PDA) and the specification (target language) is given as DFA (or NFA), for which we present decidability results with optimal complexity.

2. We also consider the *finite edit distance* (FED) problem, which asks whether there exists $k \geq 0$ such that the answer to the TED problem with threshold k is YES. For finite automata, it was shown in [2,3] that if the answer to the FED problem is YES, then a polynomial bound on k exists. In contrast, the edit distance can be exponential between DPDA and DFA. We present a matching exponential upper bound on k for the FED problem from PDA to NFA. Finally, we show that the FED problem is ExpTime-complete when the source language is given as a DPDA or PDA, and the target language as an NFA.

Our results are summarized in Tables 1, 2 and 3. Due to space constraints we omit some technical proofs, which are presented in the full version [6].

Related work. Algorithms for edit distance have been studied extensively for words [1,12,13,15–17]. The edit distance between regular languages was studied in [2,3], between timed automata in [8], and between straight line programs in [9,14]. A near-linear time algorithm to approximate the edit distance for a word to a Dyck language has been presented in [18].

2 Preliminaries

2.1 Words, Languages and Automata

Words. Given a finite alphabet Σ of letters, a *word* w is a finite sequence of letters. For a word w, we define $w[i]$ as the i-th letter of w and $|w|$ as its length. We denote the set of all words over Σ by Σ^*. We use ϵ to denote the empty word.

Pushdown Automata. A *(non-deterministic) pushdown automaton* (PDA) is a tuple $(\Sigma, \Gamma, Q, S, \delta, F)$, where Σ is the input alphabet, Γ is a finite stack alphabet, Q is a finite set of states, $S \subseteq Q$ is a set of initial states, $\delta \subseteq Q \times \Sigma \times (\Gamma \cup \{\perp\}) \times Q \times \Gamma^*$ is a finite transition relation and $F \subseteq Q$ is a set of final (accepting) states. A PDA $(\Sigma, \Gamma, Q, S, \delta, F)$ is a *deterministic pushdown automaton* (DPDA) if $|S| = 1$ and δ is a function from $Q \times \Sigma \times (\Gamma \cup \{\perp\})$ to $Q \times \Gamma^*$. We denote the class of all PDA (resp., DPDA) by PDA (resp., DPDA). We define the size of a PDA $\mathcal{A} = (\Sigma, \Gamma, Q, S, \delta, F)$, denoted by $|\mathcal{A}|$, as $|Q| + |\delta|$.

Runs of Pushdown Automata. Given a PDA $\mathcal{A}$ and a word $w = w[1] \dots w[k]$ over Σ, a *run* π of $\mathcal{A}$ on w is a sequence of elements from $Q \times \Gamma^*$ of length $k+1$ such that $\pi[0] \in S \times \{\epsilon\}$ and for every $i \in \{1, \dots, k\}$ either (1) $\pi[i-1] = (q, \epsilon)$, $\pi[i] = (q', u')$ and $(q, w[i], \perp, q', u') \in \delta$, or (2) $\pi[i-1] = (q, ua)$, $\pi[i] = (q', uu')$ and $(q, w[i], a, q', u') \in \delta$. A run π of length $k+1$ is *accepting* if $\pi[k+1] \in F \times \{\epsilon\}$, i.e., the automaton is in an accepting state and the stack is empty. The *language recognized (or accepted) by* $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of words that have an accepting run.

Context Free Grammar (CFG). A context free grammar in Chomsky normal form (CFG) is a tuple (Σ, V, s, P), where Σ is the alphabet, V is a set of *non-terminals*, $s \in V$ is a *start symbol* and P is a set of *production rules*. A production rule p has one of the following forms: (1) $p : v \rightarrow zu$, where $v, z, u \in V$; or (2) $p : v \rightarrow \alpha$, where $v \in V$ and $\alpha \in \Sigma$; or (3) $p : s \rightarrow \epsilon$.

Languages Generated by CFGs. Fix a CFG $G = (\Sigma, V, s, P)$. We define derivation $\rightarrow_G$ as a relation on $(\Sigma \cup V)^* \times (\Sigma \cup V)^*$ as follows: $w \rightarrow_G w'$ iff $w = w_1 v w_2$, with $v \in V$, and $w' = w_1 u w_2$ for some $u \in (\Sigma \cup V)^*$ such that $v \rightarrow u$ is a production from G. We define $\rightarrow_G^*$ as the transitive closure of $\rightarrow_G$. The *language generated by* G, denoted by $\mathcal{L}(G) = \{w \in \Sigma^* \mid s \rightarrow_G^* w\}$ is the set of words that can be derived from s. CFGs and PDAs are language-wise polynomially equivalent [11] (i.e., there is a polynomial-time procedure that, given a PDA, outputs a CFG of the same language and vice versa).

