ELSEVIER

Information Processing Letters 54 (1995) 85–92

Information Processing Letters

Approximately matching context-free languages

Gene Myers [1]

Department of Computer Science, University of Arizona, Tucson, AZ 85721, USA

Communicated by G. Andrews; received 14 July 1994; revised 23 December 1994

Abstract

Given a string w and a pattern p, approximate pattern matching merges traditional sequence comparison and pattern matching by asking for the minimum difference between w and a string exactly matched by p. We give an $O(PN^2(N + \log P))$ algorithm for approximately matching a string of length N and a context-free language specified by a grammar of size P. The algorithm generalizes the Cocke–Younger–Kasami algorithm for determining membership in a context-free language. We further sketch an $O(P^5N^8 8^P)$ algorithm for the problem where gap costs are concave and pose two open problems for such general comparison cost models.

Keywords: Design of algorithms; Pattern matching; Sequence comparison

1. Introduction

Approximate pattern matching has a long history [1,2,9–12] but most work has focused on simple patterns such as keywords and simple, unit-cost comparison models such as edit distance. This note generalizes the exact matching, dynamic programming "CYK-algorithm" of Cocke, Younger and Kasami [8,14] to efficient algorithms for context-free languages under comparison models permitting weighted alignment scores and gap costs of several varieties. The most relevant earlier work is by Aho and Peterson [1] and Lyons [10] who gave $O(P^2N^3)$ algorithms for approximate matching under the edit-distance model that are based on generalizations of Earley's algorithm [5]. Their results are only applicable to edit distance, taking advantage of the unit-cost scores in a way that does not generalize, and their algorithms are not as efficient as a function of grammar size.

An alignment between two strings $w = a_1a_2 \cdots a_N$ and $v = b_1b_2 \cdots b_M$ is a set of pairs of aligned symbols such that if one were to draw lines between aligned symbols they would not cross as in Fig. 1. Formally, alignment $T = \{(i_1, j_1), (i_2, j_2), \ldots, (i_L, j_L)\}$ aligns symbols a_{i_k} and b_{j_k} for all k, and satisfies the property that $i_k < i_{k+1}$ and $j_k < j_{k+1}$ for all $k < L$. The score of an alignment is determined by a function δ that assigns a real-valued score, $\delta(a, b)$, to each aligned/mismatched pair and a score $\delta(u)$ to each gap $u \in \Sigma^+$. The score of alignment T under δ is:

$$\sum_{k=1}^{L} \delta(a_{i_k}, b_{j_k}) + \sum_{k=0}^{L} \delta(a_{i_k+1}a_{i_k+2} \cdots a_{i_{k+1}-1}) + \sum_{k=0}^{L} \delta(b_{j_k+1}b_{j_k+2} \cdots b_{j_{k+1}-1}).$$

[1] Partially supported by NLM Grant LM-04960 and the Aspen Center for Physics. Email: gene@cs.arizona.edu.

SSDI 0020-0190(95)00007-0

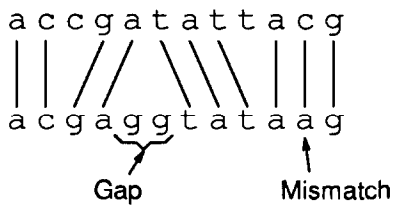

Fig. 1. An alignment.

where for simplicity it is assumed that $i_0 = j_0 = 0$, $i_{L+1} = N$, $j_{L+1} = M$, and $\delta(\varepsilon) = 0$. Usually $\delta(a, a) = 0$ but this need not be the case. Typically, gaps are attributed scores according to (1) their length, i.e. $\delta(c_1 c_2 \cdots c_h) = gap(h)$, or (2) as the sum over the symbols in the gap of scores attributed to leaving each symbol unaligned, i.e., $\delta(c_1 c_2 \cdots c_h) = \sum_{k=1}^{h} \sigma(c_k)$. This paper considers the latter, symbol-dependent model in its central algorithm and a number of versions of the length-dependent model in its latter sections.

When comparing strings w and v, we seek an alignment of minimal score and call the minimal score, $\delta(w, v)$, the difference between w and v. Generalizing, the difference $\delta(p, w)$ between a *pattern* p and a string w is $\min\{\delta(v, w)\colon v \in \mathcal{L}(p)\}$ where $\mathcal{L}(p)$ is the set of strings exactly matched by p, also called the *language accepted by* p.

