Information and Computation 247 (2016) 278–289

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Approximate matching between a context-free grammar and a finite-state automaton ☆

Sang-Ki Ko [a], Yo-Sub Han [a,*], Kai Salomaa [b]

[a] Department of Computer Science, Yonsei University, Seoul 120-749, Republic of Korea
[b] School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada

ARTICLE INFO

Article history:
Received 6 October 2014
Received in revised form 29 July 2015
Available online 8 February 2016

Keywords:
Approximate matching
Edit-distance
Gap penalty
Context-free grammars
Finite-state automata

ABSTRACT

For a given context-free grammar (CFG) and a finite-state automaton (FA), we tackle the edit-distance problem—the problem of computing the most similar pair of strings in the two respective languages. In particular, we consider three different gap cost models for the edit-distance that are crucial for finding a proper alignment between two bio sequences: the linear, affine and concave models. We design efficient algorithms for the edit-distance between a CFG and an FA under these gap cost models. The time complexity of our algorithm for computing the linear or affine gap distance is polynomial and the time complexity for the concave gap distance is exponential.

1. Introduction

Given a pattern w and a text T, the string matching problem is to find occurrences of w in T and the approximate matching problem is to find occurrences of w' in T such that w and w' are similar by a predefined similarity metric. The most common similarity metric is the edit-distance [14]. Many researchers investigated the approximate pattern matching problem with various types of mismatches [1,6,15,16,18,23]. For example, Aho and Peterson [1], and Lyon [15] developed an $O(m^2n^3)$ algorithm for the approximate matching between a string of length n and a context-free language specified by a grammar of size m. They generalized Earley's algorithm [6] for parsing context-free languages and considered the edit-distance model [14] with a unit-cost function. Myers [18] considered several variants of the problem under various gap cost functions such as the linear, affine and concave gap costs; these gap cost models are important for identifying proper alignments between two bio sequences in practice [19,20]. Myers designed $O(mn^2(n+\log m))$ algorithms for the linear and the affine gap costs and sketched an $O(m^5n^8 8^m)$ algorithm for the concave gap cost.

The approximate matching problem is based on the edit-distance between two strings, or between a string and a language. This led researchers to examine the edit-distance between two formal languages. Mohri [17] proved that computing the edit-distance between two context-free languages is undecidable and provided a quadratic time algorithm for computing the edit-distance between two regular languages. Choffrut and Pighizzini [5] considered the relative edit-distance between languages and the reflexivity of binary relations. Konstantinidis [13] considered the edit-distance of a regular language—the minimum edit-distance among all pairs of distinct strings of the language. Recently, the authors [9] studied the Levenshtein distance [14] between a context-free language and a regular language given, respectively, by a pushdown automaton (PDA) P

☆ A preliminary version appeared in *Proceedings of the 18th International Conference on Implementation and Application of Automata*, CIAA 2013, LNCS **7982**, 146–157, Springer-Verlag, 2013.

* Corresponding author.
E-mail addresses: narame7@yonsei.ac.kr (S.-K. Ko), emmous@yonsei.ac.kr (Y.-S. Han), ksalomaa@cs.queensu.ca (K. Salomaa).

http://dx.doi.org/10.1016/j.ic.2016.02.001
0890-5401/

and a finite-state automaton (FA) A. Their algorithm constructs an *alignment PDA* that computes all possible alignments between $L(A)$ and $L(P)$, converts the alignment PDA into a CFG and finds the optimal alignment from the resulting grammar. Under the assumption that a PDA can push at most two symbols and pop at most one symbol, it takes $O(n^3)$ time to convert a PDA of size n into a CFG and the size of the resulting grammar is at most $O(n^3)$ [10]. If a context-free language is given by a CFG instead of a PDA, then we need to construct a PDA for an input CFG before computing the alignment PDA. This leads us to design algorithms that compute the edit-distance between a CFG and an FA without constructing a PDA, and to extend this problem to considering the approximate matching between a CFG and an FA.

We calculate the minimum edit-distance and the optimal alignment between the most similar pair of strings generated, respectively, by a CFG and an FA. We present algorithms for computing optimal alignments between a CFG and an FA under various gap distances. While several previous algorithms for the edit-distance between two formal languages are based on the Cartesian product construction [9,11,13,17], we design dynamic programming algorithms for the considered problem. Assume that the size of the input CFG is m and the size of the input FA is n. Our algorithms compute the linear and the affine gap distances in $O(m^2n^5)$ time, and the concave gap distance in $O(m^5n^8 8^m \log^3 n)$ time.

In Section 2, we briefly recall basic notation and terminology. We define the edit-distance model in Section 3. In Section 4, we introduce a dynamic programming algorithm for computing the edit-distance between a CFG and an FA. The next two sections then extend the algorithm to the problems of computing the affine and the concave gap distances.

2. Preliminaries

In the following, Σ is always a finite alphabet of characters and Σ^* denotes the set of all strings over Σ. The size $|\Sigma|$ of Σ is the number of characters in Σ. A language over Σ is any subset of Σ^*. Given a set X, 2^X denotes the power set of X.

The symbol $\emptyset$ denotes the empty language and the symbol λ denotes the null string. The length of a string $w \in \Sigma^*$ is $|w|$. Given a string $w \in \Sigma^*$, let $w[i]$ be the i'th character of w and $w[i, j]$ be the substring of w from $w[i]$ to $w[j]$—$w[i, j] = w[i]w[i+1]\cdots w[j]$—where $1 \le i < j \le |w|$. A (nondeterministic) finite-state automaton (FA) is specified by a tuple $M = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a set-valued transition function, $s \in Q$ is the start state and $F \subseteq Q$ is a set of final states. If F consists of a single state f, we use f instead of $\{f\}$ for short. We let $M_{q,p} = (Q, \Sigma, \delta, q, p)$ denote the new FA obtained from M by replacing the start state with q and the only final state with p.

The transition function δ is extended in the natural way to a function $Q \times \Sigma^* \to 2^Q$ that reflects state changes on a sequence of characters. A string x over Σ is accepted by M if there is a labeled path from s to a state in F such that this path spells out the string x. In other words, $\delta(s, x) \cap F \neq \emptyset$. The language $L(M)$ of M is the set of all accepted strings. Let $|Q|$ be the number of states in M and $|\delta|$ be the number of transitions in M. We define the size $|M|$ of M to be $|Q| + |\delta|$.

A context-free grammar (CFG) G is specified by a tuple $G = (V, \Sigma, R, S)$, where V is a set of variables, $R \subseteq V \times (V \cup \Sigma)^*$ is a finite set of productions and $S \in V$ is the start symbol. Let $\alpha A \beta$ be a string over $V \cup \Sigma$, where A is a variable in V and $A \to \gamma$ is a production of G. Then, we say that $\alpha\gamma\beta$ is obtained from $\alpha A\beta$ in a single derivation step and this is denoted by $\alpha A\beta \Rightarrow \alpha\gamma\beta$. The reflexive, transitive closure of $\Rightarrow$ is $\overset{*}{\Rightarrow}$. The context-free language defined by G is $L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$. We say that a variable $A \in V$ is *useless* if a CFG $G_A = (V, \Sigma, R, A)$—A is the start symbol in G—generates the empty language. Note that we assume that a CFG has no useless variables. By the size $|G|$ of the grammar G we mean the sum $\sum_{A\to\alpha\in R}(1 + |\alpha|)$ of the lengths of all productions of G.