Finite Automata. A *non-deterministic finite automaton* (NFA) is a PDA with empty stack alphabet. We will omit Γ while referring to NFA, i.e., we will consider them as tuples $(\Sigma, Q, S, \delta, F)$. We denote the class of all NFA by NFA. Analogously to DPDA we define *deterministic finite automata* (DFA).

Language Inclusion. Let $\mathcal{C}_1, \mathcal{C}_2$ be subclasses of PDA. The *inclusion problem from* $\mathcal{C}_1$ *in* $\mathcal{C}_2$ asks, given $\mathcal{A}_1 \in \mathcal{C}_1$, $\mathcal{A}_2 \in \mathcal{C}_2$, whether $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$.

Edit Distance between Words. Given two words w_1, w_2, the edit distance between w_1, w_2, denoted by $ed(w_1, w_2)$, is the minimal number of single letter operations: insertions, deletions, and substitutions, necessary to transform w_1 into w_2.

Edit Distance between Languages. Let $\mathcal{L}_1, \mathcal{L}_2$ be languages. We define the edit distance *from* $\mathcal{L}_1$ *to* $\mathcal{L}_2$, denoted $ed(\mathcal{L}_1, \mathcal{L}_2)$, as $\sup_{w_1 \in \mathcal{L}_1} \inf_{w_2 \in \mathcal{L}_2} ed(w_1, w_2)$. The edit distance between languages is not a distance function. In particular, it is not symmetric.

2.2 Problem Statement

In this section we define the problems of interest. Then, we recall the previous results and succinctly state our results.

Definition 1. *For* $\mathcal{C}_1, \mathcal{C}_2 \in \{\mathsf{DFA}, \mathsf{NFA}, \mathsf{DPDA}, \mathsf{PDA}\}$ *we define the following questions:*

1. The threshold edit distance problem from $\mathcal{C}_1$ to $\mathcal{C}_2$ (denoted $\mathsf{TED}(\mathcal{C}_1, \mathcal{C}_2)$): *Given automata* $\mathcal{A}_1 \in \mathcal{C}_1$*,* $\mathcal{A}_2 \in \mathcal{C}_2$ *and an integer threshold* $k \geq 0$*, decide whether* $ed(\mathcal{L}(\mathcal{A}_1), \mathcal{L}(\mathcal{A}_2)) \leq k$*.*
2. The finite edit distance problem from $\mathcal{C}_1$ to $\mathcal{C}_2$ (denoted $\mathsf{FED}(\mathcal{C}_1, \mathcal{C}_2)$): *Given automata* $\mathcal{A}_1 \in \mathcal{C}_1$*,* $\mathcal{A}_2 \in \mathcal{C}_2$*, decide whether* $ed(\mathcal{L}(\mathcal{A}_1), \mathcal{L}(\mathcal{A}_2)) < \infty$*.*
3. Computation of edit distance from $\mathcal{C}_1$ to $\mathcal{C}_2$: *Given automata* $\mathcal{A}_1 \in \mathcal{C}_1$*,* $\mathcal{A}_2 \in \mathcal{C}_2$*, compute* $ed(\mathcal{L}(\mathcal{A}_1), \mathcal{L}(\mathcal{A}_2))$*.*

We establish the complete complexity picture for the TED problem for all combinations of source and target languages given by $\mathsf{DFA}, \mathsf{NFA}, \mathsf{DPDA}$ and PDA:

1. TED for regular languages has been studied in [2], where PSpace-completeness of $\mathsf{TED}(\mathcal{C}_1, \mathcal{C}_2)$ for $\mathcal{C}_1, \mathcal{C}_2 \in \{\mathsf{DFA}, \mathsf{NFA}\}$ has been established.
2. In Section 3, we study the TED problem for source languages given by pushdown automata and target languages given by finite automata. We establish $\mathsf{ExpTime}$-completeness of $\mathsf{TED}(\mathcal{C}_1, \mathcal{C}_2)$ for $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$ and $\mathcal{C}_2 \in \{\mathsf{DFA}, \mathsf{NFA}\}$.
3. In Section 5, we study the TED problem for target languages given by pushdown automata. We show that $\mathsf{TED}(\mathcal{C}_1, \mathcal{C}_2)$ is undecidable for $\mathcal{C}_1 \in \{\mathsf{DFA}, \mathsf{NFA}, \mathsf{DPDA}, \mathsf{PDA}\}$ and $\mathcal{C}_2 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$.

