Constrained Approximate Subtree Matching by Finite Automata

Eliška Šestáková, Bořivoj Melichar, and Jan Janoušek

Faculty of Information Technology,
Czech Technical University in Prague,
Thákurova 9, 160 00 Praha 6, Czech Republic
Eliska.Sestakova@fit.cvut.cz
Borivoj.Melichar@fit.cvut.cz
Jan.Janousek@fit.cvut.cz

Abstract. Processing tree data structures usually requires a pushdown automaton as a model of computation. Therefore, it is interesting that a finite automaton can be used to solve the constrained approximate subtree pattern matching problem. A systematic approach to the construction of such matcher by finite automaton, which reads input trees in prefix bar notation, is presented. Given a tree pattern and an input tree with m and n nodes, respectively, the nondeterministic finite automaton for the pattern is constructed and it is able to find all occurrences of the pattern to subtrees of the input tree with maximum given distance k. The distance between the pattern and subtrees of an input tree is measured by minimal number of restricted tree edit operations, called leaf nodes edit operations. The corresponding deterministic finite automaton finds all occurrences in time $\mathcal{O}(n)$ and has $\mathcal{O}(|A|^k m^{k+1})$ states, where A is an alphabet containing all possible node labels. Note that the size is not exponential in the number of nodes of the tree pattern but only in the number of errors. In practise, the number of errors is expected to be a small constant that is much smaller than the size of the pattern. To achieve better space complexity, it is also shown how dynamic programming approach can be used to simulate the nondeterministic automaton. The space complexity of this approach is $\mathcal{O}(m)$, while the time complexity is $\mathcal{O}(mn)$.

Keywords: finite automaton, approximate tree pattern matching, subtree matching, constrained tree edit distance, dynamic programming

1 Introduction

Exact tree pattern matching, the process of finding all matches of a tree pattern in an input tree, is an analogous problem to the string pattern matching. One of the approaches used for string pattern matching is to construct a finite automaton for the pattern [3,13]. The automata approach for solving exact tree pattern matching problem has also been studied in [5,9] using pushdown automaton as a model of computation. For other methods used to solve exact tree pattern matching problem see [6,7] and [2].

Approximate tree pattern matching problem is an extension of both exact tree pattern matching and approximate string pattern matching. The goal of the approximate string pattern matching problem is to find a substring in an input text with the minimal distance (string-to-string correction problem) to a given pattern. Similarly, as for the exact string matching problem, the automata approach can be used again to solve the approximate string pattern matching problem as well, see [12,13].

The goal of the approximate tree pattern matching problem is to find all occurrences of a given tree pattern in an input tree with the minimal distance. Similarly, approximate subtree pattern matching is the process of finding all occurrences of a tree pattern to subtrees of an input tree with the minimal distance. To measure the distance between two trees (tree-to-tree correction problem [17]), a tree edit distance is used. Common operations used are node rename, node insertion and node deletion, proposed in [10]. The recent algorithm proposed in [4] computes the tree edit distance between two rooted ordered trees with m and n nodes in $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space, where $m \leq n$.

Several authors also proposed restricted forms of tree-to-tree correction problem where only a limited set of edit operations is allowed. For example, Selkow in [14] introduced a constrained tree edit distance, sometimes referred to as 1-degree edit distance, where delete and insert operations are restricted to leaf nodes of a tree. Selkow's approach is recursive, so the distance between two trees can always be computed. For more details on the tree edit distance survey see [1,8].

Both tree-to-tree correction and approximate tree pattern matching problems have applications is several areas such as genetics, XML processing and databases, compiler optimization or natural language processing. Therefore, many solutions exist. For example, see [14,16,17,18] and [1,6,11]. However, most of them lack clear references to a systematic approach of the standard theory of formal languages and automata.

This paper shows that finite automata can be used to solve approximate subtree pattern matching problem with a constrained set of tree edit operations allowed. This is quite interesting since processing tree data structures usually requires a pushdown automaton as a model of computation. However, using only a special restricted set of tree edit operations allows a finite automaton to be the sufficient model of computation. The proposed method defines leaf nodes edit operations involving only simple tasks, such as node rename, leaf insertion and leaf deletion. However, it is not allowed to use these operations repeatedly such as Selkow [14]. In other words, it is not allowed to use operations leaf node insertion and leaf node deletion recursively to insert or delete a subtree of an arbitrary size. Therefore, the distance between two trees can be unknown, if the trees cannot be transformed to each other using only leaf nodes edit operations.

First of all, the proposed method forms linear notations for a given tree pattern and an input tree by traversing their tree structures in a sequential way. After that, a finite automaton for constrained approximate subtree matching, which is directly analogous to approximate string matching automata, is constructed. The proposed automaton is built over a tree pattern P based on a maximum distance (number of errors) k desired and is able to find all occurrences of subtree P within a given input tree T in time linear to the number of nodes of T.