A CFG is in Chomsky normal form (CNF) if all of its production rules are of the form: $A \to BC$ or $A \to a$, where $A, B, C \in V$ and $a \in \Sigma$. Every context-free grammar G generating a language not containing the null string can be converted into an equivalent CNF grammar of size $O(|G|^2)$.

We also consider *pseudo-CNF* grammars that in addition to the above rules allow rules of the form $A \to B$, where $A, B \in V$. It is known that an arbitrary grammar G can be transformed into an equivalent pseudo-CNF grammar whose size is still $O(|G|)$ [21].

For more details on automata theory, we refer the reader to the textbooks [10,24].

3. Edit-distance

The edit-distance between two strings x and y is the smallest number of atomic operations that transform x to y. Researchers considered different atomic edit operations in different applications in the literature [4,22]. Here we consider three basic operations—insertion, deletion and substitution—for simplicity. Given an alphabet Σ, let

$$\Omega = \{(a \to b) \mid a, b \in \Sigma \cup \{\lambda\}, a = b \text{ implies } a \neq \lambda\}$$

be a set of edit operations. Namely, Ω is an alphabet of all edit operations for *deletions* $(a \to \lambda)$, *insertions* $(\lambda \to a)$ and *substitutions* $(a \to b)$.

Let h be the morphism from Ω^* into $\Sigma^* \times \Sigma^*$ defined by setting

$$h((a_1 \to b_1)\cdots(a_n \to b_n)) = (a_1 \cdots a_n, b_1 \cdots b_n).$$

We say that $\omega \in \Omega^*$ is an *alignment* of strings $x, y \in \Sigma^*$ if $h(\omega) = (x, y)$. For example, a string $\omega = (a \to \lambda)(b \to b)(\lambda \to c)(c \to c)$ over Ω is an alignment between abc and bcc, and $h(\omega) = (abc, bcc)$. We associate a non-negative edit cost

$c(\sigma)$ to each edit operation $\sigma \in \Omega$, where c is a function $\Omega \to \mathbb{R}_+$. We can extend the function to give the cost of an alignment $\omega = \sigma_1 \cdots \sigma_n$, where $\sigma_i \in \Omega$, $1 \le i \le n$, in the natural way:

$$c(\omega) = \sum_{i=1}^{n} c(\sigma_i).$$

Definition 1. The edit-distance $d(x, y)$ of two strings x and y is the minimal cost of an alignment ω between x and y:

$$d(x, y) = \min\{c(\omega) \mid h(\omega) = (x, y)\}.$$

We say that ω is optimal if $d(x, y) = c(\omega)$.

We simply denote the cost $c(\sigma)$ of an individual edit operation $\sigma = (a \to b)$ by $c(a, b)$ instead of $c((a \to b))$, where $a, b \in \Sigma \cup \{\lambda\}$. We assume that the cost function c satisfies two conditions: $c(a, b) = c(b, a)$ and $c(a, a) = 0$, for $a, b \in \Sigma \cup \{\lambda\}$. We extend the definition of edit-distance for languages as follows:

Definition 2. The edit-distance $d(L, R)$ between two languages $L, R \subseteq \Sigma^*$ is the minimum edit-distance of two strings, one is from L and the other is from R:

$$d(L, R) = \min\{d(x, y) \mid x \in L \text{ and } y \in R\}.$$

The edit-distance in Definition 2 is the distance between the closest pair of strings from, respectively, L and R, where the distance is measured based on the cost of the edit operations. In other words, the most similar pair of strings from two languages defines the edit-distance between the languages.

4. Main algorithm

We compute the edit-distance between two languages defined by a CFG and an FA. We assume that an input CFG $G = (V, \Sigma, R, S)$ is in pseudo-CNF. We use a pseudo-CNF grammar (instead of a CNF grammar) because an arbitrary CFG can be converted to a pseudo-CNF grammar with only linear increase in size [21].

The main idea of our algorithm is to compute the edit-distance between two languages, where one language is a set of strings derived from a variable of the CFG and the other language is a set of strings spelled out by paths of the FA. We compute distances for all variables of the CFG and all pairs of states of the FA. Note that this approach is similar to the procedure for constructing a CFG that generates the intersection of the languages generated by a CFG and an FA [2].

Given a variable $A \in V$ and two states p and q of Q, let $\mathcal{C}(A, q, p)$ be the minimum edit-distance between two strings x and y, where $A \stackrel{*}{\Rightarrow} x$ and y is spelled out by a computation from q to p. Namely,

$$\mathcal{C}(A, q, p) = \min\{d(v, w) \mid v \in L(G_A) \text{ and } w \in L(M_{q,p})\},$$

where $G_A = (V, \Sigma, R, A)$ and $M_{q,p} = (Q, \Sigma, \delta, q, p)$. Then, $\min\{\mathcal{C}(S, s, f) \mid f \in F\}$ is the edit-distance between $L(G)$ and $L(M)$. In Lemma 3, we provide a recurrence for computing the $\mathcal{C}$-values.

For a variable $A \in V$ and states $q, p \in Q$, we define:

(i) $X(A, q, p) = \min\{\mathcal{C}(B, q, r) + \mathcal{C}(D, r, p) \mid r \in Q,\ A \to BD \in R\}$,
(ii) $Y(A, q, p) = \min\{\mathcal{C}(B, q, p) \mid A \to B \in R\}$, and
(iii) $Z(A, q, p) = \min\{d(a, w) \mid A \to a \in R,\ w \in L(M_{q,p})\}$.

Lemma 3. *For all variables $A \in V$ of a CFG and states $q, p \in Q$ of an FA,*

$$\mathcal{C}(A, q, p) = \min\{X(A, q, p),\ Y(A, q, p),\ Z(A, q, p)\}. \tag{1}$$

Proof. First we show that $\mathcal{C}(A, q, p)$ is not greater than the right-hand side in Equation (1).

(i) Suppose that the following conditions hold:
- $B \stackrel{*}{\Rightarrow} t$ and $D \stackrel{*}{\Rightarrow} u$, where $t, u \in \Sigma^*$,
- $r \in \delta(q, w)$ and $p \in \delta(r, w')$, where $w, w' \in \Sigma^*$, and
- $\mathcal{C}(B, q, r) = d(t, w)$ and $\mathcal{C}(D, r, p) = d(u, w')$.

If $A \to BD \in R$, then $A \stackrel{*}{\Rightarrow} tu$. We also have $p \in \delta(q, ww')$. Therefore,

$$\mathcal{C}(A, q, p) \le d(tu, ww') \le d(t, w) + d(u, w') = \mathcal{C}(B, q, r) + \mathcal{C}(D, r, p).$$

(ii) If $A \to B \in R$, then it follows that $\mathcal{C}(A, q, p) \le \mathcal{C}(B, q, p)$.

(iii) If $A \to a \in R$ for $a \in \Sigma$ and $p \in \delta(q, w)$, then $a \in L(G_A)$ and $\mathcal{C}(A, q, p) \le d(a, w)$ because $\mathcal{C}(A, q, p)$ is the minimum of a set that contains $d(a, w)$.

Hence $\mathcal{C}(A, q, p)$ is less than or equal to every term in the right-hand side in Equation (1). Next, we prove that $\mathcal{C}(A, q, p)$ is no less than the right-hand side. Assume that $\mathcal{C}(A, q, p) = d(v, w)$, where $A \overset{*}{\Rightarrow} v$ and $p \in \delta(q, w)$. Since the input CFG is in pseudo-CNF, there are three possible cases in which the derivation $A \overset{*}{\Rightarrow} v$ can begin:

(i) $A \Rightarrow BD \overset{*}{\Rightarrow} v$,
(ii) $A \Rightarrow B \overset{*}{\Rightarrow} v$,
(iii) $A \Rightarrow a = v$.