We study the FED problem for all combinations of source and target languages given by $\mathsf{DFA}, \mathsf{NFA}, \mathsf{DPDA}$ and PDA and obtain the following results:

1. FED for regular languages has been studied in [3]. It has been shown that for $\mathcal{C}_1 \in \{\mathsf{DFA}, \mathsf{NFA}\}$, the problem $\mathsf{FED}(\mathcal{C}_1, \mathsf{DFA})$ is coNP-complete, while the problem $\mathsf{FED}(\mathcal{C}_1, \mathsf{NFA})$ is PSpace-complete.

2. We show in Section 4 that for $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$, the problem $\mathsf{FED}(\mathcal{C}_1, \mathsf{NFA})$ is $\mathsf{ExpTime}$-complete.
3. We show in Section 5 that (1) for $\mathcal{C}_1 \in \{\mathsf{DFA}, \mathsf{NFA}, \mathsf{DPDA}, \mathsf{PDA}\}$, the problem $\mathsf{FED}(\mathcal{C}_1, \mathsf{PDA})$ is undecidable, and (2) the problem $\mathsf{FED}(\mathsf{DPDA}, \mathsf{DPDA})$ is undecidable.

3 Threshold Edit Distance from Pushdown to Regular Languages

In this section we establish the complexity of the TED problem from pushdown to finite automata.

Theorem 2. *(1) For $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$ and $\mathcal{C}_2 \in \{\mathsf{DFA}, \mathsf{NFA}\}$, the $\mathsf{TED}(\mathcal{C}_1, \mathcal{C}_2)$ problem is $\mathsf{ExpTime}$-complete. (2) For $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$, the language inclusion problem from $\mathcal{C}_1$ in NFA is $\mathsf{ExpTime}$-complete.*

We establish the above theorem as follows: In Section 3.1, we present an exponential-time algorithm for $\mathsf{TED}(\mathsf{PDA}, \mathsf{NFA})$ (for the upper bound of (1)). Then, in Section 3.2 we show (2), in a slightly stronger form, and reduce it (that stronger problem) to $\mathsf{TED}(\mathsf{DPDA}, \mathsf{DFA})$, which shows the $\mathsf{ExpTime}$-hardness part of (1).

3.1 Upper Bound

We present an $\mathsf{ExpTime}$ algorithm that, given (1) a PDA $\mathcal{A}_P$; (2) an NFA $\mathcal{A}_N$; and (3) a threshold t given in binary, decides whether the edit distance from $\mathcal{A}_P$ to $\mathcal{A}_N$ is above t. The algorithm extends a construction for NFA by Benedikt et al. [2].

Intuition. The construction uses the idea that for a given word w and an NFA $\mathcal{A}_N$ the following are equivalent: (i) $ed(w, \mathcal{A}_N) > t$, and (ii) for each accepting state s of $\mathcal{A}_N$ and for every word w', if $\mathcal{A}_N$ can reach s from some initial state upon reading w', then $ed(w, w') > t$. We construct a PDA $\mathcal{A}_I$ which simulates the PDA $\mathcal{A}_P$ and stores in its states all states of the NFA $\mathcal{A}_N$ reachable with at most t edits. More precisely, the PDA $\mathcal{A}_I$ remembers in its states, for every state s of the NFA $\mathcal{A}_N$, the minimal number of edit operations necessary to transform the currently read prefix w_p of the input word into a word w'_p, upon which $\mathcal{A}_N$ can reach s from some initial state. If for some state the number of edit operations exceeds t, then we associate with this state a special symbol # to denote this. Then, we show that a word w accepted by the PDA $\mathcal{A}_P$ has $ed(w, \mathcal{A}_N) > t$ iff the automaton $\mathcal{A}_I$ has a run on w that ends (1) in an accepting state of simulated $\mathcal{A}_P$, (2) with the simulated stack of $\mathcal{A}_P$ empty, and (3) the symbol # is associated with every accepting state of $\mathcal{A}_N$.