The major issue in automata theory is often the size of the deterministic automaton, which can be exponential in the number of nodes of the tree pattern. However, the size of the automaton in this case is $\mathcal{O}(|A|^k m^{k+1})$, where m is the number of nodes of the tree pattern P, k is the maximum number of errors and A is an alphabet containing all possible node labels. In practise the number of errors is expected to be a small constant, that is much smaller than the size of the pattern. The paper also presents how dynamic programming can be used to simulate the nondeterministic finite automaton. This approach comes with the $\mathcal{O}(m)$ space complexity and $\mathcal{O}(mn)$ time complexity.

The rest of this paper is organised as follows. Basic definitions are given in Section 2. The problem definition is presented in Section 3. Section 4 introduces the nondeterministic finite automaton for the constrained approximate subtree pattern matching. Following Section 5 deals with the dynamic programming approach used to simulate the nondeterministic automaton. Section 6 discusses the corresponding deterministic finite automaton and finally, Section 7 is the conclusion and future work discussion.

2 Basic Notions

An *alphabet* A is a finite non-empty set whose elements are called symbols. A *non-deterministic finite automaton* (NFA) is a 5-tuple $M = (Q, A, \delta, q_0, F)$, where Q is a finite set of states, A is an alphabet, δ is a state transition function from $Q \times A$ to the power set of Q, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of final states. A finite automaton is *deterministic* (DFA) if $\forall a \in A, q \in Q : |\delta(q, a)| \leq 1$.

A *rooted and directed tree* T is an acyclic connected directed graph $T = (N, E)$, where N is a set of nodes and E is a set of ordered pairs of nodes called directed edges. A *root* is a special node $r \in N$ with in-degree 0. All other nodes of a tree T have in-degree 1. There is just one path from the root r to every node $n \in N$, where $n \neq r$. A node n_1 is a *direct descendant* of a node n_2 if a pair $(n_2, n_1) \in E$.

T is called a *labelled* tree if there is a symbol from a finite alphabet A assigned to each node. T is called an *ordered* tree if a left-to-right sibling ordering in T is given. A *subtree* of a tree $T = (N, E)$ rooted at node $n \in N$ is a tree $T_n = (N_n, E_n)$, such that n is the root of T_n and N_n, E_n is the greatest possible subset of N, E, respectively.

The *size* of a tree $T = (N, E)$ denoted as $|T|$ is the cardinality of N. Any node of a tree with out-degree 0 is called a *leaf*. Leaves(T) stands for a set of all leaf nodes of the tree T. In the following, P and T will be used to denote rooted, ordered and labelled trees called a *tree pattern* and an *input tree*, respectively.

3 Problem Statement

A special type of the approximate tree pattern matching problem called approximate subtree matching is considered, where the goal is to find all occurrences of a tree pattern P to subtrees of an input tree T with maximum of k errors. Formally, the approximate subtree pattern matching problem for a maximum given distance is defined in the following way:

Definition 1 (Approximate subtree pattern matching with maximum of k errors). *A tree pattern P matches an input tree $T = (N, E)$ in a node $n \in N$ if the distance D between the pattern P and the subtree of T rooted at n is less than or equal to k, i.e., $D(P, T_n) \leq k$.*

The distance between a tree pattern and subtrees of an input tree is measured by minimal number of simple operations, called leaf nodes edit operations, applied to the tree pattern. Formal definitions of the leaf nodes edit distance and leaf nodes edit operations follows.

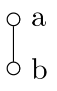
Definition 2 (Leaf nodes edit distance). *Let T_1 and T_2 be two rooted, ordered and labelled trees. The leaf nodes edit distance between T_1 and T_2, noted as $D_L(T_1, T_2)$, is the minimal number of leaf nodes edit operations needed to transform T_1 to T_2. The distance D_L is unknown if T_1 cannot be transformed to T_2 by using only leaf nodes edit operations.*

Definition 3 (Leaf nodes edit operations). *Let $T = (N, E)$ be a rooted, ordered and labelled tree and S_1, S_2, S_3 be sets such that $S_1 = N$, $S_2 = \text{Leaves}(T)$, $S_3 = N$. Then leaf nodes edit operations for T are defined as follows:*

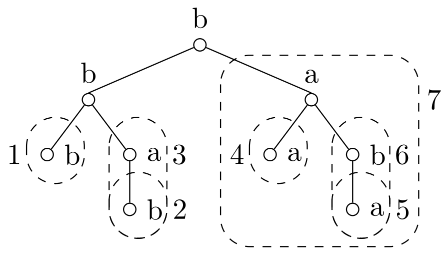
1. *node rename: change the label of a node $n \in S_1$ and assign $S_1 = S_1 \setminus \{n\}$,*
2. *leaf node deletion: delete a non-root leaf node $n \in S_2$ and assign $S_2 = S_2 \setminus \{n\}$,*
3. *leaf node insertion: insert a leaf node n_1 as a child of a node $n_2 \in S_3$. Do not update any set.*

In other words, it is not allowed to use operations node rename, leaf node insertion and leaf node deletion recursively to insert or delete a subtree of an arbitrary size.

