Edit Distance for Pushdown Automata*

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

IST Austria

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 a pushdown automaton 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 [14] 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 and 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" 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) 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 automaton, 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

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

	Language inclusion		Edit distance (TED)	
Source \ Target	DFA	NFA	DFA	NFA
DPDA	PTime	**ExpTime-c (Th. 2 (2))**	**ExpTime-c (Th. 2 (1))**	
PDA				

Table 1. Complexity of language inclusion and edit distance. Our contributions are boldfaced.

specification. Among all implementations that violate a specification, the closer the implementation traces are to the specification, the better [6, 7, 11]. 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 [12]: (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 edit inclusion 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 DFA (or NFA) is ExpTime-complete. Thus we present a complete picture of the complexity of the TED problem, 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 more 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, we show that for a source language DPDA and target language DFA, there is an exponential lower bound on k, even if the answer to FED problem is YES (see Example 16). We present a matching exponential

	$\mathcal{C}_2 = \mathsf{DFA}$	$\mathcal{C}_2 = \mathsf{NFA}$	$\mathcal{C}_2 = \mathsf{DPDA}$	$\mathcal{C}_2 = \mathsf{PDA}$
$\mathcal{C}_1 \in \{\mathsf{DFA}, \mathsf{NFA}\}$	coNP-c [3]	PSpace-c [3]	open (Conj. 18)	
$\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$	coNP-hard [3] **in ExpTime (Th. 11)**	**ExpTime-c (Th. 11)**	**undecidable (Prop. 19)**	

Table 2. Complexity of $\mathsf{FED}(\mathcal{C}_1, \mathcal{C}_2)$. Our results are boldfaced. See Conjecture 23 for the open complexity problem.

upper bound on k for the FED problem from PDA to NFA. Finally, we also show that the FED problem is ExpTime-complete when the source language is given as a DPDA or PDA, and the target language is an NFA.

Our results are summarized in Table 1 and Table 2.

Related work. Algorithms for edit distance have been studied extensively for words [14, 1, 17, 18, 13, 16]. The edit distance between regular languages was studied in [2, 3], between timed automata in [8], and between straight line programs in [15, 10]. A near-linear time algorithm to approximate the edit distance for a word to a DYCK language has been presented in [19].

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, $\delta \subseteq Q \times \Sigma \times (\Gamma \cup \{\bot\}) \times Q \times \Gamma^*$ 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. 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 $(q, w[i], \bot, q', u') \in \delta$, 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 the accepting state and the stack is empty.

Language recognized by a PDA. Given a PDA $\mathcal{A}$, we define the *language recognized (or accepted) by* $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, as $\{w \in \Sigma^* : \mathcal{A} \text{ has an accepting run on } w\}$.

Finite automata. A *non-deterministic finite automaton* (NFA) is a pushdown automaton 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. For example: $ed(\{a\}^*, \{a,b\}^*) = 0$, while $ed(\{a,b\}^*, \{a\}^*) = \infty$ because for every n, we have $ed(\{b^n\}, \{a\}^*) = n$.

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 DFA, NFA, 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 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 $\mathcal{C}_2 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$.

The FED for regular languages has been studied in [3]. It has been showed 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. 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 ExpTime-complete. Finally, 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 ExpTime-complete. (2) For $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$, the inclusion of automata from $\mathcal{C}_1$ in NFA is 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, an reduce it (that stronger problem), to $\mathsf{TED}(\mathsf{DPDA}, \mathsf{DFA})$, which shows the ExpTime-hardness part of (1). We conclude this section with a brief discussion on parametrized complexity of TED in Section 3.3.

3.1 Upper bound

We present an 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 of Benedikt, Puppis and Riveros [2] for NFA.