Lemma 3. *(1) Given (i) a PDA $\mathcal{A}_P$; (ii) an NFA $\mathcal{A}_N$; and (iii) a threshold t given in binary, the decision problem of whether $ed(\mathcal{A}_P, \mathcal{A}_N) \leq t$ can be reduced to the emptiness problem for a PDA of size $O(|\mathcal{A}_P| \cdot (t+2)^{|\mathcal{A}_N|})$. (2) $\mathsf{TED}(\mathsf{PDA}, \mathsf{NFA})$ is in $\mathsf{ExpTime}$.*

3.2 Lower Bound

Our ExpTime-hardness proof of TED(DPDA, DFA) extends the idea from [2] that shows PSpace-hardness of the edit distance for DFA. The standard proof of PSpace-hardness of the universality problem for NFA [11] is by reduction to the halting problem of a fixed Turing machine M working on a bounded tape. The Turing machine M is the one that simulates other Turing machines (such a machine is called universal). The input to that problem is the initial configuration C_1 and the tape is bounded by its size $|C_1|$. In the reduction, the NFA recognizes the language of all words that do not encode valid computation of M starting from the initial configuration C_1, i.e., it checks the following four conditions: (1) the given word is a sequence of configurations, (2) the state of the Turing machine and the adjunct letters follow from transitions of M, (3) the first configuration is not C_1 and (4) the tape's cells are changed only by M, i.e., they do not change values spontaneously. While conditions (1), (2) and (3) can be checked by a DFA of polynomial size, condition (4) can be encoded by a polynomial-size NFA but not a polynomial-size DFA. However, to check (4) the automaton has to make only a single non-deterministic choice to pick a position in the encoding of the computation, which violates (4), i.e., the value at that position is different from the value $|C_1| + 1$ letters further, which corresponds to the same memory cell in the successive configuration, and the head of M does not change it. We can transform a non-deterministic automaton $\mathcal{A}_N$ checking (4) into a deterministic automaton $\mathcal{A}_D$ by encoding such a non-deterministic pick using an external letter. Since we need only one external symbol, we have $\mathcal{L}(\mathcal{A}_N) = \Sigma^*$ iff $ed(\Sigma^*, \mathcal{L}(\mathcal{A}_D)) = 1$. This suggests the following definition:

Definition 4. *An NFA $\mathcal{A} = (\Sigma, Q, S, \delta, F)$ is* nearly-deterministic *if $|S| = 1$ and $\delta = \delta_1 \cup \delta_2$, where δ_1 is a function and in every accepting run the automaton takes a transition from δ_2 exactly once.*

Lemma 5. *There exists a DPDA $\mathcal{A}_P$ such that the problem, given a nearly-deterministic NFA $\mathcal{A}_N$, decide whether $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_N)$, is ExpTime-hard.*

Proof. Consider the *linear-space halting* problem for a (fixed) alternating Turing machine (ATM) M: given an input word w over an alphabet Σ, decide whether M halts on w with the tape bounded by $|w|$. There exists an ATM M_U, such that the linear-space halting problem for M_U is ExpTime-complete [4]. We show the ExpTime-hardness of the problem from the lemma statement by reduction from the linear-space halting problem for M_U.

We w.l.o.g. assume that existential and universal transitions of M_U alternate. Fix an input of length n. The main idea is to construct the language L of words that encode valid terminating computation trees of M_U on the given input. Observe that the language L depends on the given input. We encode a single configuration of M_U as a word of length $n+1$ of the form $\Sigma^i q \Sigma^{n-i}$, where q is a state of M_U. Recall that a computation of an ATM is a tree, where every node of the tree is a configuration of M_U, and it is accepting if every leaf node is an accepting configuration. We encode computation trees T of M_U by traversing T preorder and executing the following: if the current node has only one successor, then write down the current configuration C, terminate it with $\#$ and move down to the successor node in T. Otherwise, if the current node has two successors s, t in the tree, then write down in order (1) the reversed current configuration C^R; and (2) the results of traversals on s and t, each surrounded by parentheses (and), i.e., $C^R\,(\,u^s\,)\,(\,u^t\,)$, where u^s (resp., u^t) is the result of the traversal of the subtree of T rooted at s (resp., t). Finally, if the current node is a leaf, write down the corresponding configuration and terminate with \$. For example, consider a computation with the initial configuration C_1, from which an existential transition leads to C_2, which in turn has a universal transition to C_3 and C_4. Such a computation tree is encoded as follows:

$$C_1 \# C_2^R\ (\,C_3 \ldots \$\,)\ (\,C_4 \ldots \$\,).$$

We define automata $\mathcal{A}_N$ and $\mathcal{A}_P$ over the alphabet $\Sigma \cup \{\#, \$, (,)\}$. The automaton $\mathcal{A}_N$ is a nearly deterministic NFA that recognizes only (but not all) words not encoding valid computation trees of M_U. More precisely, $\mathcal{A}_N$ accepts in four cases: (1) The word does not encode a tree (except that the parentheses may not match as the automaton cannot check that) of computation as presented above. (2) The initial configuration is different from the one given as the input. (3) The successive configurations, i.e., those that result from existential transitions or left-branch universal transitions (like C_2 to C_3), are valid. The right-branch universal transitions, which are preceded by the word ")(", are not checked by $\mathcal{A}_N$. For example, the consistency of the transition C_2 to C_4 is not checked by $\mathcal{A}_N$. Finally, (4) $\mathcal{A}_N$ accepts words in which at least one final configuration, which is a configuration followed by \$, is not final for M_U.