Example 4. Let P and T be a tree pattern and an input tree as depicted in Figure 1a and Figure 1b, respectively. Having only leaf nodes edit operations allowed the tree pattern P can be modified as follows: (1) rename the root node a or the leaf node b, (2) delete the leaf node b, (3) add leaf nodes as children of the node b or as left or right siblings of the node b.



(a) Tree pattern P



(b) Input tree T

Figure 1: Graphical representation of the approximate subtree matching problem using leaf nodes edit distance $k = 2$

Thus, there are seven occurrences of the tree pattern P in the input tree T with maximum of $k = 2$ errors as shown in Figure 1b. These occurrences are marked with dashed lines and have numbers 1–7 assigned. Starting from the left, there are following errors in the occurrences of the tree pattern P to subtrees of the input tree T:

(1. match) 2 errors: rename the node a to b, delete the node b,
(2. match) 2 errors: rename the node a to b, delete the node b,
(3. match) exact match, 0 errors,
(4. match) 1 error: delete the node b,
(5. match) 1 error: delete the node b,
(6. match) 2 errors: rename the node a to b, rename the node b to a,
(7. match) 2 errors: add a leaf a as a child and as a left sibling of the node b.

Every sequential algorithm traverses a processed tree structure in a sequential order of nodes, which forms a corresponding linear notation of the tree structure. The proposed method uses the following linear notation of trees called the prefix bar notation which was introduced by Stoklasa, Janoušek and Melichar in [15].

Definition 5 (Prefix bar notation). *The prefix bar notation* $\text{pref\_bar}(T)$ *of a tree* T *is defined as follows:*

1. $\text{pref\_bar}(a) = a \mid$ *if* a *is both the root and a leaf,*
2. $\text{pref\_bar}(T) = a\ \text{pref\_bar}(b_1)\ \text{pref\_bar}(b_2) \cdots \text{pref\_bar}(b_n) \mid$ *if* a *is the root of the tree* T *and* $b_1, b_2, \ldots, b_n$ *are direct descendants of* a.

Example 6. Let P and T be a tree pattern and an input tree as depicted in Figure 1a and Figure 1b, respectively. The prefix bar notations of P and T are described as follows: $\text{pref\_bar}(P) = a\,b\,|\,|$ and $\text{pref\_bar}(T) = b\,b\,b\,|\,a\,b\,|\,|\,|\,a\,a\,|\,b\,a\,|\,|\,|\,|$.

4 Nondeterministic Finite Automaton for Constrained Approximate Subtree Matching

This section deals with the constrained approximate subtree pattern matching by a nondeterministic finite automaton, which reads an input tree T in the prefix bar notation. The finite automaton is able to find all approximate occurrences of a tree pattern P with maximum of k errors to subtrees of an input tree T using leaf nodes edit distance.

The method is analogous to the construction of approximate string pattern matching automata. The NFA is built for the tree pattern P to recognize a language $A^*X(P,k)$, where A is an alphabet containing all possible node labels that may occur in both the tree pattern and the input tree. The alphabet also includes a special symbol called bar (noted as $|$), introduced in the definition of the prefix bar notation. $X(P,k)$ is a finite language generated for the number of allowed errors $k \geq 1$ from a given tree pattern P using leaf nodes edit operations, formally defined as $X(P,k) = \{\text{pref\_bar}(S) : \text{pref\_bar}(S) \in A^*, D_L(P,S) \leq k\}$. Thus, the proposed automaton can find all occurrences of $x \in X(P,k)$ in a given prefix bar notation of an input tree. The construction of the NFA is described by Algorithm 1 in detail.

Example 7. Let P be a tree pattern as shown in Figure 1a with its prefix bar notation $\text{pref\_bar}(P) = a\,b\,|\,|$. The transition diagram of the NFA constructed for the tree pattern P and maximum number of errors $k = 2$ by Algorithm 1 is shown in Figure 2.