In this note we focus on the basic approximate matching problem of computing the difference between a pattern and a string, analogous to the basic problem of determining membership in the language of a pattern. Algorithms for variations, such as determining a string v in $\mathcal{L}(p)$ realizing the difference and its alignment to w, follow directly from the algorithm for the basic problem. Our basic algorithm for context-free languages also solves the searching variation where one is searching a text t for all substrings whose difference from p is within some threshold.

Let $\mathcal{G} = \langle V, S, \Sigma, \Pi \rangle$ be a context-free grammar where V is the set of *non-terminals*, $S \in V$ is the *sentence symbol*, Σ is the *terminal alphabet*, and $\Pi \subset V \times (V \cup \Sigma)^*$ is the *production set*. Productions will be denoted $A \to \alpha$ where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$. We write $\alpha \Rightarrow \beta$ when the string β can be derived from α in one rewriting step, and $\alpha \Rightarrow^* \beta$ if α can derive β in zero or more steps. The language accepted by $\mathcal{G}$, $\mathcal{L}(\mathcal{G})$ is the set of all $w \in \Sigma^*$ such that $S \Rightarrow^* w$, i.e., all terminal strings derivable from S. We assume without loss of generality that $\mathcal{G}$ does not contain any useless productions [7, pp. 88–90].

We consider the size, P, of a grammar to be, $\sum_{A \to \alpha}(|\alpha| + 1)$, the sum of the lengths of all productions. For reasons of both simplicity and algorithmic complexity, we first transform this grammar into Chomsky Normal Form (CNF) [7, pp. 92–94] *save for the removal of unit productions*. This leaves us with an equivalent transformed grammar $\mathcal{G}' = \langle V', S, \Sigma, \Pi' \rangle$ whose size is still $O(P)$ but for which every production is of the form $A \to BC$, $A \to B$, or $A \to a$ where $A, B, C \in V$ and $a \in \Sigma$, save for the one production $S \to \varepsilon$ iff $\varepsilon \in \mathcal{L}(\mathcal{G})$. We call this form *pseudo-CNF* to emphasize that we have not removed unit productions (e.g., $A \to B$) from the grammar. These are normally removed as a last step in the CNF transformation, but we do not do so as their removal can create a grammar of size $O(P^2)$.

2. The central algorithm

Consider computing $\delta(\mathcal{G}, w)$ where $\mathcal{G}$ is a context-free grammar that is in pseudo-CNF form, $w = a_1 a_2 \cdots a_N$ is a string of length N, and δ is a weighted and symbol-dependent scoring scheme with non-negative gap costs. That is, $\delta(c_1 c_2 \cdots c_h) = \sum_{k=1}^{h} \sigma(c_k)$, and $\delta(a, b)$ and $\sigma(c) \geqslant 0$ are real numbers. The CYK-algorithm for exact matching computes the set of non-terminals that can derive every substring $w_{i\ldots j} = a_i a_{i+1} \cdots a_j$ of w. Instead, we compute for all $A \in V$ and $0 \leqslant i - 1 \leqslant j \leqslant N$, the score $C(A, i, j)$ of the best alignment between a non-empty string derivable from A and the substring $w_{i\ldots j}$ (where $w_{i\ldots i-1} = \varepsilon$). Formally:

$$C(A,i,j) = \min\{\delta(v, w_{i\ldots j}) \colon A \Rightarrow^* v \neq \varepsilon\}.$$

Theorem 1 gives a recurrence for computing these C-values.

Theorem 1. *For all* $A \in V$ *and* $0 \leqslant i-1 \leqslant j \leqslant N$:

$$\begin{aligned} C(A,i,j) = \min\{&\min_{A\to BC, k\in[i-1,j]} C(B,i,k) + C(C,k+1,j), \\ &\min_{A\to B} C(B,i,j), \\ &\min_{A\to a, k\in[i,j]} \delta(a,a_k) + G(i,j) - \sigma(a_k), \\ &\min_{A\to a} \sigma(a) + G(i,j)\}, \end{aligned} \tag{1}$$

where $G(i,j) \equiv g(j) - g(i-1)$ *and*

$$g(i) = \begin{cases} 0 & \text{if } i = 0, \\ g(i-1) + \sigma(a_i) & \text{if } i > 0. \end{cases}$$