Next, we define $\mathcal{A}_P$ as a DPDA that accepts words in which parentheses match and right-branch universal transitions are consistent, e.g., it checks consistency of a transition from C_2 to C_4. The automaton $\mathcal{A}_P$ pushes configurations on even levels of the computation tree (e.g., C_2^R), which are reversed, on the stack and pops these configurations from the stack to compare them with the following configuration in the right subtree (e.g., C_4). In the example this means that, while the automaton processes the subword $(\,C_3 \ldots \$\,)$, it can use its stack to check consistency of universal transitions in that subword. We assumed that M_U does not have consecutive universal transitions. This means that, for example, $\mathcal{A}_P$ does not need to check the consistency of C_4 with its successive configuration. By construction, we have $L = \mathcal{L}(\mathcal{A}_P) \cap \mathcal{L}(\mathcal{A}_N)^c$ (recall that L is the language of encodings of computations of M_U on the given input) and M_U halts on the given input if and only if $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_N)$ fails. Observe that $\mathcal{A}_P$ is fixed for all inputs, since it only depends on the fixed Turing machine M_U. $\square$

Now, the following lemma, which is (2) of Theorem 2, follows from Lemma 5.

Lemma 6. *The inclusion problem of* DPDA *in* NFA *is* ExpTime*-complete.*

Proof. The ExpTime upper bound is immediate (basically, an exponential determinization of the NFA, followed by complementation, product construction with the PDA, and the emptiness check of the product PDA in polynomial time in the size of the product). ExpTime-hardness of the problem follows from Lemma 5. □

Now, we show that the inclusion problem of DPDA in nearly-deterministic NFA, which is ExpTime-complete by Lemma 5, reduces to $\mathsf{TED}(\mathsf{DPDA}, \mathsf{DFA})$. In the reduction, we transform a nearly-deterministic NFA $\mathcal{A}_N$ over the alphabet Σ into a DFA $\mathcal{A}_D$ by encoding a single non-deterministic choice by auxiliary letters. More precisely, for the transition relation $\delta = \delta_1 \cup \delta_2$ of $\mathcal{A}_N$, we transform every transition $(q, a, q') \in \delta_2$ into $(q, b^{(q,a,q')}, q')$, where $b^{(q,a,q')}$ is a fresh auxiliary letter. Now, consider a DPDA $\mathcal{A}_P$ over the alphabet Σ. As every word in $\mathcal{L}(\mathcal{A}_D)$ contains a single auxiliary letter $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_D)) \geq 1$. Conversely, for every word $w \in \Sigma^*$ we have $ed(w, \mathcal{L}(\mathcal{A}_D)) \leq 1$ implies $w \in \mathcal{A}_N$. Therefore, $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_D)) \leq 1$ if and only if $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_N)$.

Lemma 7. $\mathsf{TED}(\mathsf{DPDA}, \mathsf{DFA})$ *is ExpTime-hard.*

4 Finite Edit Distance from Pushdown to Regular Languages

In this section we study the complexity of the FED problem from pushdown automata to finite automata.

Theorem 8. *(1) For $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$ and $\mathcal{C}_2 \in \{\mathsf{DFA}, \mathsf{NFA}\}$ we have the following dichotomy: for all $\mathcal{A}_1 \in \mathcal{C}_1, \mathcal{A}_2 \in \mathcal{C}_2$ either $ed(\mathcal{L}(\mathcal{A}_1), \mathcal{L}(\mathcal{A}_2))$ is exponentially bounded in $|\mathcal{A}_1| + |\mathcal{A}_2|$ or $ed(\mathcal{L}(\mathcal{A}_1), \mathcal{L}(\mathcal{A}_2))$ is infinite. Conversely, for every n there exist a DPDA $\mathcal{A}_P$ and a DFA $\mathcal{A}_D$, both of the size $O(n)$, such that $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_D))$ is finite and exponential in n (i.e., the dichotomy is asymptotically tight). (2) For $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$ the $\mathsf{FED}(\mathcal{C}_1, \mathsf{NFA})$ problem is ExpTime-complete. (3) Given a PDA $\mathcal{A}_P$ and an NFA $\mathcal{A}_N$, we can compute the edit distance $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_N))$ in time exponential in $|\mathcal{A}_P| + |\mathcal{A}_N|$.*