The NFA has a regular structure. State q_{ij} is at depth i (a position in the pattern) and on level j (number of errors). States $q_{i'j'}$ are assistant nodes used for insert leaf operations allowing inserting bars. Insert leaf operations are represented by two subsequent "vertical" transitions. The first transitions are labelled by all symbols of the alphabet A (except $|$) and they are followed by second transitions labelled by $|$. Rename node operations are represented by "diagonal" transitions labelled by those symbols of the alphabet A (except $|$) for which no direct transition to the next state exists. "Diagonal" ε-transitions represent delete leaf operations.

```
Input: A tree pattern $P$ ($|P| = m$) and its prefix bar notation pref_bar($P$) $= p_1 p_2 \cdots p_{2m}$,
        maximum number $k$ of errors allowed.
Output: NFA $M$ accepting language $A^* X(P,k)$.
 1. Let $M' = (\{q_0, q_1, \ldots, q_{2m}\}, A, \delta, q_0, \{q_{2m}\})$ be an exact pattern matching automaton, where
    (1) $\forall i,\ 0 \le i < 2m : q_{i+1} \in \delta(q_i, p_{i+1})$,
    (2) $\forall a \in A : q_0 \in \delta(q_0, a)$.
 2. Create a sequence of $k+1$ instances of $M'$ such that: $M'_j = (Q_j, A, \delta_j, q_{0j}, F_j)$,
    $j = 0, 1, 2, \ldots, k$. $Q_j = \{q_{0j}, q_{1j}, \ldots, q_{2mj}\}, F_j = \{q_{2mj}\}$.
 3. Construct the automaton $M = (Q, A, \delta, q_0, F)$ as follows:
    (a) $Q = \bigcup_{j=0}^{k} Q_j,\ q_0 = q_{00},\ F = \bigcup_{j=0}^{k} F_j$
    (b) (copy) $\forall q \in Q, a \in A, j = 0, 1, 2, \ldots, k$ do $\delta(q, a) = \delta_j(q, a)$,
        (rename) $\forall i = 0, 1, \ldots, 2m-1, j = 0, 1, \ldots, k-1, a \in A \setminus \{p_{i+1}, |\}$ do
            if $(p_{i+1} \ne |)$ then $q_{i+1,j+1} \in \delta(q_{ij}, a)$,
        (delete) $\forall i = 0, 1, \ldots, 2m-2, j = 0, 1, \ldots, k-1$ do
            if $((p_{i+1} \ne |) \wedge (p_{i+2} = |))$ then $q_{i+2,j+1} \in \delta(q_{ij}, \varepsilon)$
        (insert) $\forall i = 1, 2, \ldots, 2m-1, j = 0, 1, \ldots, k-1, a \in A \setminus \{|\}$ do
            $Q = Q \cup \{q_{i'j'}\}, q_{i'j'} \in \delta(q_{ij}, a), q_{i,j+1} \in \delta(q_{i'j'}, |)$.
 4. Remove all states inaccessible from state $q_0$ in $M$.
```

Algorithm 1: Construction of NFA that finds all approximate occurrences of a tree pattern P to subtrees of an input tree using leaf nodes edit distance.

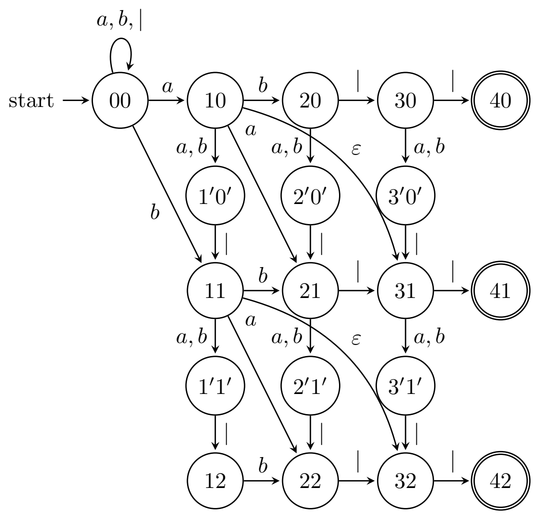


Figure 2: Transition diagram of NFA from Example 7 accepting all approximate occurrences of a tree pattern P in prefix bar notation pref_bar(P) $= a\,b\,|\,|$ in an input tree using leaf nodes edit operations with maximum of $k = 2$ errors

5 Simulation of the Nondeterministic Finite Automaton for Constrained Approximate Subtree Matching

This section describes how the dynamic programming approach can be used to simulate the nondeterministic finite automaton for constrained approximate subreee matching. Given a tree pattern P with m nodes and an input tree T with n nodes, the algorithm computes a matrix D of size $(2m+1) \times (2n+1)$. Each element of the matrix $d_{i,j}$ $(0 \leq i \leq 2m, 0 \leq j \leq 2n)$ contains the constrained distance between the partial trees represented by prefix bar notations of the tree pattern of length i and the input tree of length j. Elements of the matrix D are computed as follows:

