Institute of Science and Technology

Edit Distance for Pushdown Automata

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

Technical Report No. IST-2015-334-v1+1
Deposited at 05 May 2015 08:04
http://repository.ist.ac.at/334/1/report.pdf

IST Austria (Institute of Science and Technology Austria)
Am Campus 1
A-3400 Klosterneuburg, Austria

Edit Distance for Pushdown Automata*

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

IST Austria

April 29, 2015

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 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) 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 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].

*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.

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

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

	$\mathcal{C}_2 = \mathsf{DFA}$	$\mathcal{C}_2 = \mathsf{NFA}$	$\mathcal{C}_2 = \mathsf{DPDA}$	$\mathcal{C}_2 = \mathsf{PDA}$
$\mathcal{C}_1 \in \{\mathsf{DFA}, \mathsf{NFA}\}$	coNP-c [3]	PSpace-c [3]	open (Conj. 23)	
$\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$	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 18 for the open complexity problem.

	$\mathcal{C}_2 = \mathsf{DFA}$	$\mathcal{C}_2 = \mathsf{NFA}$	$\mathcal{C}_2 = \mathsf{DPDA}$	$\mathcal{C}_2 = \mathsf{PDA}$
$\mathcal{C}_1 \in \{\mathsf{DFA}, \mathsf{NFA}\}$	PSpace-c [2]		**undecidable (Prop. 22)**	
$\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$	**ExpTime-c (Th. 2 (1))**		undecidable	

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

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 distance 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, 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, the edit distance can be exponential between DPDAs and DFAs. We present a matching exponential 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 Tables 1, 2 and 3.

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 from $Q \times \Sigma \times (\Gamma \cup \{\bot\})$ 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 $(q, w[i], \bot, q', u') \in \delta$, 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. 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) $p : v \to zu$, where $v, z, u \in V$; or (2) $p : v \to \alpha$, where $v \in V$ and $\alpha \in \Sigma$; or (3) $p : s \to \epsilon$.

Languages generated by CFGs. Fix a CFG $G = (\Sigma, V, s, P)$. We define derivations $\to_G$ as a relation on $(\Sigma \cup V)^* \times (\Sigma \cup V)^*$ as follows: $w \to_G w'$ 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 $v \to u$ is a production from G. We define $\to_G^*$ as the transitive closure of $\to_G$. The *language generated by* G, denoted by $\mathcal{L}(G) = \{w \in \Sigma^* \mid s \to_G^* w\}$ is the set of words that can be derived from s. It is well-known [12] that CFGs and PDAs are language-wise polynomial equivalent (i.e., there is a polynomial time procedure that, given a PDA, outputs a CFG of the same language and vice versa).

Derivation trees of CFGs. Fix a CFG (Σ, V, s, P). The CFG defines a (typically infinite) set of *derivation trees*. A derivation tree is an ordered tree[1] where (1) each leaf is associated with an element of $\{\Sigma \cup V\}$; and (2) each internal node q is associated with a non-terminal $v \in V$ and production rule $p : v \to w$, such that v has $|w|$ children and the i'th child, for each i, is associated with $w[i]$ (and some production rule, in case the i'th child is not a leaf). For a node q of a derivation tree we let $T(q)$ be the sub-tree under q. A derivation tree T defines a string $w(T)$ over $\{\Sigma \cup V\}$ formed by the leaves in order.

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

[1] that is, the children of each internal node has an order

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

We study the FED problem for all combinations of source and target languages given by DFA, NFA, 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 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 ExpTime-complete. (2) For $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$, the language inclusion problem 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, and 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 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. *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 3 implies the following:

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

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 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 [12] 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-szie 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 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. 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 [5]. 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 a 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 fixed input. We encode a single configuration of M_U as words 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 and it is accepting if every leaf is an accepting configuration. We encode computation tree 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 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, 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 transitions from C_2 to C_4. The automaton $\mathcal{A}_P$ pushes even configurations (e.g., C_2^R) on the stack (which are reversed) 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 $(C_3 \dots \$)$, it can use its stack to check consistency of universal transitions in that word. We assumed that M_U does not have consecutive universal transitions. That 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 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. □

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

Lemma 7. *The language inclusion problem from* DPDA *to* 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, which is ExpTime-complete by Lemma 6, reduces to $\mathsf{TED}(\mathsf{DPDA}, \mathsf{DFA})$. In the reduction, we transform a nearly-deterministic NFA $\mathcal{A}_N$ 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. 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, $ed(w, \mathcal{L}(\mathcal{A}_D)) \leq 1$ implies $w \in \mathcal{A}_N$. Therefore, $ed(\mathcal{L}(\mathcal{A}_P), \mathcal{L}(\mathcal{A}_D)) \leq 1$ if and only if $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_N)$.