Proof. First we show $C(A,i,j)$ is not greater than the right-hand side. Suppose for some $k \in [i-1,j]$ that $C(B,i,k) = \delta(t, w_{i\ldots k})$ where $B \Rightarrow^* t$, and $C(C,k+1,j) = \delta(u, w_{k+1\ldots j})$ where $C \Rightarrow^* u$. If $A \to BC \in \Pi$ then $A \Rightarrow BC \Rightarrow^* tu$, and so $C(A,i,j) \leqslant \delta(tu, w_{i\ldots j}) \leqslant \delta(t, w_{i\ldots k}) + \delta(u, w_{k+1\ldots j}) = C(B,i,k) + C(C,k+1,j)$. Similarly, if $A \to B \in \Pi$ then one can conclude $C(A,i,j) \leqslant C(B,i,j)$. Next, it is easy to see that $g(i) = \sum_{h\leqslant i} \sigma(a_h)$ and thus that $G(i,j) = \sum_{i\leqslant h\leqslant j} \sigma(a_h) = \delta(\varepsilon, w_{i\ldots j})$, the cost of leaving $w_{i\ldots j}$ unaligned. So if $A \to a \in \Pi$ for $a \in \Sigma$, then one can conclude $C(A,i,j) \leqslant \delta(a, w_{i\ldots j})$, but by the additivity of δ it follows that $\delta(a, w_{i\ldots j})$ is the minimum of either $\delta(a,\varepsilon) + \delta(\varepsilon, w_{i\ldots j}) = \sigma(a) + G(i,j)$ or $\min_k \delta(\varepsilon, w_{i\ldots k-1}) + \delta(a,a_k) + \delta(\varepsilon, w_{k+1\ldots j}) = \min_k \delta(a,a_k) + G(i,j) - \sigma(a_k)$. We do not need to consider the possible production $S \to \varepsilon$ as A must derive a non-empty string. Thus $C(A,i,j)$ is not greater than any term in the minimum of the right-hand side.

For the converse, suppose that $A \Rightarrow^* v \neq \varepsilon$ and $C(A,i,j) = \delta(v, w_{i\ldots j})$. Since there must be some first step in the derivation of v from A, one of the following must be true: (1) $A \Rightarrow BC \Rightarrow^* v$, (2) $A \Rightarrow B \Rightarrow^* v$, or (3) $A \Rightarrow a = v$. If (1) is true then there are strings t and u such that $v = tu$, $B \Rightarrow^* t$, and $C \Rightarrow^* u$. Moreover, there must be some $k^* \in [i-1,j]$ such that $\delta(v, w_{i\ldots j}) = \delta(t, w_{i\ldots k^*}) + \delta(u, w_{k^*+1\ldots j})$. It then follows that $C(A,i,j) = \delta(v, w_{i\ldots j}) = \delta(t, w_{i\ldots k^*}) + \delta(u, w_{k^*+1\ldots j}) \geqslant C(B,i,k^*) + C(C,k^*+1,j)$. Similarly, if (2) or (3) are true then $C(A,i,j) \geqslant C(B,i,j)$ or $C(A,i,j) \geqslant \delta(a, w_{i\ldots j})$, respectively. Thus, $C(A,i,j)$ must be greater or equal to at least one of the terms in the minimum of the right-hand side. □

The recurrence of Theorem 1 expresses $C(A,i,j)$ in terms of other C-values for which $j-i$ is smaller except when i and j remain unchanged in the unit-production minimum of Recurrence (1) and more subtly when $k = i-1$ or $k = j$ in the first minimum of Recurrence (1). Thus, $C(A,i,j)$ may potentially depend on itself since in a recursive grammar A can derive a string containing A. This is even more evident in the boundary case where $j = i-1$. First note that $C(A,i,i-1) \equiv E(A)$ for all values of i, where $E(A) = \min\{\delta(v,\varepsilon) \colon A \Rightarrow^* v \neq \varepsilon\}$. Thus, a non-asymptotic efficiency gain is possible in the algorithm to follow by computing $E(A)$ once instead of $N+1$ times. Specializing Recurrence (1) to $E(A)$ yields the recurrence:

$$E(A) = \min\left\{ \min_{A\to BC} E(B) + E(C), \min_{A\to B} E(B), \min_{A\to a} \sigma(a) \right\} \tag{2}$$

whose cyclic dependencies are evident. Such cyclic dependencies do not arise in the CYK algorithm, and their presence precludes a direct application of dynamic programming.