$$
\begin{aligned}
&d_{i,0} = k+1 \qquad 0 < i \leq 2m \\
&d_{0,j} = 0 \qquad\quad 0 \leq j \leq 2n \\
&d_{i,j} = min \begin{cases}
\text{if } (p_i = t_j) \text{ then } d_{i-1,j-1} & \text{(match)} \\
\text{if } (p_i \neq t_j \wedge p_i, t_j \neq |) \text{ then } d_{i-1,j-1} + 1, & \text{(rename)} \\
\text{if } (i > 1 \wedge p_i = | \wedge p_{i-1} \neq |) \text{ then } d_{i-2,j} + 1, & \text{(delete)} \\
\text{if } (j > 1 \wedge t_j = | \wedge t_{j-1} \neq |) \text{ then } d_{i,j-2} + 1, & \text{(insert)} \\
k+1 & \text{(otherwise)}
\end{cases} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad 0 < i \leq 2m,\ 0 < j \leq 2n
\end{aligned}
$$

This formula represents the simulation of the nondeterministic finite automaton introduced in the previous section. If $d_{2m,j} \leq k$ then the tree pattern occurs in the input tree T with $d_{2m,j}$ errors ending at position j in the pref_bar(T). Note, that all values $d_{i,j} > k$ in the matrix D can be replaced by the value $k+1$ representing number of errors higher than k.

Example 8. Let P be a tree pattern (pref_bar$(P) = a\,b\,|\,|$) and T be an input tree (pref_bar$(T) = b\,b\,b\,|\,a\,b\,|\,|\,|\,a\,a\,|\,b\,a\,|\,|\,|\,|$) as shown in Figure 1a and Figure 1b, respectively. The matrix D computed for the tree pattern P and the input tree T with maximum number of errors $k = 2$ is shown in Table 1. The occurrences of the tree pattern P in the input tree T are marked with bold. All errors higher than 2 are in the matrix represented by number 3.

D	-	b	b	b	$\vert$	a	b	$\vert$	$\vert$	$\vert$	a	a	$\vert$	b	a	$\vert$	$\vert$	$\vert$	$\vert$
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	3	1	1	1	2	0	1	1	3	3	0	0	1	1	0	2	3	3	3
b	3	3	1	1	2	3	0	3	3	3	3	1	3	1	2	2	3	3	3
$\vert$	3	2	2	2	1	1	2	0	3	3	1	1	1	2	1	2	2	3	3
$\vert$	3	3	3	3	**2**	3	3	**2**	**0**	3	3	3	**1**	3	3	**1**	**2**	**2**	3

Table 1: Matrix D for a tree pattern P, input tree T and $k = 2$, where pref_bar$(P) = a\,b\,|\,|$ and pref_bar$(T) = b\,b\,b\,|\,a\,b\,|\,|\,|\,a\,a\,|\,b\,a\,|\,|\,|\,|$

Theorem 9. *The dynamic programming approach described by the formula can be used to simulate a run of the NFA for constrained approximate subtree matching using leaf nodes edit operations.*

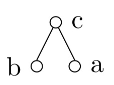
Proof. The NFA has a regular structure. It consists of depths i $(0 \le i \le 2m)$ and levels j $(0 \le j \le 2n)$. A depth represents a position in the pattern, while a level stands for the number of existing errors. Therefore, state q_{ij} is at depth i and on level j. In the matrix D, each row i represents the depth i and each column j corresponds to a j-th step of a run of the NFA (i.e., j symbols of pref_bar(T) read).

Every value $d_{i,j}$ of the matrix D stands for a level number (number of errors) of the topmost active state in i-th depth of the NFA and j-th step of the run of the NFA. If the value $d_{i,j} > k$ it simply means that there is no active state for the particular depth and step of the NFA.

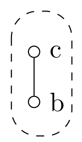
At the beginning only the initial state is active, which is in the formula represented by setting $d_{0,0} = 0$ and $d_{i,0} = k + 1$ $(0 < i \le 2m)$. The second part of the formula $d_{0,j} = 0$ $(0 \le j \le 2n)$ simulates the self loop in the initial state. Third part of the formula describes the individual operations. The term $d_{i-1,j-1}$ simulates a matching transition – the value is copied to $d_{i,j}$ as the level has not changed and depth and step of the NFA is increased by 1. Term $d_{i-1,j-1} + 1$ corresponds to a rename transition – the level, depth and step are all increased by 1. Delete transition is represented by term $d_{i-2,j} + 1$, level is increased by 1, depth (position in the pattern) is increased by 2, but position in the text is not changed. The term $d_{i,j-2} + 1$ simulates insert transitions – level is again increased by 1, position in the text is increased by 2, but the depth is not increased. The last term sets $d_{i,j}$ to $k + 1$, which means there is no possible transition.