First, we show in Section 4.1 the exponential upper bound for (1), which together with Theorem 2, implies the ExpTime upper bound for (2). Next, in Section 4.2, we show that $\mathsf{FED}(\mathsf{DPDA}, \mathsf{NFA})$ is ExpTime-hard. We also present the exponential lower bound for (1). Finally, (1), (2), and Theorem 2 imply (3) (by iteratively testing with increasing thresholds up to exponential bounds along with the decision procedure from Theorem 2).

4.1 Upper Bound

In this section we consider the problem of deciding whether the edit distance from a PDA to an NFA is finite. We start with a reduction of the problem. Given a language $\mathcal{L}$, we define $\mathcal{L} = \{u : u$ is a prefix of some word from $\mathcal{L}\}$. We call an automaton $\mathcal{A}$ *safety* if every state of $\mathcal{A}$ is accepting. Note that an automaton is not necessarily total, i.e., some states might not have an outgoing transition for some input symbols, and thus a safety automaton does not necessarily accept all words. Note that for every NFA $\mathcal{A}_N$, the language $\mathcal{L}(\mathcal{A}_N)$ is the language of a safety NFA. We show that $\mathsf{FED}(\mathsf{PDA}, \mathsf{NFA})$ reduces to FED from PDA to safety NFA.

Lemma 9. *Let $\mathcal{A}_P$ be a PDA and $\mathcal{A}_N$ an NFA. The following inequalities hold:*

$$ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_N)) \geq ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_N)) \geq ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_N)) - |\mathcal{A}_N|$$

The following definition and lemma can be seen as a reverse version of the pumping lemma for context free grammars (in that we ensure that the part which can not be pumped is small).

Compact G-decomposition. Given a CFG $G = (\Sigma, V, s, P)$, where $T = |V|$, and a word $w \in \mathcal{L}(G)$ we define *compact G-decomposition* of w as $w = (s_i u_i)_{i=1}^k s_{k+1}$, where s_i and u_i are subwords of w for all i, such that

1. for all ℓ, the word $w(\ell) := (s_i u_i^\ell)_{i=1}^k s_{k+1}$ is in $\mathcal{L}(G)$; and
2. $|w(0)| = \sum_{i=1}^{k+1} |s_i| \leq 2^T$ and $k \leq 2^{T+1} - 2$.

Lemma 10. *For every CFG $G = (\Sigma, V, s, P)$, every word $w \in \mathcal{L}(G)$ admits a compact G-decomposition.*

Intuition. The proof follows by repeated application of the principle behind the pumping lemma, until the part which is not pump-able is small.

Reachability Sets. Fix an NFA. Given a state q in the NFA and a word w, let Q_q^w be the set of states reachable upon reading w, starting in q. The set of states $R(w, q)$ is then the set of states reachable from Q_q^w upon reading any word. For a set Q' and word w, the set $R(w, Q')$ is $\bigcup_{q \in Q'} R(w, q)$.

We have the following **property of reachability sets**: Fix a word u, a number ℓ, an NFA and a set of states Q' of the NFA, where Q' is *closed under reachablity*, i.e., for all $q \in Q'$ and $a \in \Sigma$ we have $\delta(q, a) \subseteq Q'$. Let u' be a word with ℓ non-overlapping occurrences of u (e.g. u^ℓ). Consider any word w with edit distance strictly less than ℓ from u'. Any run on w, starting in some state of Q', reaches a state of $R(u, Q')$. This is because u must be a sub-word of w.

Lemma 11. *Let G be a CFG with a set of non-terminals of size T and let $\mathcal{A}_N$ be a safety NFA with state set Q of size n. The following conditions are equivalent:*

(i) the edit distance $ed(\mathcal{L}(G), \mathcal{L}(\mathcal{A}_N))$ is infinite,
(ii) the edit distance $ed(\mathcal{L}(G), \mathcal{L}(\mathcal{A}_N))$ exceeds $B := (2^{T+1} - 2) \cdot n + 2^T$, and
(iii) there exists a word $w \in \mathcal{L}(G)$, with compact G-decomposition $w = (s_i u_i)_{i=1}^k s_{k+1}$, such that $R(u_k, R(u_{k-1}, R(u_{k-2}, \ldots R(u_1, Q) \ldots))) = \emptyset$.