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. *Given (1) a PDA $\mathcal{A}_P$; (2) an NFA $\mathcal{A}_N$; and (3) 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 $O(|\mathcal{A}_P| \cdot (t+2)^{|\mathcal{A}_N|})$.*

Proof. Let Q_N (resp., F_N) be the set of states (resp., accepting states) of $\mathcal{A}_N$. For $i \in \mathbb{N}$ and a word w, we define $T^i_w = \{s \in Q_N :$ there exists w' with $ed(w, w') = i$ such that $\mathcal{A}_N$ has a run on the word w' ending in $s\}$. For a pair of states $s, s' \in Q_N$ and $\alpha \in \Sigma \cup \{\epsilon\}$, we define $m(s, s', \alpha)$ as the minimum number of edits needed to apply to α so that $\mathcal{A}_N$ has a run on the resulting word from s' to s. For all $s, s' \in Q_N$ and $\alpha \in \Sigma \cup \{\epsilon\}$, we can compute $m(s, s', \alpha)$ in polynomial time in $|\mathcal{A}_N|$. For a state $s \in Q_N$ and a word w let $d^s_w = \min\{i \geq 0 \mid s \in T^i_w\}$, i.e., d^s_w is the minimal number of edits necessary to apply to w such that $\mathcal{A}_N$ reaches s upon reading the resulting word. We will argue that the following condition (*) holds: (*) $d^s_{wa} = \min_{s' \in Q_N}(d^{s'}_w + m(s, s', a))$. Consider a run witnessing d^s_{wa}. As shown by [20] we can split the run into two parts, one sub-run accepting w ending in s', for some s', and one sub-run accepting a starting in s'. Clearly, the sub-run accepting w has used $d^{s'}_w$ edits and the one accepting a has used $m(s, s', a)$ edits.

Let Q_P (resp., F_P) be the set of states (resp., accepting states) of the PDA $\mathcal{A}_P$. For all word w and state $q \in Q_P$ such that there is a run on w ending in q, we define $\mathsf{Impact}(w, q, \mathcal{A}_P, \mathcal{A}_N, t)$ as a pair (q, λ) in $Q_P \times \{0, 1, \ldots, t, \#\}^{|Q_N|}$, where λ is defined as follows: for every $s \in Q_N$ we have $\lambda(s) = d^s_w$ if $d^s_w \leq t$, and $\lambda(s) = \#$ otherwise. Clearly, the edit distance from $\mathcal{A}_P$ to $\mathcal{A}_N$ exceeds t if there is a word w and an accepting state q of $\mathcal{A}_P$ such that $\mathsf{Impact}(w, q, \mathcal{A}_P, \mathcal{A}_N, t)$ is a pair (q, λ) and for every $s \in F_N$ we have $\lambda(s) = \#$ (i.e., the word w is in $\mathcal{L}(\mathcal{A}_P)$ but any run of $\mathcal{A}_N$ ending in F_N has distance exceeding t).

We can now construct an *impact automaton*, a PDA $\mathcal{A}_I$, with state space $Q_P \times \{0, 1, \ldots, t, \#\}^{Q_N}$ and the transition relation defined as follows: A tuple $(\langle q, \lambda_1\rangle, a, \gamma, \langle q', \lambda_2\rangle, u)$ is a transition of $\mathcal{A}_I$ iff the following conditions hold:

1. the tuple projected to the first component of its state (i.e., the tuple (q, a, γ, q', u)) is a transition of $\mathcal{A}_P$, and
2. the second component λ_2 is computed from λ_1 using the condition (*), i.e., for every $s \in Q_N$ we have $\lambda_2(s) = \min_{s' \in Q_N}(\lambda_1(s') + m(s, s', a))$.

The initial states of $\mathcal{A}_I$ are $S_P \times \{\lambda_0\}$, where S_P are initial states of $\mathcal{A}_P$ and λ_0 is defined as follows. For every $s \in Q_N$ we have $\lambda_0(s) = \min_{s' \in S_N} m(s, s', \epsilon)$, where S_N are initial states of $\mathcal{A}_N$ (i.e., a start state of $\mathcal{A}_I$ is a pair of a start state of $\mathcal{A}_P$ together with the vector where the entry describing s is the minimum number of edits needed to get to the state s on the empty word). Also, the accepting states are $\{\langle q, \lambda\rangle \mid q \in F_P$ and for every $s \in F_N$ we have $\lambda(s) = \#\}$. Observe that for a run of $\mathcal{A}_I$ on w ending in (s, λ), the vector $\mathsf{Impact}(w, s, \mathcal{A}_P, \mathcal{A}_N, t)$ is precisely (s, λ). Thus, the PDA $\mathcal{A}_I$ accepts a word w iff the edit distance between $\mathcal{A}_P$ and $\mathcal{A}_N$ is above t. Since the size of $\mathcal{A}_I$ is $O(|\mathcal{A}_P| \cdot (t+2)^{|\mathcal{A}_N|})$ we obtain the desired result. □

Lemma 4. $\mathsf{TED}(\mathsf{PDA}, \mathsf{NFA})$ *is in* ExpTime.