Therefore, all transitions of the NFA are considered. If $d_{2m,j} \le k$ then a final state $q_{2m,d_{2m,j}}$ is active and the match is reported. The tree pattern occurs in the input tree T with $d_{2m,j}$ errors ending at position j in the pref_bar(T).
Possibly, there can exist more ways in the nondeterministic automaton leading to an accepting state. To ensure that values $d_{2m,j}$ are minimal possible, the third part of the formula contains minimum function *min*. Example 10 supports this statement. □

Example 10. Let P be a tree pattern and T be an input tree as shown in Figure 3a and Figure 3b, respectively, where pref_bar$(P) = c\,b\,|\,a\,|\,|$, pref_bar$(T) = c\,b\,|\,|$. Obviously, the pattern P occurs in the tree T just once with 1 error (delete leaf node a). However, the pattern P also occurs in T with 2 errors (delete leaf node b, rename a to b). Without minimum function in the dynamic programming formula, the second match would be reported instead the first one.



(a) Tree pattern P



(b) Input tree T

Figure 3: Graphical representation of Example 10

Theorem 11. *The simulation of the NFA for constrained approximate subtree matching using leaf nodes edit operations by dynamic programming has time complexity* $(2m + 1) \times (2n + 1) = \mathcal{O}(mn)$ *and space complexity* $4m = \mathcal{O}(m)$.

Proof. The dynamic programming approach builds a matrix D of size $(2m + 1) \times (2n + 1)$. All operations introduced in the formula can be done in constant time, hence the time complexity of the simulation is $\mathcal{O}(mn)$. In the matrix D every column is computed using just two previous columns. Therefore, only $4m$ space is needed in order to compute all values and the space complexity results in $\mathcal{O}(m)$.

6 Deterministic Finite Automaton for Constrained Approximate Subtree Matching

To achieve better time complexity, the NFA can be turned into the DFA by using the standard determinisation algorithm (see [13], Algorithm 1.40). To compute the positions of all occurrences of the tree pattern P in the input tree T, the DFA is simply run on the prefix bar notation of the input tree T. The DFA reports a match every time it goes through a final state. All occurrences are located in time linear to the number of nodes of T.

Theorem 12. *Given an input tree T ($|T| = n$) and a tree pattern P ($|P| = m$), the deterministic automaton for approximate subtree matching that is obtained using standard determinisation algorithm over the NFA constructed by Algorithm 1 finds all approximate occurrences of the subtree P in the input tree T using leaf nodes edit distance in time $2n = \mathcal{O}(n)$.*

Proof. The prefix bar notation of the input tree T is in the searching phase read exactly once, symbol by symbol from left to right. The appropriate transition is taken each time a symbol is read, resulting in exactly $2n$ transitions. Approximate occurrences of the tree pattern P are reported each time that DFA goes through a final state.

In order to prove the upper bound of the state complexity of the deterministic finite automaton that finds all approximate occurrences of subtree P in an input tree T using leaf nodes edit distance, some of the results of the previous research concerning the dictionary matching problem will be used. The goal of the dictionary matching problem is to preprocess the dictionary, a finite set of words X, in order to locate words of X that occur in any given input word.

Crochemore at al. in [3] propose an algorithm for a direct construction of the deterministic dictionary matching automaton that recognizes a language A^*X and thus it can find all occurrences of words $x \in X$ in a given text. They have proven the automaton has $\mathcal{O}(\sum_{x \in X} |x|)$ states.

Later in [13], Melichar showed the equivalence of Crochemore's automaton to a finite automaton, that is created from a nondeterministic one, using standard determinisation algorithm based on a subset construction. The nondeterministic automaton has a tree-like structure with the self loop in the initial state for all symbols of the alphabet.

Moreover, it was shown that any acyclic automaton accepting language X can be transformed into a deterministic dictionary matching automaton accepting language A^*X by just adding the self loop in the initial state and using standard determinisation algorithm. It has been proven that the number of states of such created automaton in not greater than $\mathcal{O}(\sum_{x \in X} |x|)$.

The proposed nondeterministic automaton that finds all approximate occurrences of a tree pattern P to subtrees of an input tree T using leaf nodes edit distance can be viewed as a nondeterministic dictionary matching automaton for a dictionary $X(P, k)$. The finite language $X(P, k)$ was defined in the previous section as follows:

$$X(P,k) = \{\text{pref\_bar}(S) \,:\, \text{pref\_bar}(S) \in A^*, D_L(P,S) \le k\}.$$

Therefore, the deterministic automaton M accepting language $A^*X(P,k)$, noted as $M(A^*X(P,k))$, has maximum of $\mathcal{O}(\sum_{x \in X(P,k)} |x|)$ states. Hence, the goal is to determine the size of the language $X(P,k)$ by finding the number of strings that represent prefix bar notations of trees created from the tree pattern P by using leaf nodes edit operations.