Using the dynamic programming paradigm, the algorithm of Fig. 2 below computes a table, $C[A,i,j]$, of the C-values in increasing order of $len = j - i + 1$. So when one is about to compute C-values for a given i

```
0.  Var C: array [V,1..N+1,0..N] of real
1.  For len ← 0 to N do
2.    For i ← 1 to N − len + 1 do
3.      { j ← i + len − 1
4.        For A ∈ V do
5.          C[A,i,j] ← known(A,i,j)
6.        H ← heap of V (ordered by C[?,i,j])
7.        While H ≠ ∅ do
8.          { A → extract_min(H)
9.          For B ∈ H and (B → AC ∈ Π or B → CA ∈ Π) do
10.           { C[B,i,j] ← min{C[B,i,j], C[A,i,j] + E(C)}, reheap(H,B) }
11.         For B ∈ H and B → A ∈ Π do
12.           { C[B,i,j] ← min{C[B,i,j], C[A,i,j]}, reheap(H,B) }
13.         }
14.     }
15.   Print "δ(G,w) is" · min{C[S,1,N], G(0,N) if S → ε}
```

Fig. 2. $O(PN^2(N+\log P))$ algorithm for computing $\delta(\mathcal{G},w)$.

and j in lines 4-12, one can assume that $C(B,g,h)$ has been correctly computed and stored in $C[B,g,h]$ for all B, g, and h such that $h-g<j-i$. So all the terms of Recurrence (1) in:

$$\begin{aligned} known(A,i,j) = \min\{&\min_{A\to BC,k\in[i,j-1]} C(B,i,k)+C(C,k+1,j),\\ &\min_{A\to a,k\in[i,j]} \delta(a,a_k)+G(i,j)-\sigma(a_k),\\ &\min_{A\to a} \sigma(a)+G(i,j)\} \end{aligned} \tag{3}$$

have been computed. For the remainder, $C(A,i,j)$ depends on $C(B,i,j)$ iff $A\to BC$, $A\to CB$, or $A\to B$. Consider the following weighted graph model of these dependencies. There is a vertex for each non-terminal in $\mathcal{G}$ and a special source vertex ϕ. There is an edge from ϕ to every vertex A whose weight is $known(A,i,j)$. Also there is an edge of weight 0 from B to A iff $A\to B$ and an edge from B to A with weight $E(C)$ iff $A\to BC$ or $A\to CB$. Because $C(A,i,j)=\min\{known(A,i,j),\ C(B,i,j)$ if $A\to B,\ C(B,i,j)+E(C)$ if $A\to BC$ or $A\to CB\}$, it follows by construction that the value of $C(A,i,j)$ is the weight of a shortest path from ϕ to A in this graph. Moreover, since σ is positive, $E(A)$ is positive for every non-terminal, and so every edge weight is positive. Thus Dijkstra's shortest path algorithm [4] may be applied directly as detailed in lines 4–12 of Fig. 2.

Theorem 2. *After* $O(PN^2(N+\log P))$ *time the algorithm of Fig.* 2 *concludes with* $C[A,i,j]=C(A,i,j)$ *for all* A, i, j. *Also* $\delta(\mathcal{G},w)=\min\{C[S,1,N],\ G(0,N)$ *if* $S\to\varepsilon\}$.

Proof. With regard to correctness, the preceding paragraph presents the algorithm of Fig. 2 as a dynamic programming algorithm centering on recurrence (1) with an application of Dijsktra's algorithm in the inner loop to properly handle the cyclicity induced by the grammar. Thus correctness of the C-values follows from the correctness of (1) and that of Dijkstra's algorithm. Finally, either the empty string is closest to w (if $\varepsilon\in\mathcal{L}(\mathcal{G})$) or some non-empty string is closest to w. By construction the former is $G(0,N)$ and the latter is $C[S,1,N]$. Thus line 15 correctly reports $\delta(\mathcal{G},w)$.