Finally, using Lemma 6, we can show the lower bound in (1) of Theorem 2.

Lemma 8. $\mathsf{TED}(\mathsf{DPDA}, \mathsf{DFA})$ *is* ExpTime*-hard.*

Proof. To show ExpTime-hardness of $\mathsf{TED}(\mathsf{DPDA}, \mathsf{DFA})$, we reduce the inclusion problem of DPDA in nearly-deterministic NFA to $\mathsf{TED}(\mathsf{DPDA}, \mathsf{DFA})$. 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 $\diamondsuit \in \Sigma$ and $\diamondsuit$ 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 $\{1, \dots, |\delta_2|\}$. On letters from Σ, the automaton $\mathcal{A}_D$ takes transitions from δ_1. On a letter $i \in \{1, \dots, |\delta_2|\}$, 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 $i \in \{1, \dots, |\delta_2|\}$, 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 $\diamondsuit$ 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* $\mathsf{TED}(\mathsf{DPDA}, \mathsf{DFA})$*, in which the threshold is fixed to t and DPDA is fixed to $\mathcal{A}_P$, is still* ExpTime*-complete. (2) The variant of* $\mathsf{TED}(\mathsf{PDA}, \mathsf{NFA})$*, 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 $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$ 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 $\mathcal{C}_1 \in \{\mathsf{DPDA}, \mathsf{PDA}\}$ we have $\mathsf{FED}(\mathcal{C}_1, \mathsf{NFA})$ 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 of 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 a automata is not necessarily total, i.e. some states might not have an outgoing transition for some input symbols, and thus a safety automata does not necessarily accept all words. 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 FED(PDA, 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 $|\mathcal{A}_N|$.*

In the remainder of this section we work with context-free grammars (CFGs) instead of PDAs. There are polynomial-time transformations between CFGs and PDAs that preserve the generated language; switching from PDAs to CFGs is made only to simplify the proofs. 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 $\mathcal{L}(G)$.
2. $|w(0)| = \sum_{i=1}^{k+1} |s_i| \leq 2^T$ and $k \leq 2^{T+1} - 2$.

Lemma 14. *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 applications of the idea behind the pumping lemma, until the part which is not pumped is small.

Proof. Fix some ℓ and consider some word w in $\mathcal{L}(G)$ and some derivation tree $d(w)$ for w. We will greedily construct a compact G-representation, using that we do not give bounds on $|u_i|$.

Greedy traversal and the first property. The idea is to consider nodes of $d(w)$ in a depth first pre-order traversal (ensuring that when we consider some node we have already considered its ancestors). When we consider some node v, continue with the traversal, unless there exists an descendant u of v, such that $T(v) = T(u)$. If there exists such an descendant, let u' be the bottom-most descendant (pick an arbitrary one if there are more than one such bottom-most descendants) such that $A := T(v) = T(u')$. We say that (v, u') forms a *pump pair* of w. We can then write $w(T(v))$ as $sw(T(u'))s'$ (and hence $A \to_G^* sAs'$), for some s and s' in the obvious way and s and s' will correspond to u_i and u_j respectively for some $i < j$ (i and j are defined by the traversal that we have already assigned $i - 1$ u's then we first visit v and then assign s as the u_i and then we return to the parent of v, we have assigned $j - 1$ u's and assign s' to be u_j). Observe that $A \to_G^* u_i A u_j$ implies that $A \to_G^* u_i^{\ell} A u_j^{\ell}$. Hence, $w(\ell)$ is in $\mathcal{L}(G)$, showing the first property of compact G-representation. This furthermore defines a derivation tree $d(w(0))$ for $w(0)$, which is the same as $d(w)$, except that for each pump pair (v, u'), the node v is replaced with the sub-tree $T(u')$ with root u'. So as to not split u_i or u_j up, we continue the traversal on u', which, then it is finished, continues the traversal in the parent of v, having finished with v. Notice that this ensures that each node is in at most one pump pair.

The second property. Next, consider the word $w(0)$. Observe that in $d(w(0))$, there is at most one occurrence of each non-terminal in each path to the root, since we visited all nodes of $d(w(0))$ in our defining traversal and were greedy. Hence, the height is at most T and thus, since the tree is binary, it has at most 2^{T-1} many leaves, which is then a bound on $|w(0)| = \sum_{i=1}^{k+1} |s_i|$. Notice that each node of $d(w(0))$, being a subset of $d(w)$, is in at most 1 pump pair of w. On the other hand for each pump pair (v, u') of w, we have that u' is a node of $d(w(0))$ by construction. Hence, w has at most $2^T - 1$ many pump pairs. Since each pump pair gives rise to at most 2 u_i's, we have that $k \leq 2^{T+1} - 2$. □

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 $\delta(q, a) \subseteq Q'$. Consider any word w with edit distance strictly less than ℓ from u^{ℓ}. 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 15. *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 $R(u_k, R(u_{k-1}, R(u_{k-2}, \ldots R(u_1, Q) \ldots))) = \emptyset$.