Proof. Let $\mathcal{A}_P, \mathcal{A}_N$ and t be an instance of $\mathsf{TED}(\mathsf{PDA}, \mathsf{NFA})$, where $\mathcal{A}_P$ is a PDA, $\mathcal{A}_N$ is an NFA, and t is a threshold given in binary. By Lemma 3, we can reduce TED to the emptiness question of a PDA of the size $O(|\mathcal{A}_P| \cdot (t+2)^{|\mathcal{A}_N|})$. Since $|\mathcal{A}_P| \cdot (t+2)^{|\mathcal{A}_N|}$ is exponential in $|\mathcal{A}_P| + |\mathcal{A}_N| + t$ and the emptiness problem for PDA can be decided in time polynomial in their size [12], the result follows. □

3.2 Lower bound

Our ExpTime-hardness proof of $\mathsf{TED}(\mathsf{DPDA}, \mathsf{DFA})$ extends the idea from [2] that shows PSpace-hardness of the edit distance for DFA. In the standard proof of PSpace-hardness of the universality problem for NFA [12], the NFA recognizes a language of all words that do not encode valid computation of a given Turing machine M working on the tape bounded by a given (in unary) n. Observe that such an NFA checks the following three 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, and (3) the tape's cells are changed only by M, i.e., they do not change values spontaneously. While conditions (1) and (2) can be checked by a DFA of polynomial size, condition (3) can be encoded by a polynomial NFA but not polynomial DFA. However, to check (3) the automaton has to make only a single non-deterministic choice to pick a position in the encoding of the computation, which violates (3), i.e., the value at that position is different than the value $n+1$ 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 (3) into a deterministic automaton $\mathcal{A}_D$ by encoding such a non-deterministic pick using an external letter. Since we need only at most one external symbol, we can show that $\mathcal{L}(\mathcal{A}_N) = \Sigma^*$ iff $ed(\Sigma^*, \mathcal{L}(\mathcal{A}_D)) = 1$. This suggests the following definition:

Definition 5. *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 6. *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. We show the ExpTime-hardness of the above problem by reduction from the halting problem of a fixed alternating Turing machine M in linear space on a given input. The halting problem of a fixed alternating Turing machine M working on a tape bounded by n, where n is the length of the input, is ExpTime-complete [5].

We w.l.o.g. assume that existential and universal transitions of M alternate. The main idea is to construct a language L of words that encode valid terminating computation

trees of M. We encode configurations of M as words of length $n+3$ of the form $\#\{0,1\}^i q\{0,1\}^{n-i}\#$, where q is a state of M. We use brackets to encode computation tree in the following way. Consider a computation in which the first transition is existential from C_1 to C_2 and the second transition is universal from C_2 to $C_{3,0}$ and $C_{3,1}$. The computation is encoded as follows, where the superscript R denotes reversal of a word:

$$\#C_1\#C_2^R\#\,(C_{3,0}\ldots\$)\,(C_{3,1}\ldots\$)\,.$$

We define automata $\mathcal{A}_N$ and $\mathcal{A}_P$. The automaton $\mathcal{A}_N$ is a nearly deterministic NFA that recognizes only (but not all) words not encoding valid computation trees of M. More precisely, $\mathcal{A}_N$ accepts in four cases: (1) $\mathcal{A}_N$ accepts if the word does not encode a tree (the parentheses may not match as the automaton cannot check it) of computation as presented above. (2) $\mathcal{A}_N$ accepts if the initial configuration is different than the intended input. (3) $\mathcal{A}_N$ checks that the successive configurations, i.e., those that result from existential transitions or left-branch universal transitions (like C_2 to $C_{3,0}$), are valid. The right-branch universal transitions, which are preceded by the word "(", are not checked by $\mathcal{A}_N$. E.g. the configuration $C_{3,1}$ does not have a preceding configuration for $\mathcal{A}_N$. Finally, (4) $\mathcal{A}_N$ accepts words in which at least one final configuration, a configuration followed by \$, is not final for M.