A total of $O(PN)$ time is spent for each execution of the for-loop of lines 4 and 5, assuming that $O(N)$ time is taken earlier to precompute a table of the values $g(i)$. The implementation of Dijkstra's algorithm in lines 6–14 takes $O(P\log P)$ time as the underlying graph described above has $O(P)$ edges. The result thus follows as $O(P(N+\log P))$ time is spent each time lines 3–14 are executed, and these lines are executed

$O(N^2)$ times. As a final note, observe that the best alignment between a non-empty string v and its derivation from S can be obtained with a traceback from $C[S,1,N]$ through the C-values in $O(P(M+N))$ time where M is the length of v. □

Theorem 2 requires that $E(A) \geqslant 0$ for all A, which is true whenever $\sigma(a) \geqslant 0$ for all $a \in \Sigma$. That is, in order to apply Dijkstra's algorithm, all that is required is that σ-values be non-negative, the alignment values $\delta(a,b)$ may be arbitrary. In applications in molecular biology, such a restriction on gap costs is almost always reasonable. Nonetheless, for arbitrary σ an algorithm is still possible: rather than using Dijkstra's algorithm we can use an $O(P^2)$ algorithm (e.g. [3]) for computing shortest paths over a graph with arbitrary edge weights. One subtlety: if such a subprocedure detects a negative cycle then the overall algorithm should halt and report a difference of $-\infty$. Thus we also have an $O(PN^2(N+P))$ algorithm for approximately matching a context-free language under arbitrary scoring scheme δ.

3. Affine gap costs

We now turn to sketching extensions to our basic result for the case where gap costs are length dependent. For affine gap costs [6], i.e., $gap(h) = r + sh$ for constants r and s, a simple extension gives an algorithm with the same complexity as that of the previous section. The key is to develop recurrences for:

$$C_{\triangleright}^{\triangleleft}(A,i,j) = \min\{\delta(x) + \delta(v, w_{i\ldots j}) + \delta(y) \colon A \Rightarrow^* xvy \neq \varepsilon \text{ and } |x| \triangleleft 0 \text{ and } |y| \triangleright 0\},$$

where $\triangleright, \triangleleft \in \{\text{“}=\text{”}, \text{“}\neq\text{”}\}$. The idea is to compute four different C-values modulated by $\triangleleft$ and $\triangleright$ that separate alignments with strings derivable from A (hereafter called A-strings) into four cases depending on whether the alignment begins or ends with a gap of symbols of the A-string. For example, $C_{=}^{\neq}(A,i,j)$ is the score of the best alignment between an A-string t with $w_{i\ldots j}$ that leaves the first symbol of t unaligned and its last symbol aligned.

Theorem 3. *For all $A \in V$, $0 \leqslant i-1 \leqslant j \leqslant N$, and $\triangleleft, \triangleright \in \{\text{“}=\text{”}, \text{“}\neq\text{”}\}$:*

$$\begin{aligned} C_{\triangleright}^{\triangleleft}(A,i,j) = \min\{ & \min_{A\to BC, k\in[i-1,j], \circ,\diamond\in\{=,\neq\}} \; C_{\circ}^{\triangleleft}(B,i,k) + C_{\triangleright}^{\diamond}(C,k+1,j) - (r \text{ if } \circ = \diamond = \text{“}\neq\text{”}), \\ & \min_{A\to B} \; C_{\triangleright}^{\triangleleft}(B,i,j), \\ & \textbf{if } \triangleleft = \triangleright = \text{“}=\text{”} \textbf{ then} \\ & \quad \min_{A\to a, k\in[i,j]} \; \delta(a,a_k) + (j-i)s + (r \textit{ if } k > i) + (r \textit{ if } k < j), \\ & \textbf{if } \triangleleft \neq \triangleright \textbf{ and } j \geqslant i \textbf{ and } A \to a \textbf{ then} \\ & \quad (j-i+2)s + 2r, \\ & \textbf{if } \triangleleft = \triangleright = \text{“}\neq\text{”} \textbf{ and } j < i \textbf{ and } A \to a \textbf{ then} \\ & \quad r + s\} \end{aligned}$$

Proof. The proof is a relatively direct extension of that for Theorem 1, so we make only a couple comments to assist the reader's intuition. In the minimum over productions $A \to BC$, r is subtracted when $\triangleleft = \triangleright = \text{“}\neq\text{”}$ because two gaps are being merged into one. On the other hand, one need not be concerned with end-gaps of $w_{i\ldots j}$ as the minimum is over all possible k. Finally, the basis minimums over productions $A \to a$ of Recurrence (1) have $G(i,j)$ replaced with $r + (j-i+1)s$, and $\sigma(a)$ with $r+s$, and are spread across the four choices of $\triangleleft$ and $\triangleright$, according to the underlying alignment. □

$gap(h) \geqslant 0$ for all h if and only if $r + s \geqslant 0$ and $s \geqslant 0$. In this event we have a scoring scheme with non-negative gap costs and a direct adaptation of the algorithm of Fig. 2 to the recurrence of Theorem 3 yields an $O(PN^2(N + \log P))$ algorithm for approximate matching of context-free languages with affine gap costs. When $gap(h)$ is negative for some h then we can develop an $O(PN^2(N + P))$ algorithm by an appeal to a more powerful shortest paths algorithm as noted earlier.