Before we proceed with the proof of Lemma 15 we motivate condition (3) from its statement.

The necessity of the recursive applications of the R operator. In the next lemma, we argue that the edit distance is either finite or 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 $R(u_k, R(u_{k-1}, R(u_{k-2}, \dots R(u_1, Q)\dots))) = \emptyset$. We will first argue by example that the nested applications of the R function is necessary. Consider for instance the alternate requirement that at least one of $R(u_i, Q)$ is empty, for some i. This alternate requirement would not capture that the regular language $a^*|b^*$ has infinite edit distance to the pushdown language $\{a^n \# b^n \mid n \in \mathbb{N}\}$ — for any word in the pushdown language $w = a^n \# b^n$, for some fixed n, a compact G-representation of w is $u_1 = a^n$, $s_2 = \#$ and $u_2 = b^n$ (and the remaining words are empty). But clearly $R(u_1, Q)$ and $R(u_2, Q)$ are not empty since both strings are in the regular language. On the other hand $R(u_2, R(u_1, Q))$ is empty.

of Lemma 15. The implication **(i)** $\Rightarrow$ **(ii)** is trivial.

For the implication **(ii)** $\Rightarrow$ **(iii)** consider a word $w \in \mathcal{L}(G)$ with $ed(w, \mathcal{L}(\mathcal{A}_N)) > B$ and its compact G representation $w = (s_i u_i)_{i=1}^{k} s_{k+1}$ (which exists due to Lemma 14). We claim that $R(u_k, R(u_{k-1}, R(u_{k-2}, \dots R(u_1, Q)\dots))) = \emptyset$. Towards contradiction, assume that $R(u_k, R(u_{k-1}, R(u_{k-2}, \dots R(u_1, Q)\dots))) \neq \emptyset$ and we will construct a run spelling a word w' in $\mathcal{A}_N$ which have edit distance at most B to w. The description of the run is iteratively in i. First, go to some state q_i such that there exists a run on u_i. This can be done in n steps spelling some word s'_i. Afterwards follow the run on u_i and go to the next iteration. This run spells the word $w' := (s'_i u_i)_{i=1}^{k}$. All the choices of q_i's can be made since $R(u_k, R(u_{k-1}, R(u_{k-2}, \dots R(u_1, Q)\dots))) \neq \emptyset$. Also, since $\mathcal{A}_N$ is a safety automata, this run is accepting. To edit w' into w change each s'_i into s_i and insert s_{k+1} at the end. In the worst case, each s_i is empty except for $i = k+1$ and in that case it requires $k \cdot n + |w(0)| \leq B$ edits for deleting each s'_i and inserting s_{k+1} at the end (in any other case, we would be able to substitute some letters when we change some s'_i into s_i which would make the edit distance smaller). This is a contradiction.

For the implication **(iii)** $\Rightarrow$ **(i)** consider $w \in \mathcal{L}(G)$ with its compact G-decomposition $w = (s_i u_i)_{i=1}^{k} s_{k+1}$. We will argue that for all ℓ, the word $w(\ell) = (s_i u_i^{\ell})_{i=1}^{k} s_{k+1} \in \mathcal{L}(G)$ requires at least ℓ edits. Consider $w(\ell)$ for some ℓ. By the property of reachability sets, any run on $s_1 u_1^{\ell}$ has entered $R(u_1, Q)$ or made at least ℓ edits. That is because, in the best case, one can enter some arbitrary state upon reading s_1, but still, the sub-run on u_1^{ℓ} must have made ℓ edits or entered $R(u_1, Q)$, which is closed under reachability by definition. Similarly, for any j, any run on $(s_i u_i^{\ell})_{i=1}^{j}$ has either entered $R(u_j, R(u_{j-1}, R(u_{j-2}, \dots R(u_1, Q)\dots)))$ or there has been at least ℓ edits, using an argument like in the case for $j = 1$. Since $R(u_k, R(u_{k-1}, R(u_{k-2}, \dots R(u_1, Q)\dots))) = \emptyset$, no run can enter that set and thus there has been at least ℓ edits on $w(\ell)$. The implication and thus the lemma follows. □

The equivalence of (i) and (ii) gives a bound on the maximum finite edit distance for safety automata. The following lemma follows from Lemmas 12 and 15, together with Lemma 16 for (2).