Next, we define $\mathcal{A}_P$ as a pushdown automaton that accepts words in which parentheses match and right-branch universal transitions are consistent. E.g. it checks consistency of transition C_2 to $C_{3,1}$. Observe that the automaton can put on the stack C_2^R and pull from the stack the configuration C_2 (not reversed) letter-by-letter to compare it with $C_{3,1}$. While the automaton processes the subword $(C_{3,0}\ldots\$)$, it can use its stack to check consistency of universal transitions in that word. We assumed that M does not have consecutive universal transitions, therefore $\mathcal{A}_P$ does not need to check consistency of $C_{3,1}$ its successive configuration. By the construction, we have $L = \mathcal{L}(\mathcal{A}_P) \cap \mathcal{L}(\mathcal{A}_N)^c$ and M halts on the given input if and only if $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_N)$ fails. □

Now, the following lemma, which is (2) of Theorem 2, follows from Lemma 6.

Lemma 7. *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 6. □

Now, we show that the inclusion problem of DPDA in nearly-deterministic NFA reduces to TED(DPDA, DFA).

Lemma 8. TED(DPDA, DFA) *is* ExpTime*-hard.*

Proof. To show ExpTime-hardness of TED(DPDA, DFA), we reduce the inclusion problem of DPDA in nearly-deterministic NFA to TED(DPDA, DFA). Consider a DPDA $\mathcal{A}_P$ and a nearly-deterministic NFA $\mathcal{A}_N$ over an alphabet Σ. Without loss of generality we assume that letters on even positions are $\$ \in \Sigma$ and \$ do not appear on the odd positions. Let $\delta = \delta_1 \cup \delta_2$ be the transition relation of $\mathcal{A}_N$, where δ_1 is a function and along each accepting run, $\mathcal{A}_N$ takes exactly one transition from δ_2. We transform the NFA $\mathcal{A}_N$ to a DFA $\mathcal{A}_D$ by extending the alphabet Σ with external letters $\{1,\ldots,|\delta_2|\}$. On letters from Σ, the automaton $\mathcal{A}_D$ takes transitions from δ_1. On a letter $i \in \{1,\ldots,|\delta_2|\}$, the automaton $\mathcal{A}_D$ takes the i-th transition from δ_2.

We claim that $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_N)$ iff $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_D)) = 1$. Every word $w \in \mathcal{L}(\mathcal{A}_D)$ contains a letter $i \in \{1, \ldots, |\delta_2|\}$, which does not belong to Σ. Therefore, $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_D)) \geq 1$. But, if we substitute letter i by the letter in the i-th transition of δ_2, we get a word from $\mathcal{L}(\mathcal{A}_N)$. If we simply delete the letter i, we get a word which does not belong to $\mathcal{L}(\mathcal{A}_N)$ as it has letter \$ on an odd position. Therefore, $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_D)) \leq 1$ implies $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_N)$. Finally, consider a word $w' \in \mathcal{L}(\mathcal{A}_N)$. The automaton $\mathcal{A}_N$ has an accepting run on w', which takes exactly once a transition from δ_2. Say the taken transition is the i-th transition and the position in w' is p. Then, the word w, obtained from w' by substituting the letter at position p by letter i, is accepted by $\mathcal{A}_D$. Therefore, $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_N)$ implies $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_D)) \leq 1$. Thus we have $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_N)$ iff $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_D)) = 1$. □

3.3 Parameterized complexity

Problems of high complexity can be practically viable if the complexity is caused by a parameter, which tends to be small in the applications. In this section we discuss the dependence of the complexity of TED based on its input values.