In the first case, v is defined as tu where $B \overset{*}{\Rightarrow} t$ and $D \overset{*}{\Rightarrow} u$. It follows that for some $1 \le k \le |w|$ we have

$$\mathcal{C}(A, q, p) = d(v, w) = d(t, w[1, k]) + d(u, w[k+1, |w|]) \ge \mathcal{C}(B, q, r) + \mathcal{C}(D, r, p),$$

where $r \in \delta(q, w[1, k])$ and $p \in \delta(r, w[k+1, |w|])$. Similarly in case (ii) it is easy to verify that $\mathcal{C}(A, q, p) \ge \mathcal{C}(B, q, p)$ and in case (iii) that $\mathcal{C}(A, q, p) \ge d(a, w)$. □

Note that Equation (1) is an essential recurrence equation for computing $d(L(G), L(M))$ in a bottom-up dynamic programming approach. We first compute the third term $Z(A, q, p)$—the case when there exists a production rule $A \to a \in R$—in Equation (1) since it is independent from the other terms.

We use $\overline{Z}$-values to store Z-values temporarily. After updating all $\overline{Z}$-values by considering all possible cases, we set Z-values as the final $\overline{Z}$-values.

Initially, we set

$$\overline{Z}(A, q, p) = \min\{c(a, b),\ c(a, \lambda) + c(\lambda, b)\},$$

if there exists a production rule $A \to b$ and $p \in \delta(q, a)$, where $a \in \Sigma \cup \{\lambda\}$. Note that the state p can be the same state with q if there is a self-loop transition for the state q or if $a = \lambda$.

Notice that we are able to compute the $\overline{Z}(A, q, p)$ only when A has a production rule with a character on the right-hand side like $A \to a$, and there is a transition from q to p in M, or q and p are the same state. Otherwise, $\overline{Z}(A, q, p)$ values are still undefined.

Now we need to compute the $\overline{Z}(A, q, p)$ values for the pairs of states where there is no transition. We can deal with the case by computing the minimum-cost paths between all pairs of states q and p. We transform the FA M into a weighted directed graph $G_M = (V_M, E_M)$, where V_M is a set of nodes and E_M is a set of weighted edges. We set $V_M = Q$ and $E_M = \{(q, p) \mid p \in \delta(q, a), a \in \Sigma\}$. Namely, each node of G_M corresponds to the states of M. Then, we define the weight function $f : E_M \to \mathbb{R}$ as follows:

$$f((q, p)) = \min\{c(a, \lambda) \mid p \in \delta(q, a),\ a \in \Sigma\}.$$

After this, we run the Floyd–Warshall algorithm [8] on the graph to find the minimum-cost paths between all pairs of nodes in the graph. Then, the cost of the minimum-cost path from q to p is the cost of the minimum-cost string spelled out by the path in M from q to p. We denote the cost of the minimum-cost string spelled out by the path from q to p by $\mathcal{D}(q, p)$. Now we can compute the undefined $\overline{Z}(A, q, p)$ values by using $\mathcal{D}$ values. The procedure is described in lines 1–14 of Algorithm 1. Now we set $Z(A, q, p) = \overline{Z}(A, q, p)$ for all $q, p \in Q$ and $A \in V$.

Once we have calculated all $Z(A, q, p)$ values, we repeatedly compute $\mathcal{C}(A, q, p)$ by taking the minimum from $X(A, q, p)$ and $Y(A, q, p)$. We need to repeat the computation at most $|Q|^2|V|$ times since the value of $\mathcal{C}(A, q, p)$ cannot be computed based on the value of $\mathcal{C}(A, q, p)$ itself.

We analyze the runtime of Algorithm 1.

Theorem 4. *Given a CFG $G = (V, \Sigma, R, S)$ and an FA $M = (Q, \Sigma, \delta, s, F)$, we can compute the edit-distance between $L(G)$ and $L(M)$ in $O(m^2n^5)$ worst-case time, where $m = |G|$, $n = |M|$.*

Proof. Note that given a graph with k nodes, the Floyd–Warshall algorithm runs in $O(k^3)$ time [8]. Thus, lines 1–3 can be executed in $O(|Q|^3)$ time by converting M into a directed weighted graph G_M and computing the cost of the minimum-cost paths between all pairs of nodes in G_M. Lines 4–8 take $O(|\delta||P|)$ time. Lines 9–14 also take $O(|Q|^3|V|)$ time since we iterate the **for**-loops for all pairs of a state triple (q, r, p) and a variable A, where $q, r, p \in Q$ and $A \in V$.

After executing lines 1–14, $\overline{C}(A, q, p) = Z(A, q, p)$ must hold since we have considered every possible matching for a production rule $A \to a \in R$ and a pair of states q and p. Now we have to compare $\overline{C}(A, q, p)$ with $X(A, q, p)$ and $Y(A, q, p)$, and find the minimum value to compute $\mathcal{C}(A, q, p)$ values as defined in Lemma 3. Lines 16 and 19 show the steps in which we compare $\overline{C}(A, q, p)$ with $X(A, q, p)$ and $Y(A, q, p)$, respectively.

The **while**-loop in lines 15–24 can be executed at most $|Q|^2|V|$ times. Assume that we have an optimal matching ω between $L(G_A)$ and $L(M_{q,p})$. Then, the cost $c(\omega)$ of the optimal matching is $\mathcal{C}(A, q, p)$. Since we have already computed

Algorithm 1 The algorithm for computing $d(L(G), L(M))$.

Input: A CFG $G = (V, \Sigma, R, S)$ and an FA $M = (Q, \Sigma, \delta, s, F)$

```
1: for q, p ∈ Q do
2:     D(q, p) ← min{c(ω) | h(ω) = (w, λ), w ∈ L(M_{q,p})}
3: end for
4: for p, q ∈ Q and A → b ∈ R do
5:     if p ∈ δ(q, a) then
6:         C̄(A, q, p) ← min{c(a, b), c(a, λ) + c(λ, b)}
7:     end if
8: end for
9: for q, r, p ∈ Q and A ∈ V do
10:    C̄(A, q, p) ← min{C̄(A, q, p), D(q, r) + C̄(A, r, p)}
11: end for
12: for q, r, p ∈ Q and A ∈ V do
13:    C̄(A, q, p) ← min{C̄(A, q, p), C̄(A, q, r) + D(r, p)}
14: end for
15: while there is an updated C̄-value do
16:    for q, p ∈ Q do
17:        for A → BD ∈ R and r ∈ Q do
18:            C̄(A, q, p) ← min{C̄(A, q, p), C̄(B, q, r) + C̄(D, r, p)}
19:        end for
20:        for A → B ∈ R do
21:            C̄(A, q, p) ← min{C̄(A, q, p), C̄(B, q, p)}
22:        end for
23:    end for
24: end while
25: return min{C̄(S, s, f) | f ∈ F}
```

Output: $d(L(G), L(M))$

$Z(A, q, p)$, now $\overline{\mathcal{C}}(A, q, p)$ is the optimal matching between two strings, where a string from G is derived by one production rule of form $A \to a$ for $A \in V$ and $a \in \Sigma$. Assume that we process the **while**-loop in lines 15–24 once and have an updated $\overline{\mathcal{C}}(A, q, p)$-value in line 18. After this $\overline{\mathcal{C}}(A, q, p)$ is the cost of an optimal matching between a string derived by G_A in two steps and a string of $L(M_{q,p})$. Note that $\overline{\mathcal{C}}(A, q, p)$ can be updated based on the other $\overline{\mathcal{C}}$-values since we take the minimum between two values. This implies that the **while**-loop in lines 15–24 can be executed at most $|Q|^2|V|$ times because otherwise we always have to update $\overline{\mathcal{C}}$-value based on the same value.