Lemma 16. *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. As mentioned in the introduction, it is easy to define a family of context free grammars only accepting an exponential length word, using repeated doubling and thus the edit distance can be exponential between DPDAs and DFAs. We can also show that the inclusion problem reduces to the finite edit distance problem $\mathsf{FED}(\mathsf{DPDA}, \mathsf{NFA})$ and get the following lemma.

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., $\{(q, \#, s) : q \in F, s \in S\}$ for NFA (resp., $\{(q, \#, \bot, s) : q \in F, s \in S\}$ for DPDA). For DPDA the additional transitions are possible only with empty stack. □

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 18. 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 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 Lemma 17. In conclusion, we have the following proposition.

Proposition 19. *(1) For every class* $\mathcal{C} \in \{$DFA, NFA, DPDA, PDA$\}$*, the problems* TED$(\mathcal{C}$, PDA$)$ *and* FED$(\mathcal{C}$, PDA$)$ *are undecidable. (2) For every class* $\mathcal{C} \in \{$DPDA, PDA$\}$*, the problem* FED$(\mathcal{C}$, 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 Lemma 8 we show a reduction of the universality problem for nearly-deterministic PDA to TED(DFA, DPDA).

Proposition 22. *For every class* $\mathcal{C} \in \{$DFA, NFA, DPDA, PDA$\}$*, the problem* TED$(\mathcal{C}$, 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 Lemma 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 TED$(\mathcal{C}_1, \mathcal{C}_2)$, for $\mathcal{C}_1 \in \{$DFA, NFA, DPDA, PDA$\}$ and $\mathcal{C}_2 \in \{$DPDA, PDA$\}$. To complete the characterization of the problems FED$(\mathcal{C}_1, \mathcal{C}_2)$, with respect to their decidability, we still need to settle the decidability (and complexity) status of FED(DFA, DPDA). We leave it as an open problem, but conjecture that it is coNP-complete. This is because, if true and if Conjecture 18 is true as well, then the complexity of the FED problem matches the one for the language inclusion problem, except that PTime is changed to coNP-complete. See Table 1 and 2 of the introduction.

Conjecture 23. FED(DFA, DPDA) *is* coNP*-complete.*

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] A. Aho and T. Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM J. of Computing*, 1:305–312, 1972.

[2] Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Regular repair of specifications. In *LICS'11*, pages 335–344, 2011.

[3] Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Bounded repairability of word languages. *J. Comput. Syst. Sci.*, 79(8):1302–1321, 2013.

[4] Michael Benedikt, Gabriele Puppis, and Cristian Riveros. The per-character cost of repairing word languages. *Theor. Comput. Sci.*, 539:38–67, 2014.

[5] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, January 1981.

[6] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4), 2010.

[7] Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Nested weighted automata. *CoRR*, abs/1504.06117, 2015.

[8] Krishnendu Chatterjee, Rasmus Ibsen-Jensen, and Rupak Majumdar. Edit distance for timed automata. In *HSCC'14*, pages 303–312, 2014.

[9] Krishnendu Chatterjee and Yaron Velner. Mean-payoff pushdown games. In *LICS*, pages 195–204, 2012.

[10] Paweł Gawrychowski. Faster algorithm for computing the edit distance between slp-compressed strings. In *SPIRE'12*, pages 229–236, 2012.

[11] Thomas A. Henzinger and Jan Otop. From model checking to model measuring. In *CONCUR'13*, pages 273–287, 2013.

[12] John E. Hopcroft and Jefferey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Adison-Wesley Publishing Company, Reading, Massachusets, USA, 1979.

[13] R.M. Karp. Mapping the genome: some combinatorial problems arising in molecular biology. In *STOC 93*, pages 278–285. ACM, 1993.

[14] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[15] Yury Lifshits. Processing compressed texts: A tractability border. In *Combinatorial Pattern Matching*, pages 228–240. Springer, 2007.

[16] M. Mohri. Edit-distance of weighted automata: general definitions and algorithms. *Intl. J. of Foundations of Comp. Sci.*, 14:957–982, 2003.

[17] T. Okuda, E. Tanaka, and T. Kasai. A method for the correction of garbled words based on the levenshtein metric. *IEEE Trans. Comput.*, 25:172–178, 1976.

[18] G. Pighizzini. How hard is computing the edit distance? *Information and Computation*, 165:1–13, 2001.

[19] Barna Saha. The dyck language edit distance problem in near-linear time. In *FOCS'14*, pages 611–620, 2014.

[20] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, pages 168–173, 1974.