Proposition 9. *(1) There exist a threshold $t > 0$ and a* DPDA *$\mathcal{A}_P$ such that the variant of* TED(DPDA, DFA)*, in which the threshold is fixed to t and DPDA is fixed to $\mathcal{A}_P$, is still ExpTime-complete. (2) The variant of* TED(PDA, NFA)*, in which the threshold is given in unary and* NFA *is fixed, is in PTime.*

Proof. (1): The inclusion problem of DPDA in nearly-deterministic NFA is ExpTime-complete even if a DPDA is fixed (Lemma 6). Therefore, the reduction in Lemma 8 works for threshold 1 and fixed DPDA.
(2): In the reduction from Lemma 3, the resulting PDA has size $|\mathcal{A}_P| \cdot (t+2)^{|\mathcal{A}_N|}$, where $\mathcal{A}_P$ is a PDA, $\mathcal{A}_N$ is an NFA and t is a threshold. If $\mathcal{A}_N$ is fixed and t is given in unary, then $|\mathcal{A}_P| \cdot (t+2)^{|\mathcal{A}_N|}$ is polynomial in the size of the input and we can decide its non-emptiness in polynomial time. □

Conjecture 10 completes the study of the parameterized complexity of TED.

Conjecture 10. The variant of TED(PDA, NFA), in which the threshold is given in binary and NFA is fixed, is in PTime.

4 Finite edit distance from pushdown to regular languages

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

Theorem 11. *(1) For $\mathcal{C}_1 \in \{$DPDA, PDA$\}$ and $\mathcal{C}_2 \in \{$DFA, 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 $\mathcal{C}_1 \in \{$DPDA, PDA$\}$ we have* FED$(\mathcal{C}_1,$ NFA$)$ *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 FED(DPDA, 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 upto 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 for the problem. Given a language $\mathcal{L}$, we define $\mathsf{prefix}(\mathcal{L}) = \{u : u \text{ is a prefix of some word from } \mathcal{L}\}$. We call an automaton $\mathcal{A}$ *safety* if every state of $\mathcal{A}$ is accepting. Note that for every NFA $\mathcal{A}_N$, the language $\mathsf{prefix}(\mathcal{L}(\mathcal{A}_N))$ 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 12. *Let $\mathcal{A}_P$ be a PDA and $\mathcal{A}_N$ an NFA. Then the following inequalities hold:*

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

Proof. Since $\mathcal{L}(\mathcal{A}_N) \subseteq \mathsf{prefix}(\mathcal{L}(\mathcal{A}_N))$, we have

$$ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_N)) \geq ed(\mathcal{L}(\mathcal{A}_P), \mathsf{prefix}(\mathcal{L}(\mathcal{A}_N)))$$

as the latter is the minimum over a larger set by definition.

Hence, we only need to show the other inequality. First observe that for every $w \in \mathsf{prefix}(\mathcal{L}(\mathcal{A}_N))$, upon reading w, the automaton $\mathcal{A}_N$ can reach a state from which an accepting state is reachable and thus, an accepting state can be reached in at most $|\mathcal{A}_N|$ steps. Therefore, for every $w \in \mathsf{prefix}(\mathcal{L}(\mathcal{A}_N))$ there exists w' of length bounded by $|\mathcal{A}_N|$ such that $ww' \in \mathcal{L}(\mathcal{A}_N)$. It follows that $ed(\mathcal{L}(\mathcal{A}_P), \mathsf{prefix}(\mathcal{L}(\mathcal{A}_N))) \geq ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_N)) - |\mathcal{A}_N|$. □

Remark 13. Consider an NFA $\mathcal{A}_N$ recognizing a language such that $\mathsf{prefix}(\mathcal{L}(\mathcal{A}_N)) = \Sigma^*$. For every PDA $\mathcal{A}_P$, the edit distance $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_N))$ is bounded by $|\mathcal{A}_N|$.

Let us recall the pumping lemma for context-free languages [12]. It states that for a given PDA $\mathcal{A}_P$, there is a constant η such that every word $w \in \mathcal{L}(\mathcal{A}_P)$ of length greater than η can be partitioned into $v_1 \cdot u_1 \cdot v_2 \cdot u_2 \cdot v_3$ with $|v_1 v_2 v_3| \leq \eta$ such that for every ℓ we have $v_1 \cdot u_1^\ell \cdot v_2 \cdot u_2^\ell \cdot v_3 \in \mathcal{L}(\mathcal{A}_P)$. The minimum such constant η for $\mathcal{A}_P$ is called the *pumping constant* for $\mathcal{A}_P$.