Theorem 13. *Given a tree pattern P ($|P| = m$), the number of strings, standing for prefix bar notations of trees, created from P by at most k rename node operations is $\mathcal{O}(|A|^k m^k)$.*

Proof. The set of strings created by exactly i rename node operations $(0 \le i \le k)$ is made by replacing exactly i symbols of pref_bar(P) by other symbols. There are $\binom{m}{i}$ possibilities for choosing i symbols from pref_bar(P) and $|A| - 2$ possibilities for choosing the new symbol. Hence, the number of generated strings is at most $\binom{m}{i}(|A|-2)^i = \mathcal{O}(m^i)(|A|-2)^i = \mathcal{O}(|A|^i m^i)$. Therefore, the size of the set of strings created by at most k rename node operations is $\sum_{i=0}^{k} \mathcal{O}(|A|^i m^i) = \mathcal{O}(|A|^k m^k)$.

Theorem 14. *Given a tree pattern P ($|P| = m$), the number of strings, standing for prefix bar notations of trees, created from P by at most k delete leaf node operations is $\mathcal{O}(|\text{Leaves}(P)|^k)$.*

Proof. The set of strings created by exactly i delete leaf operations $(0 \le i \le k)$ is made by deleting exactly $2i$ symbols from pref_bar(P). There are $\binom{|\text{Leaves}(P)|}{i} = \mathcal{O}(|\text{Leaves}(P)|^i)$ possibilities for choosing i leaf nodes from pref_bar(P). Therefore, the size of the set of strings created by at most k delete leaf operations can be specified as follows: $\sum_{i=0}^{k} \mathcal{O}(|\text{Leaves}(P)|^i) = \mathcal{O}(|\text{Leaves}(P)|^k)$.

Theorem 15. *Given a tree pattern P ($|P| = m$), the number of strings, standing for prefix bar notations of trees, created from P by at most k insert leaf node operations is $\mathcal{O}(|A|^k m^k)$.*

Proof. The number of strings created by exactly i insert leaf node operations can be transformed to the number of dipaths (directed paths) that can be found in a nondeterministic finite automaton constructed by Algorithm 1 with rename node and delete leaf nodes transitions removed. A dipath starts at the initial state and goes to some of final states. There are $2m$ steps needed to be done to the right (reading the pattern) and i steps down (insert operations, the two transitions representing an insert operation can be viewed as one step). Hence, every dipath is represented by a string containing $2m$ letters R (right) and i letters D (down). Moreover, the dipath needs to start and end with the letter R. The number of such strings is $\binom{2m+i-2}{i}$. Each letter D can represent $|A|-1$ inserted symbols, so the total number of strings created by exactly i insert leaf node operations is at most $\binom{2m+i-2}{i}(|A|-1)^i = \mathcal{O}(m^i|A|^i)$. Therefore, the size of the set of strings created by at most k insert leaf node operations is $\sum_{i=0}^{k} \mathcal{O}(|A|^i m^i) = \mathcal{O}(|A|^k m^k)$.

Theorem 16. *Given a tree pattern P ($|P| = m$), the number of strings, standing for prefix bar notations of trees, created from P by at most k operations node rename, leaf node insertion and leaf node deletion is $\mathcal{O}(|A|^k m^k)$.*

Proof. The number of such strings can be computed as follows:

$$\sum_{x=0}^{k}\sum_{y=0}^{k-x}\sum_{z=0}^{k-x-y} \underbrace{\mathcal{O}(|A|^x m^x)}_{\substack{\text{rename}\\ \text{x nodes}}}\underbrace{\mathcal{O}(|A|^y m^y)}_{\substack{\text{insert}\\ \text{y leaves}}}\underbrace{\mathcal{O}(|\text{Leaves}(P)|^z)}_{\substack{\text{delete}\\ \text{z leaves}}}$$

$$\sum_{x=0}^{k}\sum_{y=0}^{k-x}\sum_{z=0}^{k-x-y} \mathcal{O}(|A|^{x+y} m^{x+y} |\text{Leaves}(P)|^z)$$

$$\sum_{x=0}^{k}\sum_{y=0}^{k-x}\sum_{z=0}^{k-x-y} \mathcal{O}(|A|^{x+y} m^{x+y+z}) = \mathcal{O}(|A^k| m^k).$$

Theorem 17. *Let P be a tree pattern such that $|P| = m$ and k be the maximum number of errors allowed. The number of states of the deterministic finite automaton $M(A^*X(P,k))$ that is obtained using standard determinisation algorithm over NFA constructed by Algorithm 1 is $\mathcal{O}(|A|^k m^{k+1})$.*

Proof. As stated in [13], [3] the state complexity of this automaton is at most the same as the size of the language $X(P,k)$. Since $\forall x \in X(P,k) : |x| \leq 2m + 2k$, the size of the language $X(P,k)$ is

$$\mathcal{O}(\sum_{x \in X(P,k)} |x|) = \mathcal{O}((2m+2k)|A|^k m^k) = \mathcal{O}(|A|^k m^{k+1}).$$

7 Conclusion and Future Work

It was shown that finite automata can be used to solve constrained approximate subtree pattern matching problem. This is quite interesting since processing tree data structures usually requires a pushdown automaton as a model of computation. The proposed method creates a nondeterministic finite automaton for a given tree pattern P which is able to find all occurrences of a tree pattern P to subtrees of an input tree T with maximum distance (number of errors) k. The distance between a tree pattern P and subtrees of an input tree T is measured by minimal number of simple operations called leaf nodes edit operations.