The computation of lines 16–23 takes $O(|Q|^3|R|)$ time since we need to consider all production rules of G and all state triples of M in the double **for**-loops in lines 16 and 17. Therefore, the algorithm to compute the edit-distance between $L(G)$ and $L(M)$ runs in time $O(|Q|^5|R||V|)$ and this value is upper bounded by $O(m^2n^5)$. □

Once we have calculated the edit-distance, we can also retrieve an optimal alignment by the traceback of the computation.

Theorem 5. *Given a CFG $G = (V, \Sigma, R, S)$ and an FA $M = (Q, \Sigma, \delta, s, F)$, we can compute the optimal alignment between $L(G)$ and $L(M)$ in $O(mnk)$ worst-case time, where $m = |G|$, $n = |M|$ and k is the length of the optimal alignment.*

Proof. We can retrieve the optimal alignment between a CFG G and an FA M that results in the edit-distance by the traceback. We can assume that all $\mathcal{C}$-values are already computed during the computation of the edit-distance.

From $\mathcal{C}(S, s, f)$, we trace back to the initial step to find the optimal alignment. For example, we check where the value of $\mathcal{C}(S, s, f)$ is from by comparing $\mathcal{C}(S, s, f)$ with the following cases:

(i) $c(\lambda, b)$ if $s = f$ and $S \to b \in R$;
(ii) $c(a, b)$, $c(a, \lambda) + c(\lambda, b)$ if there is a production rule $S \to b \in R$ and $f \in \delta(s, a)$;
(iii) $\mathcal{D}(s, r_1) + \mathcal{C}(S, r_1, r_2) + \mathcal{D}(r_2, f)$;
(iv) $\mathcal{C}(B, s, r) + \mathcal{C}(D, r, f)$ if there is a production rule $S \to BD \in R$; and
(v) $\mathcal{C}(A, s, f)$ if there is a production rule $S \to A \in R$.

By Algorithm 1, one of the five values should be equal to $\mathcal{C}(S, s, f)$. Now assume that $\mathcal{C}(S, s, f) = \mathcal{C}(B, s, r) + \mathcal{C}(D, r, f)$. Then, we check where the values $\mathcal{C}(B, s, r)$ and $\mathcal{C}(D, r, f)$ are from in a similar way. We repeat this procedure until we find matchings between all $\mathcal{C}$-values and base cases (i)–(iii). After finding all matchings for all $\mathcal{C}$-values, we can retrieve the corresponding alignment. Since we consider all production rules of G and states of M at each step of the traceback, it takes $O(mnk)$ time to retrieve the optimal alignment, where k is the length of the optimal alignment. Note that k can be exponential in the size of G as the shortest string in $L(G)$ can be of length $2^{|V|-1}$. If the FA M only accepts an empty string λ and the length of the shortest string accepted by G is $2^{|V|-1}$, then the length of the optimal alignment between $L(G)$ and $L(M)$ should be exponential in the size of G. We present an asymptotic upper bound for the length k of an optimal alignment between $L(G)$ and $L(M)$ by analyzing Algorithm 1. After executing lines 1–12, we have optimal alignments

(a)

$x =$	A	C	T	T	A	G	T	A	G	A	T	C	C
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$y =$	A	C	T	T	$-$	G	$-$	A	$-$	$-$	T	C	C

(b)

$x =$	A	C	T	T	A	G	T	A	G	A	T	C	C
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$y =$	A	C	T	T	$-$	$-$	$-$	$-$	G	A	T	C	C

Fig. 1. Two examples of aligning x and y. The first alignment (a) has three short gaps and the second alignment (b) has one long gap.

of length $O(|Q|)$ since we only consider the matchings between a string of length one derived by a production rule of form $A \to a \in R$ and a string from an FA M. Now we repeat the computations $O(|Q|^2|V|)$ times and each computation doubles the length of optimal alignments in the worst-case.

Therefore, the asymptotic upper bound for k is $O(|Q| \cdot 2^{|Q|^2|V|})$ which is upper bounded by $O(n \cdot 2^{n^2 m})$.

Finally, we mention that we can retrieve an optimal alignment in $O(k)$ time if we save back-pointers when we run Algorithm 1 since we do not need to scan all production rules of G and states of M at each step. □

5. Affine gap distance

The approximate pattern matching problem is closely related to the sequence alignment problem in bioinformatics [19,20]. A biological sequence alignment is a process of arranging sequences of nucleotides describing DNA or RNA molecules, or protein amino acid sequences, and examining the similarity between the two sequences.

Consider the two alignments in Fig. 1. Both alignments have gaps—sequences of deletions, or sequences of insertions—of total length four. However, the second alignment is biologically better since the deletion of four consecutive elements is more likely to occur than the deletion or insertion of three separated elements. Therefore, it is desirable to assign a higher penalty to an alignment containing many short gaps than few long gaps. Assume that an alignment ω has k consecutive insertions or deletions—a gap of length k. Then, the cost of ω is linearly proportional to $|\omega|$. Namely, $c(\omega) = g \cdot |\omega|$, for a constant g. Instead of using this linear gap penalty function, we can use the *affine gap penalty* function that helps to obtain biologically better alignments [3,7].

Here the alphabet Ω of edit operations consists of deletions $(a \to \lambda)$, insertions $(\lambda \to a)$, substitutions $(a \to b)$ that change the symbol, and trivial substitutions $(a \to a)$ that do not change the symbol. We denote $\Omega_{\text{del}} = \{(a \to \lambda) \mid a \in \Sigma\}$, $\Omega_{\text{ins}} = \{(\lambda \to a) \mid a \in \Sigma\}$, $\Omega_{\text{sub}} = \{(a \to b) \mid a, b \in \Sigma, a \neq b\}$ and $\Omega_{\text{triv}} = \{(a \to a) \mid a \in \Sigma\}$, and thus

$$\Omega = \Omega_{\text{del}} \cup \Omega_{\text{ins}} \cup \Omega_{\text{sub}} \cup \Omega_{\text{triv}}.$$

Let $\omega \in \Omega^+$ be a sequence of edit operations. The *(maximal) IDS-decomposition of* ω (insertion–deletion–substitution decomposition of ω) is

$$\text{comp}_{\text{IDS}}(\omega) = (\omega_1, \omega_2, \ldots, \omega_k),$$

where $\omega_1\omega_2 \ldots \omega_k = \omega, \omega_i \in \Omega^+_{\text{del}} \cup \Omega^+_{\text{ins}} \cup \Omega^+_{\text{sub}} \cup \Omega^+_{\text{triv}}$, for $i = 1, \ldots, k$, and for any $1 \leq j < k$ the strings ω_j and ω_{j+1} belong to different sets Ω^+_{del}, Ω^+_{ins}, Ω^+_{sub} and Ω^+_{triv}, respectively.