Lemma 14. *Let $\mathcal{A}_P$ be a PDA, $\mathcal{A}_N$ be a safety NFA and let η be the pumping constant for $\mathcal{A}_P$. If the edit distance $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_N)) > \eta + 2 \cdot |\mathcal{A}_N|$, then $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_N)) = \infty$.*

Proof. Let Q_N be the set of states of $\mathcal{A}_N$. For a word w and a set of states $Q' \subseteq Q_N$ we define $R(w, Q') \subseteq Q_N$ as the set of states that are reachable from the states reachable from Q' upon reading w. We will use the following property (*) of safety NFA: (*) if $R(w, Q_N)$ is empty, then for any word w'' containing c occurrences of w as disjoint sub-words (for instance w^c), $ed(w'', \mathcal{A}_N) \geq c$. Let $\ell = \eta + 2 \cdot |\mathcal{A}_N| - 2$. Consider a word $w \in \mathcal{L}(\mathcal{A}_P)$ such that $ed(w, \mathcal{L}(\mathcal{A}_N)) > \ell$. By the pumping lemma, we can write w as $v_1 \cdot u_1 \cdot v_2 \cdot u_2 \cdot v_3$, such that $|v_1 v_2 v_3| \leq \eta$. For convenience of notation, we will consider that the state space of $\mathcal{A}_N$ is pruned in the sense that all states can be reached in some way from some start state (other states can simply be removed without affecting the language).

Our proof will proceed as follows: First we will show that at least one of $R(u_1, Q_N)$ and $R(u_2, R(u_1, Q_N))$ must be empty, and then use that together with the property (*) to show that for every $c > 1$, the word $v_1 \cdot u_1^c \cdot v_2 \cdot u_2^c \cdot v_3$ (which is in $\mathcal{A}_P$ by the pumping lemma) has edit distance to $\mathcal{L}(\mathcal{A}_N)$ at least c.

We first argue that one of the sets $R(u_1, Q_N)$ and $R(u_2, R(u_1, Q_N))$ is empty. Assume towards contradiction that they are both non-empty. Then w could be accepted by $\mathcal{A}_N$ with

at most ℓ edits by a run as follows: Starting from a start state s_1 of $\mathcal{A}_N$ go to a state s_2 that has an accepting run on u_1. Let v_1' be any shortest word on which $\mathcal{A}_N$ can go from state s_1 to state s_2. Next, follow a run from s_2 accepting u_1 ending in some state s_3. Then go to a state s_4 from which there exists an accepting run on u_2. Let v_2' be any shortest word on which $\mathcal{A}_N$ can go from state s_3 to state s_4. Then follow the run accepting u_2. Because both $R(u_1, Q_N)$ and $R(u_2, R(u_1, Q_N))$ are non-empty, this can be done in some way. Hence, the word $w' = v_1'u_1v_2'u_2$ is in $\mathcal{L}(\mathcal{A}_N)$. We can then edit w' into w using ℓ edits, by editing (1) v_1' into v_1; and (2) v_2' into v_2; and (3) the empty word at the end of w' into v_3. In the worst case (i) $|v_1| = |v_2| = 0$; and (ii) $|v_1'| = |v_2'| = |\mathcal{A}_N| - 1$; and (iii) $|v_3| = \eta$ and the edit distance is ℓ.

Next consider the word $v_1 \cdot u_1^c \cdot v_2 \cdot u_2^c \cdot v_3$, for some number $c > 1$, which is in $\mathcal{L}(\mathcal{A}_P)$ by the pumping lemma. We will argue that the edit distance from that word to $\mathcal{L}(\mathcal{A}_N)$ is at least c. Towards contradiction, assume that $ed(v_1 \cdot u_1^c \cdot v_2 \cdot u_2^c \cdot v_3, \mathcal{L}(\mathcal{A}_N)) < c$ and consider a run of $\mathcal{A}_N$ witnessing that inequality. Consider first the prefix $v_1 \cdot u_1^c$. Either there are c edits, one for each repetition of u_1, or the run has ended up in $R(u_1, Q_N)$ after some repetition of u_1 (and thus stayed there). In the first case, we get a contradiction and thus only need to consider the second case. Note that $R(u_1, Q_N)$ is then non-empty in this case and thus $R(u_2, R(u_1, Q_N))$ is empty. The run is still in $R(u_1, Q_N)$ after $v_1 \cdot u_1^c \cdot v_2$, but then it needs to make at least 1 edit to each of the c repetitions of the words u_2, because of the emptiness of $R(u_2, R(u_1, Q_N))$. Hence, the run needs c edits, which is a contradiction. This completes the proof. □

The following lemma follows from Lemmas 12 and 14 and the fact, that the pumping constant is exponentially bounded in the size of the PDA [12].