The NFA can be turned into DFA by using the standard determinisation algorithm. The searching phase is afterwards performed in time $\mathcal{O}(n)$, where n is the number of nodes of an input tree T. In theory, the state complexity of the deterministic automaton can be exponential in the number of nodes of the tree pattern P. However, Section 6 gives the proof that the size of the proposed deterministic automaton is only $\mathcal{O}(|A|^k m^{k+1})$, where m is the number of nodes of the tree pattern P, k is the maximum number of errors and A is an alphabet containing all possible node labels. In practise the number of errors is expected to be a constant much smaller than the size of the pattern.

In Section 5, it was also shown how dynamic programming can be used to simulate the nondeterministic finite automaton. This approach has $\mathcal{O}(m)$ space complexity and $\mathcal{O}(mn)$ time complexity.

The proposed methods solve an approximate tree pattern matching subproblem since the operations used are a subset of general tree edit operations. Hence, the techniques described here may also be relevant to other forms of approximate tree pattern matching problem, which we hope to explore in the future. Currently, we are working on the automata approach in respect to larger sets of edit operations allowed. In case of less restricted sets including general operations such as node insertion or node deletion the pushdown automata are required as models of computation.

Acknowledgements. This research has been partially supported by grant of CTU in Prague as project No. SGS17/209/OHK3/3T/18.

References

1. P. Bille: *A survey on tree edit distance and related problems.* Theoretical computer science, 337(1) 2005, pp. 217–239.
2. R. Cole and R. Hariharan: *Tree pattern matching to subset matching in linear time.* SIAM Journal on Computing, 32(4) 2003, pp. 1056–1066.
3. M. Crochemore and C. Hancart: *Automata for matching patterns*, in Handbook of formal languages, Springer, 1997, pp. 399–462.
4. E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann: *An optimal decomposition algorithm for tree edit distance.* ACM Trans. Algorithms, 6(1) Dec. 2009, pp. 2:1–2:19.
5. T. Flouri: *Pattern matching in tree structures*, PhD thesis, Czech Technical University, 2012.
6. C. M. Hoffmann and M. J. O'Donnell: *Pattern matching in trees.* Journal of the ACM (JACM), 29(1) 1982, pp. 68–95.
7. S. R. Kosaraju: *Efficient tree pattern matching*, in Foundations of Computer Science, 1989., 30th Annual Symposium on, IEEE, 1989, pp. 178–183.
8. L. Krčál: *Tree edit distance and approximate tree pattern matching problem.* Bachelor's thesis. Department of Computer Science and Engineering, Czech Technical University in Prague, Prague, Czech Republic, 2011.
9. J. Lahoda and J. Žďárek: *Simple tree pattern matching for trees in the prefix bar notation.* Discrete Applied Mathematics, 163 2014, pp. 343–351.
10. S. Y. Lu: *A tree-to-tree distance and its application to cluster analysis.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 1979, pp. 219–224.
11. F. Luccio and L. Pagli: *Simple solutions for approximate tree matching problems*, in Colloquium on Trees in Algebra and Programming, Springer, 1991, pp. 193–201.
12. B. Melichar: *Approximate string matching by finite automata*, in International Conference on Computer Analysis of Images and Patterns, Springer, 1995, pp. 342–349.
13. B. Melichar, J. Holub, and T. Polcar: *Text searching algorithms.* Available on: http://stringology.org/athens, 2005.
14. S. M. Selkow: *The tree-to-tree editing problem.* Information processing letters, 6(6) 1977, pp. 184–186.
15. J. Stoklasa, J. Janoušek, and B. Melichar: *Subtree pushdown automata for trees in bar notation, 2010.* London Stringology Days, 2010.
16. M. Svoboda and I. Holubová: *Refinement correction strategy for invalid XML documents and regular tree grammars*, in International Conference on Database and Expert Systems Applications, Springer, 2014, pp. 308–316.
17. K. C. Tai: *The tree-to-tree correction problem.* Journal of the ACM (JACM), 26(3) 1979, pp. 422–433.
18. K. Zhang, D. Shasha, and J. T. L. Wang: *Approximate tree matching in the presence of variable length don't cares.* Journal of Algorithms, 16(1) 1994, pp. 33–66.