Given ω, we can obtain the IDS-decomposition of ω by subdividing ω into maximal substrings each of which consists of only insertions, only deletions, only substitutions, or only trivial substitutions. Thus, $\text{comp}_{\text{IDS}}(\omega)$ is uniquely defined.

We define the affine gap cost of an IDS-decomposition as follows:

$$c_{\text{affine}}(\omega) = \begin{cases} l + g \cdot |\omega| & \text{if } \omega \in \Omega^+_{\text{del}} \cup \Omega^+_{\text{ins}}, \\ c(\omega) & \text{if } \omega \in \Omega^+_{\text{sub}}, \\ 0 & \text{if } \omega \in \Omega^+_{\text{triv}}, \end{cases}$$

where l and g are constants.

We then define the *affine gap cost* of an arbitrary sequence of edit operations $\omega \in \Omega^+$, where $\text{comp}_{\text{IDS}}(\omega) = (\omega_1, \omega_2, \ldots, \omega_k)$, to be

$$c_{\text{affine}}(\omega) = \sum_{i=1}^{k} c_{\text{affine}}(\omega_i).$$

For a sequence of edit operations, the affine gap cost gives a constant l penalty for each gap opening (consisting of consecutive insertions or consecutive deletions) and an additional penalty that is linear in the length of the gap. We call the edit-distance based on the affine gap cost function the *affine gap distance*. We design an algorithm for computing the affine gap distance between a CFG and an FA. This is an extension of the previous algorithm, yet has the same time complexity. The main difference is that we define $\mathcal{C}$-values with two additional parameters as follows:

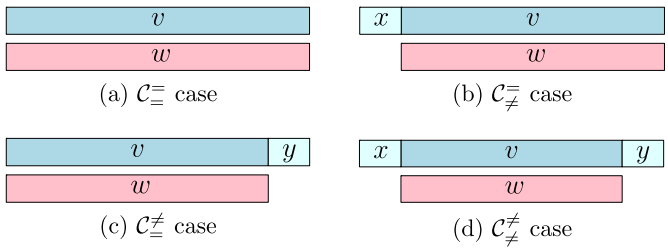

Fig. 2. The pictorial representations of $\mathcal{C}^{\lhd}_{\rhd}$-values, where $\rhd, \lhd \in \{=, \neq\}$.

$$\mathcal{C}^{\lhd}_{\rhd}(A, q, p) = \min\{c_{\text{affine}}(\omega_1) + c_{\text{affine}}(\omega_2) + c_{\text{affine}}(\omega_3) \mid A \overset{*}{\Rightarrow} xvy,\ p \in \delta(q, w),$$
$$h(\omega_1) = (x, \lambda), h(\omega_2) = (v, w), h(\omega_3) = (y, \lambda),\ |x| \rhd 0 \text{ and } |y| \lhd 0\},$$

where $\lhd, \rhd \in \{=, \neq\}$.

The additional parameters $\rhd$ and $\lhd$ imply the existence of end-gaps on both sides. Note that these parameters only consider the existence of gaps, where the substrings generated from a CFG are not matched. We remark that the parameters $\rhd$ and $\lhd$ denote the existence of the sequences of deletions on both sides, not insertions. See Fig. 2 for examples. In $\mathcal{C}^{\lhd}_{\rhd}(A, q, p)$, we denote the existence of the end-gap on the left side and the right side by $\rhd$ and $\lhd$, respectively. Thus, $\mathcal{C}^{\neq}_{=}(A, q, p)$ is the cost of the optimal matching, where the matching has no gap on the left side and has a gap on the right side as depicted in Fig. 2(c).

Now the affine gap distance between a CFG and an FA becomes

$$\min\{\mathcal{C}^{\lhd}_{\rhd}(S, s, f) \mid f \in F \text{ and } \rhd, \lhd \in \{=, \neq\}\}.$$

Before introducing the recurrence for $\mathcal{C}^{\lhd}_{\rhd}$-values, for a variable $A \in V$ and states $q, p \in Q$ of M, we define the following notations. Note that when defining $X^{\lhd}_{\rhd}(A, q, p)$ we use the symbols $\trianglerighteq$ and $\trianglelefteq$ to represent either $=$ or $\neq$ and then minimize over the different possibilities.

(i) $X^{\lhd}_{\rhd}(A, q, p) = \min\{\mathcal{C}^{\trianglelefteq}_{\rhd}(B, q, r) + \mathcal{C}^{\lhd}_{\trianglerighteq}(D, r, p) - l' \mid r \in Q,\ A \to BD \in R \text{ and } \trianglerighteq, \trianglelefteq \in \{=, \neq\}\}$, where

$$l' = \begin{cases} l & \text{if } \trianglerighteq = \trianglelefteq = \text{`}\neq\text{'}, \\ 0 & \text{otherwise,} \end{cases}$$

(ii) $Y^{\lhd}_{\rhd}(A, q, p) = \min\{\mathcal{C}^{\lhd}_{\rhd}(B, q, p) \mid A \to B \in R\}$, and
(iii) $Z^{\lhd}_{\rhd}(A, q, p) = \min\{\mathcal{I}^{\lhd}_{\rhd}(a, w) \mid A \to a \in R,\ a \in \Sigma \text{ and } w \in L(M_{q,p})\}$.

Here, $\mathcal{I}$-values are defined as follows:

- $\mathcal{I}^{=}_{=}(a, w) = \min_{k \in [1, |w|]}\{c(a, w[k]) + (|w| - 1) \cdot g + l''\}$, where

$$l'' = \begin{cases} l & \text{if } k = 1 \text{ or } k = |w|, \\ 2l & \text{otherwise,} \end{cases}$$

- $\mathcal{I}^{\neq}_{\neq}(a, w) = g + l$, and
- $\mathcal{I}^{=}_{\neq}(a, w) = \mathcal{I}^{\neq}_{=}(a, w) = (|w| + 1) \cdot g + 2l$.

Now we are ready to establish a recursive definition for $\mathcal{C}^{\lhd}_{\rhd}$-values for computing the affine gap distance between a CFG and an FA.

Lemma 6. *For all variables $A \in V$ of a CFG $G = (V, \Sigma, R, S)$ and states $q, p \in Q$ of an FA $M = (Q, \Sigma, \delta, s, F)$,*

$$\mathcal{C}^{\lhd}_{\rhd}(A, q, p) = \min\{X^{\lhd}_{\rhd}(A, q, p),\ Y^{\lhd}_{\rhd}(A, q, p),\ Z^{\lhd}_{\rhd}(A, q, p)\}.$$

Proof. We prove the statement in a way similar to the proof of Lemma 3. The first term in the recurrence is almost the same as in Lemma 3, yet we subtract l when two gaps merge into one. For the productions $A \to a$, we define four types of $\mathcal{I}^{\lhd}_{\rhd}$-values depending on $\rhd$ and $\lhd$.