Intuition behind the proof: Whenever we consider a word w, the compact G-representation of it is $w = (s_i u_i)_{i=1}^k s_{k+1}$. Let

$$R(w, j) = R(u_j, R(u_{j-1}, R(u_{j-2}, \ldots R(u_1, Q) \ldots)))$$

for all j and words w. Observe that **(i)** $\Rightarrow$ **(ii)** is trivial. Intuitively, the proof for **(ii)** $\Rightarrow$ **(iii)** is by contradiction: Consider a word w in $\mathcal{L}(G)$ with edit distance above B from $\mathcal{L}(\mathcal{A}_N)$. Assume towards contradiction that $R(w,k)$ is not empty. Then there is a word $w' = (s'_i u_i)_{i=1}^k$ in $\mathcal{L}(\mathcal{A}_N)$ where each s'_i has length at most n. But $ed(w,w') \leq B$ by definition of compact G-representation (i.e. edit each s'_i to s_i separately), which is a contradiction. To show **(iii)** $\Rightarrow$ **(i)** we consider a word w where $R(w,k)$ is empty and we show that $w(\ell)$ (from compact G-representation) requires at least ℓ edits to $\mathcal{L}(\mathcal{A}_N)$. Inductively in j, there must be either at least ℓ edits on $(s_i u_i^{\ell})_{i=1}^{j}$ or $R(w,j)$ has been reached, by the property of reachability sets. Since $R(w,k)$ is empty, there must be ℓ edits on $w(\ell)$.

The equivalence of (i) and (ii) of Lemma 11 gives a bound on the maximum finite edit distance. The following lemma follows from Lemmas 9 and 11, and Theorem 2 for testing given thresholds.

Lemma 12. *For all* $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}, \mathcal{C}_2 \in \{\mathsf{DFA}, \mathsf{NFA}\}$ *the* $\mathsf{FED}(\mathcal{C}_1, \mathcal{C}_2)$ *problem is in ExpTime.*

4.2 Lower Bound

We have shown the exponential upper bound on the edit distance if it is finite. It is easy to define a family of CFGs only accepting an exponential length word, using repeated doubling and thus the edit distance can be exponential between DPDA and DFA. We also show that the inclusion problem reduces to the finite edit distance problem FED(DPDA, NFA) and get the following lemma.

Lemma 13. FED(DPDA, NFA) *is ExpTime-hard.*

We conjecture that, as for the case of language inclusion, for the finite edit distance problem the complexity of the DPDA/PDA to DFA problem matches the one for *NFA/DFA* to DFA.

Conjecture 14. FED(PDA, DFA) is coNP-complete.

5 Edit Distance to PDA

Observe that the threshold distance problem from DFA to PDA with the fixed threshold 0 and a fixed DFA recognizing Σ^* coincides with the universality problem for PDA. Hence, the universality problem for PDA, which is undecidable, reduces to TED(DFA, PDA). The universality problem for PDA reduces to FED(DFA, PDA) as well by the same argument as in Lemma 13. Finally, we can reduce the inclusion problem from DPDA in DPDA, which is undecidable, to TED(DPDA, DPDA) (resp., FED(DPDA, DPDA)). Again, we can use the same construction as in Lemma 13. In conclusion, we have the following proposition.

Proposition 15. *(1) For every class* $\mathcal{C} \in \{\mathsf{DFA}, \mathsf{NFA}, \mathsf{DPDA}, \mathsf{PDA}\}$*, the problems* $\mathsf{TED}(\mathcal{C}, \mathsf{PDA})$ *and* $\mathsf{FED}(\mathcal{C}, \mathsf{PDA})$ *are undecidable. (2) For every class* $\mathcal{C} \in \{\mathsf{DPDA}, \mathsf{PDA}\}$*, the problem* $\mathsf{FED}(\mathcal{C}, \mathsf{DPDA})$ *is undecidable.*

The results in (1) of Proposition 15 are obtained by reduction from the universality problem for PDA. However, the universality problem for DPDA is decidable. Still we show that $\mathsf{TED}(\mathsf{DFA}, \mathsf{DPDA})$ is undecidable. The overall argument is similar to the one in Section 3.2. First, we define a pushdown counterpart of nearly-deterministic NFA. A PDA $\mathcal{A} = (\Sigma, \Gamma, Q, S, \delta, F)$ is *nearly-deterministic* if $|S| = 1$ and $\delta = \delta_1 \cup \delta_2$, where δ_1 is a function and for every accepting run, the automaton takes a transition from δ_2 exactly once.