Lemma 15. *For all $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}, \mathcal{C}_2 \in \{\mathsf{DFA}, \mathsf{NFA}\}$ the following conditions hold: (1) For all $\mathcal{A}_1 \in \mathcal{C}_1, \mathcal{A}_2 \in \mathcal{C}_2$, if $ed(\mathcal{A}_1, \mathcal{A}_2)$ is finite, then it is exponentially bounded in $\mathcal{A}_1$ and linearly bounded in $\mathcal{A}_2$. (2) $\mathsf{FED}(\mathcal{C}_1, \mathcal{C}_2)$ is in ExpTime.*

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 $|\mathcal{A}_P| + |\mathcal{A}_N|$.

4.2 Lower bound

We have shown the exponential upper bound on the edit distance if it is finite. Example 16 shows that the exponential bound is asymptotically tight, which is in contrast to the edit distance from an NFA to another NFA, where the edit distance, if finite, is polynomially bounded [2]. Finally, in Lemma 17 we show that for the target language given by NFA, the problem FED is as hard as TED.

Example 16 (The edit distance of a DPDA to a DFA can be exponential.). Given $n > 0$, consider a context free grammar $\mathcal{G}_n = (\{S, A_1, \ldots, A_n\}, \{S \to A_n, A_1 \to a\} \cup \{A_{i+1} \to A_iA_i : i = 1, \ldots, n-1\})$. The grammar $\mathcal{G}_n$ derives a single word a^{2^n}. Let $\mathcal{A}_R$ be a DFA that recognizes the language b^*. Now, the size of grammar $\mathcal{G}_n$ is polynomial in n and there exists a DPDA $\mathcal{A}_n$ equivalent to $\mathcal{G}_n$ such that the size of $\mathcal{A}_n$ is $O(n)$. The edit distance $ed(\mathcal{L}(\mathcal{A}_n), \mathcal{L}(\mathcal{A}_R))$ is 2^n. Hence, the edit distance of a DPDA to a DFA can be exponential in the size of the DPDA.

Lemma 17. $\mathsf{FED}(\mathsf{DPDA}, \mathsf{NFA})$ *is ExpTime-hard.*

Proof. We show that the inclusion problem of DPDA in NFA, which is ExpTime-hard by Lemma 6 reduces to FED(DPDA, NFA). Consider a DPDA $\mathcal{A}_P$ and an NFA $\mathcal{A}_N$. We define $\widehat{\mathcal{L}} = \{\#w_1\# \dots \#w_k\# : k \in \mathbb{N}, w_1, \dots, w_k \in \mathcal{L}\}$. Observe that either $\widehat{\mathcal{L}_1} \subseteq \widehat{\mathcal{L}_2}$ or $ed(\widehat{\mathcal{L}_1}, \widehat{\mathcal{L}_2}) = \infty$. Therefore, $ed(\widehat{\mathcal{L}_1}, \widehat{\mathcal{L}_2}) < \infty$ iff $\mathcal{L}_1 \subseteq \mathcal{L}_2$. In particular, $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_N)$ iff $ed(\widehat{\mathcal{L}(\mathcal{A}_P)}, \widehat{\mathcal{L}(\mathcal{A}_N)}) < \infty$. Observe that in polynomial time we can transform $\mathcal{A}_P$ (resp., $\mathcal{A}_N$) to a DPDA $\widehat{\mathcal{A}_P}$ (resp., an NFA $\widehat{\mathcal{A}_N}$) recognizing $\widehat{\mathcal{L}(\mathcal{A}_P)}$ (resp., $\widehat{\mathcal{L}(\mathcal{A}_P)}$). It suffices to add transitions from all final states to all initial states with the letter $\#$, i.e., $\{(q, \#, s) : q \in F, s \in S\}$. □

Conjecture 18. FED(DPDA, DFA) is ExpTime-hard.