We depict the four cases in Fig. 3. For the $\mathcal{I}^{=}_{=}$ case in Fig. 3(a), we give the cost for substituting a character a with a character $w[k]$ in the middle of the string w and the linear penalty $(|w| - 1) \cdot g$ and the gap opening penalty $2l$ since there are two gaps such that the sum of the gap lengths is $|w| - 1$. Note that if we substitute a character $w[k]$, where k is 1 or

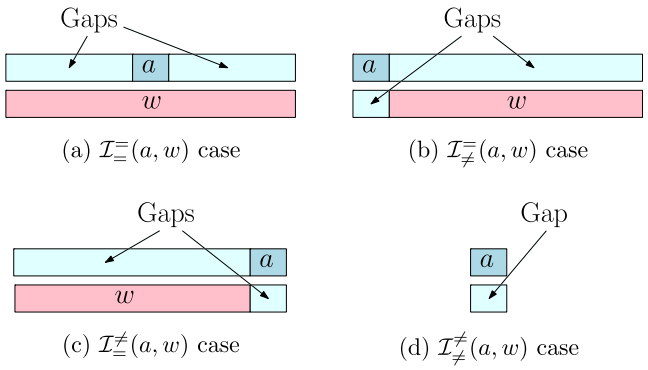

Fig. 3. The pictorial representations of $\mathcal{I}^{\triangleleft}_{\triangleright}$-values where $\triangleright, \triangleleft \in \{=, \neq\}$.

$|w|$, then the gap opening penalty should be l since there is only one gap. Now we consider the $\mathcal{I}^{=}_{\neq}$ and $\mathcal{I}^{\neq}_{=}$ cases depicted in Fig. 3(b) and Fig. 3(c). Since there are two gaps of total length $|w|+1$, we assign the linear penalty $(|w|+1)\cdot g$ and the gap opening penalty $2l$. Lastly, Fig. 3(d) represents the $\mathcal{I}^{\neq}_{\neq}$ case. Since there is only one gap of length 1, the cost becomes $g+l$. After these observations the remaining proof is similar to the proof for Lemma 3. □

Note that the time complexity of this algorithm remains essentially the same as the time used by Algorithm 1. Since we consider the four variations of $\mathcal{C}$-values, the time complexity only increases to four times the runtime of Algorithm 1.

Theorem 7. *Given a CFG G and an FA M, we can compute the affine gap distance between $L(G)$ and $L(M)$ in $O(m^2n^5)$ worst-case time, where $m = |G|$ and $n = |M|$.*

Theorem 8. *Given a CFG G and an FA M, we can compute the optimal alignment for the affine gap distance between $L(G)$ and $L(M)$ in $O(mnk)$ worst-case time, where $m = |G|$, $n = |M|$ and k is the length of the optimal alignment.*

6. Concave gap distance

Several researchers considered non-linear gap penalty functions, including the affine gap penalty function [3,7,12,16,23]. Although the affine gap penalty function prefers few longer gaps to many smaller gaps, we cannot guarantee that the affine gap penalty function always produces the best result. For example, assume that there are two alignments s_1 and s_2 for two sequences—s_1 contains two gaps of length 99 and 100, and s_2 contains just one gap of length 240. Let us assume that the remaining parts of s_1 and s_2 are perfectly matched without any substitutions. By employing the affine gap penalty function with $l = 5$ and $g = 1$, we obtain $c(s_1) = 99 + 100 + 5 \times 2 = 209$ and $c(s_2) = 240 + 5 = 245$. Even though the gap opening penalty is considered in the affine gap distance, it may not be optimal for some practical cases such as this example. This is why the *concave gap distance* is introduced [12,16,23].

Now we define the concave gap cost of ω as follows:

$$c_{\text{concave}}(\omega) = \begin{cases} l + g \cdot \log|\omega| & \text{if } \omega \in \Omega^{+}_{\text{del}} \cup \Omega^{+}_{\text{ins}}, \\ c(\omega) & \text{if } \omega \in \Omega^{+}_{\text{sub}}, \\ 0 & \text{if } \omega \in \Omega^{+}_{\text{triv}}, \end{cases}$$

where l and g are constants.

The *concave gap cost* of an arbitrary sequence of edit operations $\omega \in \Omega^{+}$, where $\text{comp}_{\text{IDS}}(\omega) = (\omega_1, \omega_2, \ldots, \omega_k)$ is then

$$c_{\text{concave}}(\omega) = \sum_{i=1}^{k} c_{\text{concave}}(\omega_i).$$

Under this gap penalty function, the shape of the penalty score with respect to the length of the gap is concave in the sense that its forward differences are non-increasing. Fig. 4 illustrates the difference among the three gap penalty functions: linear, affine and concave gap penalty functions.

We define new $\mathcal{C}$-values for computing the concave gap distance as follows:

$$\mathcal{C}^{j}_{i}(A,q,p) = \min\{c_{\text{concave}}(\omega_1) + c_{\text{concave}}(\omega_2) + c_{\text{concave}}(\omega_3) \mid A \overset{*}{\Rightarrow} xvy,$$
$$p \in \delta(q,w), h(\omega_1) = (x,\lambda), h(\omega_2) = (v,w), h(\omega_3) = (y,\lambda), |x| = i \text{ and } |y| = j\}.$$

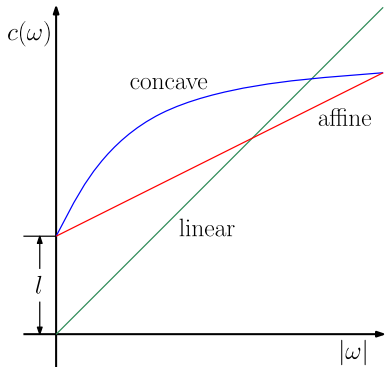

Fig. 4. Comparison of the linear, affine and concave gap costs for a sequence of deletions or insertions ω.

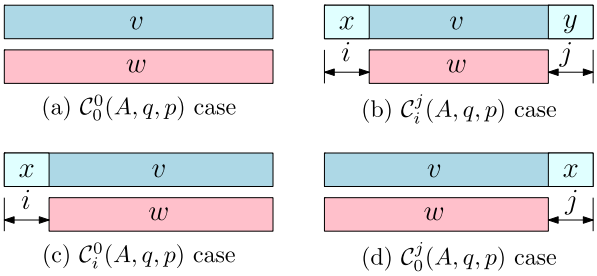

Fig. 5. The pictorial representations of $\mathcal{C}$-values where $i, j \in \mathbb{N}$.

We use two additional parameters for maintaining the lengths of gaps on both sides. In $\mathcal{C}_i^j(A,q,p)$, i and j mean the length of the end-gap on the left side and the length of the end-gap on the right side, respectively. We depict the pictorial representations of the four cases in Fig. 5.

We let $\mathcal{V}(t)$ be the set of variables that can derive terminal strings of length t:

$$\mathcal{V}(t) = \{A \mid A \in V,\ w \in L(G_A) \text{ and } |w| = t\}.$$

We can compute $\mathcal{V}(t)$ as follows:

$$\bigcup_{k=1}^{t-1} \{A \mid A \to BD \in R,\ (B,D) \in \mathcal{V}(k) \times \mathcal{V}(t-k)\} \cup \{A \mid A \to B \in R,\ B \in \mathcal{V}(t)\} \cup Z_t,$$

where

$$Z_t = \begin{cases} \{A \mid A \to a \in R\} & \text{if } t = 1, \\ \emptyset & \text{if } t \neq 1. \end{cases}$$

Next, before introducing the recurrence for $\mathcal{C}$-values for the concave gap distance, we define the following values corresponding to a variable $A \in V$ and states $q, p \in Q$ of M:

(i) $X_i^j(A,q,p) = \min\{\mathcal{C}_i^m(B,q,r) + \mathcal{C}_n^j(D,r,p) + l_1 \mid r \in Q,\ A \to BD \in R\}$, where

$$l_1 = \begin{cases} g \cdot \log|\frac{m+n}{mn}| - l & \text{if } mn \neq 0, \\ 0 & \text{otherwise,} \end{cases}$$

(ii) $Y_i^j(A,q,p) = \min\{\mathcal{C}_i^{j-t}(B,q,p) + l_2 \mid A \to BD \in R,\ t \in [1,j],\ D \in \mathcal{V}(t)\}$, where

$$l_2 = \begin{cases} l + g \cdot \log j & \text{if } j = t, \\ g \cdot \log \frac{j}{j-t} & \text{otherwise,} \end{cases}$$

(iii) $Z_i^j(A,q,p) = \min\{\mathcal{C}_{i-t}^j(D,q,p) + l_3 \mid A \to BD \in R,\ t \in [1,i],\ B \in \mathcal{V}(t)\}$, where

$$l_3 = \begin{cases} l + g \cdot \log i & \text{if } i = t, \\ g \cdot \log \frac{i}{i-t} & \text{otherwise,} \end{cases}$$

(iv) $U_i^j(A, q, p) = \min\{C_i^j(B, q, p) \mid A \to B \in R\}$, and
(v) $W_i^j(A, q, p) = \min\{\mathcal{I}_i^j(a, w) \mid A \to a,\ 0 \le i, j \le 1,\ w \in L(M_{q,p})\}$.