By carefully reviewing the standard reduction of the halting problem for Turing machines to the universality problem for pushdown automata [11], we observe that the PDA that appear in the reduction are nearly-deterministic.

Lemma 16. *The problem, given a nearly-deterministic PDA $\mathcal{A}_P$, decide whether $\mathcal{L}(\mathcal{A}_P) = \Sigma^*$, is undecidable.*

Using the same construction as in Lemma 7 we show a reduction of the universality problem for nearly-deterministic PDA to $\mathsf{TED}(\mathsf{DFA}, \mathsf{DPDA})$.

Proposition 17. *For every class $\mathcal{C} \in \{\mathsf{DFA}, \mathsf{NFA}, \mathsf{DPDA}, \mathsf{PDA}\}$, the problem $\mathsf{TED}(\mathcal{C}, \mathsf{DPDA})$ is undecidable.*

We presented the complete decidability picture for the problems $\mathsf{TED}(\mathcal{C}_1, \mathcal{C}_2)$, for $\mathcal{C}_1 \in \{\mathsf{DFA}, \mathsf{NFA}, \mathsf{DPDA}, \mathsf{PDA}\}$ and $\mathcal{C}_2 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$. To complete the characterization of the problems $\mathsf{FED}(\mathcal{C}_1, \mathcal{C}_2)$, with respect to their decidability, we still need to settle the decidability (and complexity) status of $\mathsf{FED}(\mathsf{DFA}, \mathsf{DPDA})$. We leave it as an open problem, but conjecture that it is coNP-complete.

Conjecture 18. $\mathsf{FED}(\mathsf{DFA}, \mathsf{DPDA})$ is coNP-complete.

References

1. Aho, A., Peterson, T.: A minimum distance error-correcting parser for context-free languages. SIAM J. of Computing **1**, 305–312 (1972)
2. Benedikt, M., Puppis, G., Riveros, C.: Regular repair of specifications. In: LICS 2011, pp. 335–344 (2011)
3. Benedikt, M., Puppis, G., Riveros, C.: Bounded repairability of word languages. J. Comput. Syst. Sci. **79**(8), 1302–1321 (2013)
4. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM **28**(1), 114–133 (1981). http://doi.acm.org/10.1145/322234.322243
5. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. ACM Trans. Comput. Log. 11(4) (2010)
6. Chatterjee, K., Henzinger, T.A., Ibsen-Jensen, R., Otop, J.: Edit distance for pushdown automata. CoRR abs/1504.08259 (2015). http://arxiv.org/abs/1504.08259
7. Chatterjee, K., Henzinger, T.A., Otop, J.: Nested weighted automata. CoRR abs/1504.06117 (2015). http://arxiv.org/abs/1504.06117 (to appear at LICS 2015)
8. Chatterjee, K., Ibsen-Jensen, R., Majumdar, R.: Edit distance for timed automata. In: HSCC 2014, pp. 303–312 (2014)

9. Gawrychowski, P.: Faster algorithm for computing the edit distance between SLP-compressed strings. In: Calderón-Benavides, L., González-Caro, C., Chávez, E., Ziviani, N. (eds.) SPIRE 2012. LNCS, vol. 7608, pp. 229–236. Springer, Heidelberg (2012)
10. Henzinger, T.A., Otop, J.: From model checking to model measuring. In: D'Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013 – Concurrency Theory. LNCS, vol. 8052, pp. 273–287. Springer, Heidelberg (2013)
11. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Adison-Wesley Publishing Company, Reading (1979)
12. Karp, R.: Mapping the genome: some combinatorial problems arising in molecular biology. In: STOC 93, pp. 278–285. ACM (1993)
13. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet physics doklady. **10**, 707–710 (1966)
14. Lifshits, Y.: Processing compressed texts: a tractability border. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 228–240. Springer, Heidelberg (2007)
15. Mohri, M.: Edit-distance of weighted automata: general definitions and algorithms. Intl. J. of Foundations of Comp. Sci. **14**, 957–982 (2003)
16. Okuda, T., Tanaka, E., Kasai, T.: A method for the correction of garbled words based on the levenshtein metric. IEEE Trans. Comput. **25**, 172–178 (1976)
17. Pighizzini, G.: How hard is computing the edit distance? Information and Computation **165**, 1–13 (2001)
18. Saha, B.: The dyck language edit distance problem in near-linear time. In: FOCS 2014, pp. 611–620 (2014)