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 Theorem 17. Finally, we can reduce the inclusion problem of DPDA in DPDA, which is undecidable, to TED(DPDA, DPDA) (resp., FED(DPDA, DPDA)). Again, we can use the same construction as in Theorem 17. In conclusion, we have the following proposition.

Proposition 19. *(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 19 are obtained by reduction from the universality problem for PDA. However, the universality problem for DPDA is decidable. Still we show that TED(DFA, DPDA) is undecidable. The overall argument is similar to the one in Section 3.2. First, we define nearly-deterministic PDA, a pushdown counterpart of nearly-deterministic NFA.

Definition 20. *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 [12], we observe that the PDA that appear as the product of the reduction are nearly-deterministic.

Lemma 21. *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 Theorem 8 we can show a reduction of the universality problem for nearly-deterministic PDA to TED(DFA, DPDA).

Proposition 22. *For every class* $\mathcal{C} \in \{\mathsf{DFA}, \mathsf{NFA}, \mathsf{DPDA}, \mathsf{PDA}\}$*, the problem* $\mathsf{TED}(\mathcal{C}, \mathsf{DPDA})$ *is undecidable.*

Proof. We show that TED(DFA, DPDA) (resp., FED(DFA, PDA)) is undecidable as it implies undecidability of the rest of the problems. The same construction as in the proof of Theorem 8 shows a reduction of the universality problem for nearly-deterministic DPDA to TED(DFA, DPDA). □

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, which is likely to be undecidable.

Conjecture 23. $\mathsf{FED}(\mathsf{DFA}, \mathsf{DPDA})$ is undecidable.

6 Conclusions

In this work we consider the edit distance problem for PDA and its subclasses and present a complete decidability and complexity picture for the TED problem. We leave some open conjectures about the parametrized complexity of the TED problem, and the complexity of FED problem when the target is a DPDA or a DFA. While in this work we count the number of edit operations, a different notion is to measure the average number of edit operations. The average-based measure is undecidable in many cases even for finite automata, and in cases when it is decidable reduces to mean-payoff games on graphs [4]. Since mean-payoff games on pushdown graphs are undecidable [9], most of the problems related to the edit distance question for average measure for DPDA and PDA are likely to be undecidable.

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'11. 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. Benedikt, M., Puppis, G., Riveros, C.: The per-character cost of repairing word languages. Theor. Comput. Sci. 539, 38–67 (2014), http://dx.doi.org/10.1016/j.tcs.2014.04.021
5. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (Jan 1981), http://doi.acm.org/10.1145/322234.322243
6. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. ACM Trans. Comput. Log. 11(4) (2010)
7. Chatterjee, K., Henzinger, T.A., Otop, J.: Nested weighted automata (2014), http://repository.ist.ac.at/170/
8. Chatterjee, K., Ibsen-Jensen, R., Majumdar, R.: Edit distance for timed automata. In: HSCC'14. pp. 303–312 (2014)
9. Chatterjee, K., Velner, Y.: Mean-payoff pushdown games. In: LICS. pp. 195–204 (2012)
10. Gawrychowski, P.: Faster algorithm for computing the edit distance between slp-compressed strings. In: SPIRE'12. pp. 229–236 (2012)
11. Henzinger, T.A., Otop, J.: From model checking to model measuring. In: CONCUR'13. pp. 273–287 (2013)
12. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Adison-Wesley Publishing Company, Reading, Massachusets, USA (1979)
13. Karp, R.: Mapping the genome: some combinatorial problems arising in molecular biology. In: STOC 93. pp. 278–285. ACM (1993)
14. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet physics doklady. vol. 10, pp. 707–710 (1966)
15. Lifshits, Y.: Processing compressed texts: A tractability border. In: Combinatorial Pattern Matching. pp. 228–240. Springer (2007)

16. Mohri, M.: Edit-distance of weighted automata: general definitions and algorithms. Intl. J. of Foundations of Comp. Sci. 14, 957–982 (2003)
17. 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)
18. Pighizzini, G.: How hard is computing the edit distance? Information and Computation 165, 1–13 (2001)
19. Saha, B.: The dyck language edit distance problem in near-linear time. In: FOCS'14. pp. 611–620 (2014)
20. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM pp. 168–173 (1974)