4. Concave gap costs

Concave gap costs [13] are characterized as having non-increasing forward differences, i.e., $\Delta gap(h) \geqslant \Delta gap(h+1)$ for all $h \geqslant 1$ where $\Delta gap(h) = gap(h+1) - gap(h)$. In this more general case we first have to be concerned with whether or not $gap(h)$ is non-decreasing. If it is not and $\mathcal{L}(\mathcal{G})$ is not finite (i.e. $\mathcal{G}$ is recursive), then the difference to w can be made arbitrarily negative by simply deriving arbitrarily large strings. So a first step is to screen this possibility and answer $-\infty$ if it arises. Hence below we assume $gap(h)$ is non-decreasing. During the treatment under this assumption, it will become clear that a corollary is an $O(PN^4 8^P)$ algorithm for the case when $\mathcal{L}(\mathcal{G})$ is finite.

Assuming concave, non-decreasing gap costs, the key is to develop recurrences for

$$C(A,i,j,l,r) = \min\{\delta(x) + \delta(v, w_{i\ldots j}) + \delta(y) \colon A \Rightarrow^* xvy \neq \varepsilon \text{ and } |x| = l \text{ and } |y| = r\},$$

where $l, r \in [0, NP2^P]$. In this case we maintain additional parameters l and r that constrain the exact size of the end-gaps that occur in an A-string's alignment with $w_{i\ldots j}$, and the C-value does not include the cost of these end-gaps. The lack of additivity in scoring gaps requires that the gap extensions modeled by aligning an A-string with the empty string be explicitly represented in the recurrence using the auxiliary quantity:

$$\mathcal{L}(t) = \{A \colon A \Rightarrow^* v \text{ and } |v| = t\}$$

which is the set of non-terminals that can derive strings of length $t \in [1, NP2^P]$. This has the compensatory effect of removing the need for the boundary case $j = i - 1$, since it is now explicitly represented in the recurrence of Theorem 4.

Theorem 4. *For all $A \in V$, $0 \leqslant i \leqslant j \leqslant N$, and $l, r \geqslant 0$:*

$$\begin{aligned}
&C(A,i,j,l,r)\\
&\quad = \min\{\min_{A\to BC, i\leqslant k<t\leqslant j, m,n\geqslant 0} \; C(B,i,k,l,m) + C(C,t,j,n,r) + gap(t-k-1) + gap(m+n),\\
&\qquad \min_{A\to BC, t\in[1,r], C\in\mathcal{L}(t)} \; C(B,i,j,l,r-t),\\
&\qquad \min_{A\to BC, t\in[1,l], B\in\mathcal{L}(t)} \; C(C,i,j,l-t,r),\\
&\qquad \min_{A\to B} \; C(B,i,j,l,r),\\
&\qquad \textbf{if } l = r = 0 \textbf{ then}\\
&\qquad\quad \min_{A\to a, k\in[i,j]} \; \delta(a,a_k) + gap(k-i) + gap(j-k),\\
&\qquad \textbf{if } l + r = 1 \textbf{ and } A \to a \textbf{ then}\\
&\qquad\quad gap(j-i+1)\}
\end{aligned}$$

and

$$\mathcal{L}(t) = \bigcup_{k=1}^{t-1} \{A \colon A \to BC \in \mathcal{L}(k) \times \mathcal{L}(t-k)\} \cup \{A \colon A \to B \in \mathcal{L}(t)\} \cup \{A \colon A \to a \text{ and } t = 1\}.$$

Proof. The correctness of the recurrence is established using a case-based argument on the string derivable from A as in Theorem 1. □