Here, $\mathcal{I}$-values are defined as follows:

- $\mathcal{I}_0^0(a, w) = \begin{cases} c(a, w[k]) + l + \log(|w| - 1) & \text{if } k = 1 \text{ or } |w|, \\ c(a, w[k]) + 2l + \log((k-1)(|w| - k)) & \text{otherwise,} \end{cases}$
- $\mathcal{I}_1^0(a, w) = \mathcal{I}_0^1(a, w) = 2l + \log|w|$.

Now we establish a recurrence for computing the concave gap distance between a CFG and an FA.

Lemma 9. *For all variables $A \in V$ of a CFG $G = (V, \Sigma, R, S)$, states $q, p \in Q$ of an FA $M = (Q, \Sigma, \delta, s, F)$ and $i, j \in [0, (|Q| - 1) \cdot 2^{\frac{l}{g} + |V| - 1}]$,*

$$C_i^j(A, q, p) = \min\{X_i^j(A, q, p),\ Y_i^j(A, q, p),\ Z_i^j(A, q, p),\ U_i^j(A, q, p),\ W_i^j(A, q, p)\}.$$

Proof. Recall that we use i and j to maintain the lengths of gaps on both sides. Note that i and j imply that there are two gaps of length i and j at both ends, where the substrings generated from the CFG G are not matched. In other words, $C_i^j(A, q, p)$ is the cost of an optimal alignment between $L(G_A)$ and $L(M_{q,p})$, where the alignment has two gaps of length i and j at both ends.

Now we show that the range of the two parameters i and j is from 0 to $(|Q| - 1) \cdot 2^{\frac{l}{g} + |V| - 1}$. First we establish an upper bound of the gaps, where the substrings generated from the CFG are not matched from the upper bound of the concave gap distance between $L(G)$ and $L(M)$. The length of the shortest string w_G in $L(G)$ can be at most $2^{|V|-1}$ as all paths of the derivation tree from the root to a terminal can contain at most $|V|$ variables. This means that the height of the tree is at most $|V|$, thus, the length of the shortest string is at most $2^{|V|-1}$. The length of the shortest string w_M in $L(M)$ is at most $|Q| - 1$. Now there exists an alignment ω of w_G and w_M such that $\omega = \omega_G \omega_M$ and $h(\omega_G) = (\lambda, w_G)$ and $h(\omega_M) = (w_M, \lambda)$. Since ω consists of two gaps whose lengths are at most $2^{|V|-1}$ and $|Q| - 1$, the concave gap distance for ω_G and ω_M is $2l + g \cdot (\log 2^{|V|-1} + \log(|Q| - 1))$, which is an upper bound of the concave gap distance between $L(G)$ and $L(M)$. Now, assume that there exists only one gap in an optimal alignment between $L(G)$ and $L(M)$. Since the concave gap distance can be at most $2l + g \cdot (\log 2^{|V|-1} + \log(|Q| - 1))$, the gap length can be upper bounded as follows:

$$\begin{aligned} 2l + g \cdot (\log 2^{|V|-1} + \log(|Q| - 1)) &= l + g \cdot (\tfrac{l}{g} + \log 2^{|V|-1} + \log(|Q| - 1)) \\ &= l + g \cdot (\tfrac{l}{g} + \log((|Q| - 1) \cdot 2^{|V|-1})) \\ &= l + g \cdot \log((|Q| - 1) \cdot 2^{\frac{l}{g} + |V| - 1}). \end{aligned}$$

Now we show that every term used to define $C_i^j(A, q, p)$ can be a possible candidate of $C_i^j(A, q, p)$. The first term is for the case depicted in Fig. 6(a). Two gaps of lengths m and n are merged into a gap of length $m + n$. We add

$$g \cdot (\log|m + n| - \log|m| - \log|n|) = g \cdot \log\left|\frac{m+n}{mn}\right|$$

since we replace two gaps of length m and n into one gap of length $m + n$. Since the number of gaps decreases as two gaps merge into one, we also subtract the gap opening penalty l. If $mn = 0$, then there is only one gap or no gap where the two alignments are merged since $m = 0$ or $n = 0$ holds. The case for $Y_i^j(A, q, p)$ is depicted in Fig. 6(b)—the string w is aligned only with a terminal string derived from D. If the length of the terminal string derived from D is j, then we add $l + g \cdot \log j$ since we put a gap of length j to the end where there is no gap. Otherwise, we add $g \cdot \log \frac{j}{j-t}$ since we change the length of the gap from $j - t$ to t by putting the gap of length t to the gap of the length $j - t$. The case for $Z_i^j(A, q, p)$ is defined in a similar way as in the $Y_i^j(A, q, p)$ case.

The case for $U_i^j(A, q, p)$ is when there is a production rule $A \to B \in R$ and therefore the value of $C_i^j(B, q, p)$ can be a candidate for $C_i^j(A, q, p)$. Lastly, $W_i^j(A, q, p)$ is the case when a character a generated from a variable $A \in V$ is aligned with a string w generated by a path of the FA M from a state q to a state p.

Since we have considered all the possible cases for aligning two strings where one string is from $L(G_A)$ and the other string is from $L(M_{q,p})$, we complete the proof. □

Based on the recurrence and the restriction on the sizes of i and j, we can compute the concave gap distance between $L(G)$ and $L(M)$ in exponential runtime.

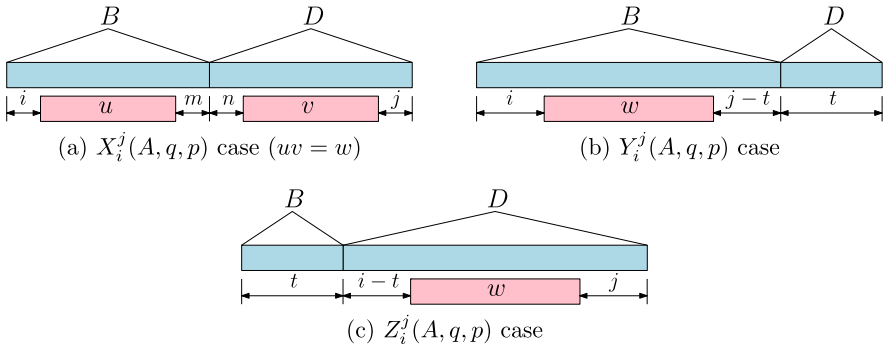

Fig. 6. The pictorial representations of three cases in the recursive definition of $\mathcal{C}$-values.

Theorem 10. *Given a CFG $G = (V, \Sigma, R, S)$ and an FA $M = (Q, \Sigma, \delta, s, F)$, we can compute the concave gap distance between $L(G)$ and $L(M)$ in $O(m^5n^8 8^m \log^3 n)$ worst-case time, where $m = |G|$ and $n = |M|$.*

Proof. We can compute the concave gap distance between $L(G)$ and $L(M)$ by computing $\mathcal{C}$-values defined in Lemma 9. Among all $\mathcal{C}$-values, the minimum of $\mathcal{C}_i^j(S, s, f)$, where $f \in F$ and $i, j \in [0, (|Q| - 1) \cdot 2^{\frac{l}{g}+|V|-1}]$, is the concave gap distance between $L(G)$ and $L(M)$. We need to compute all $\mathcal{C}$-values in the worst-case. Since the upper bound of the concave gap distance is $2l + g \cdot (\log 2^{|V|-1} + \log(|Q| - 1))$, the multiplication of the lengths of two end-gaps should be at most $2^{|V|-1} \cdot (|Q| - 1)$. Therefore, the number of the lengths of two end-gaps to be computed is

$$\sum_{i=1}^{2^{|V|-1}\cdot(|Q|-1)} \frac{2^{|V|-1} \cdot (|Q| - 1)}{i}.$$

Since

$$\sum_{i=1}^{k} \frac{k}{i} \in O(k \log k),$$

the upper bound of the number of entries to be computed is

$$O(|V||Q|^2 \cdot |Q||V| \cdot 2^{|V|} \cdot \log|Q|) = O(|V|^2|Q|^3 \cdot 2^{|V|} \cdot \log|Q|)$$

if we assume g and l as constants.

First, we need

$$O(|Q|^5|R||V|^2 \cdot 4^{|V|} \cdot \log^2|Q|)$$

time to compute all $\mathcal{C}$-values once. Note that the computed $\mathcal{C}(A, q, p)$-value is not necessarily correct since the value can be updated as we iterate the process. Note that the number of iterations required to compute correct $\mathcal{C}$-values is equivalent to the number of entries to be computed. This implies that we need to iterate $O(|V|^2|Q|^3 \cdot 2^{|V|} \cdot \log|Q|)$ times. Now the worst-case time complexity for computing the concave gap distance between $L(G)$ and $L(M)$ is $O(|V|^4|R||Q|^8 \cdot 8^{|V|} \cdot \log^3|Q|)$ which is upper bounded by $O(m^5n^8 8^m \log^3 n)$, where m is the size of the CFG G and n is the size of the FA M. □

7. Conclusions

We have considered the problem of approximately matching a context-free language and a regular language. We have examined three types of gap cost functions that are used for approximate string matching: linear, affine and concave. Based on the dynamic programming approach, we have introduced algorithms for computing the linear, affine and concave gap distance between a CFG and an FA.

Given a CFG of size m and an FA of size n, we have presented algorithms for computing linear and affine gap distances in $O(m^2n^5)$ time. We have also shown that computing the optimal alignment of length k takes $O(mnk)$ time by our algorithm when we consider linear or affine gap distance. Finally, we have proposed an $O(m^5n^8 8^m \log^3 n)$ time algorithm for computing the concave gap distance.

It will be interesting to see if we can compute the max–min distance between a CFG and an FA, or find k optimal alignments between a CFG and an FA.

Acknowledgments

We thank the anonymous referees for a careful reading of an earlier version of the paper and for many useful suggestions that have improved the presentation.

Ko and Han were supported by the Basic Science Research Program through NRF funded by MEST (2015R1D1A1A010600 97), the Yonsei University Future-leading Research Initiative of 2015 and the International Cooperation Program managed by NRF of Korea (2014K2A1A2048512), and Salomaa was supported by the Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.

References

[1] A. Aho, T. Peterson, A minimum distance error-correcting parser for context-free languages, SIAM J. Comput. 1 (4) (1972) 305–312.
[2] Y. Bar-Hillel, M. Perles, E. Shamir, On formal properties of simple phrase structure grammars, in: Y. Bar-Hillel (Ed.), Language and Information: Selected Essays on Their Theory and Application, Addison–Wesley, Reading, Massachusetts, 1964, pp. 116–150, chapter 9.
[3] G.J. Barton, M.J.E. Sternberg, Evaluation and improvements in the automatic alignment of protein sequences, Protein Eng. 1 (2) (1987) 89–94.
[4] E. Brill, R.C. Moore, An improved error model for noisy channel spelling correction, in: Proceedings of the 38th Annual Meeting on Association for Computational Linguistics, 2000, pp. 286–293.
[5] C. Choffrut, G. Pighizzini, Distances between languages and reflexivity of relations, Theor. Comput. Sci. 286 (1) (2002) 117–138.
[6] J. Earley, An efficient context-free parsing algorithm, Commun. ACM 13 (2) (1970) 94–102.
[7] W.M. Fitch, T.F. Smith, Optimal sequence alignments, in: Proceedings of the National Academy of Sciences, vol. 80, 1983, pp. 1382–1386.
[8] R.W. Floyd, Algorithm 97: shortest path, Commun. ACM 5 (6) (1962) 345–348.
[9] Y.-S. Han, S.-K. Ko, K. Salomaa, The edit-distance between a regular language and a context-free language, Int. J. Found. Comput. Sci. 24 (7) (2013) 1067–1082.
[10] J. Hopcroft, J. Ullman, Introduction to Automata Theory, Languages, and Computation, 2nd ed., Addison–Wesley, Reading, MA, 1979.
[11] L. Kari, S. Konstantinidis, Descriptional complexity of error/edit systems, J. Autom. Lang. Comb. 9 (2004) 293–309.
[12] J.R. Knight, E.W. Myers, Approximate regular expression pattern matching with concave gap penalties, Algorithmica 14 (1995) 67–78.
[13] S. Konstantinidis, Computing the edit distance of a regular language, Inf. Comput. 205 (2007) 1307–1316.
[14] V.I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Sov. Phys. Dokl. 10 (8) (1966) 707–710.
[15] G. Lyon, Syntax-directed least-errors analysis for context-free languages: a practical approach, Commun. ACM 17 (1) (1974) 3–14.
[16] W. Miller, E.W. Myers, Sequence comparison with concave weighting functions, Bull. Math. Biol. 50 (2) (1988) 97–120.
[17] M. Mohri, Edit-distance of weighted automata: general definitions and algorithms, Int. J. Found. Comput. Sci. 14 (6) (2003) 957–982.
[18] G. Myers, Approximately matching context-free languages, Inf. Process. Lett. 54 (1995) 85–92.
[19] S.B. Needleman, C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. Mol. Biol. 48 (3) (1970) 443–453.
[20] D. Sankoff, J.B. Kruskal, Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, Addison–Wesley, 1983.
[21] S. Sippu, E. Soisalon-Soininen, Parsing Theory: Languages and Parsing, EATCS Monographs on Theoretical Computer Science, vol. 1, Springer-Verlag, 1988.
[22] W.F. Tichy, The string-to-string correction problem with block moves, ACM Trans. Comput. Syst. 2 (4) (1984) 309–321.
[23] M.S. Waterman, Efficient sequence alignment algorithms, J. Theor. Biol. 108 (1984) 333–337.
[24] D. Wood, Theory of Computation, Harper & Row, 1987.