The one essential point that we will prove by an appeal to the pumping lemma is that it suffices to compute the recurrence only for end gaps up to a maximum size of $NP2^P$, i.e. the variables l, r, m, and n of the recurrence can be restricted to range over the interval $[0, NP2^P]$ without loss of generality. To do so it suffices to show that $Best(A,i,j) = \min_{l,r\in[0,NP2^P]} C(A,i,j,l,r)$ where $Best(A,i,j) = \min\{\delta(v, w_{i\ldots j})\colon A \Rightarrow^* xvy$ for some x and $y\}$ is the best alignment between a substring of an A-string and $w_{i\ldots j}$. Suppose that $Best(A,i,j) = \delta(u, w_{i\ldots j})$ and that the subsequence u^* is the sequence of symbols in u aligned with those of $w_{i\ldots j}$ in the optimal alignment. In the derivation tree of u from A, let a *junction vertex* be an interior vertex with two children (in CNF outdegree is one or two) both of which derive at least one symbol in u^*. Any subpath between two junction vertices with more than $|V|$ vertices must have a repeated non-terminal label. In such a case one could produce a derivation from A with shorter gaps between symbols in u^* by cutting out the section between the repeated labels, a contradiction to optimality. Thus in the derivation of u from A, all subpaths between consecutive junction vertices must have $|V|$-or-less vertices, and each *off-child* of a vertex on a subpath which itself is not on the subpath must produce a shortest possible string derivable from its non-terminal label. But then from the facts that $\max_{A\in V} \min_{A\Rightarrow^* v} |v| \leqslant 2^{P-1}$ and $|u^*| \leqslant |w_{i\ldots j}| \leqslant N$, it follows that the largest gap in the alignment with u is not greater than the number of leftmost/rightmost subpaths between junction vertices times their length times the length of the minimum strings derived by an off-child, i.e. $(N+1)(P-1)2^{P-1} \leqslant NP2^P$.

Given that the recurrence of Theorem 4 need only be computed in the range of indices proved above, it follows that one need only compute these $O(|V|P^2N^4 4^P)$ entries. The cost of handling the cyclic dependencies introduced by the unit production terms is dominated by the $O(P_A P^2 N^4 4^P)$ time needed to computed the first minimum of the recurrence, where P_A is the number of productions with A as their left-hand side. Thus, we can compute the difference between a context-free language and a string under a concave, non-decreasing gap-penalty scoring scheme in worst-case time $O(P^5N^8 8^P)$. Note that when $\mathcal{L}(\mathcal{G})$ is finite, its longest string is of length not greater than 2^{P-1}. Thus, in this case, the end-gap indices need only be varied over the range $[0, 2^{P-1}]$ and an $O(PN^4 8^P)$ algorithm ensues regardless of whether $gap(h)$ is non-decreasing or not.

5. Open problems

The immediate question is whether one can do better than above since the powers are very high and the dependence on grammar size is exponential. This appears very difficult. Finally, if gap costs can be an arbitrary function of length, we know of no result and ponder whether the problem is computable by any Turing machine.

References

[1] A.V. Aho and T.G. Peterson, A minimum distance error-correcting parser for context-free languages, *SIAM J. Comput.* **1** (4) (1972) 305–312.

[2] T. Akutsu, Approximate string matching with don't care symbols, in: *Proc. 5th Combinatorial Pattern Matching Conf.*, Asilomar, CA, Lecture Notes in Computer Science **807** (Springer, New York, 1994) 240–249.

[3] R.E. Bellman, On a routing problem, *Quant. Appl. Math* **16** (1) (1958) 87–90.

[4] E.W. Dijkstra, A note on two problems in connexion with graphs, *Numer. Math.* **1** (1959) 269–271.

[5] J. Earley, An efficient context-free parsing algorithm, *Comm. ACM* **13** (1970) 94–102.

[6] W. Fitch and T. Smith, Optimal sequence alignments, *Proc. Nat. Acad. Sci. USA* **80** (1983) 1382–1386.

[7] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computations* (Addison-Wesley, Reading, MA, 1979).

[8] T. Kasami, An efficient recognition and syntax analysis algorithm for context-free languages, AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA, 1963.

[9] J.R. Knight and E.W. Myers, Approximate regular expression pattern matching with concave gap costs, in: *Proc. 3rd Combinatorial Pattern Matching Conf.*, Tucson, AZ, Lecture Notes in Computer Science **64** (Springer, New York, 1992) 67–76 (to appear in *Algorithmica*).
[10] G. Lyon, Syntax-directed least-errors analysis for context-free languages: A practical approach, *Comm. ACM* **17** (1) (1974) 3–14.
[11] E.W. Myers and W. Miller, Approximate matching of regular expressions, *Bull. Math. Biol.* **51** (1) (1989) 5–37.
[12] R.A. Wagner and J.I. Seiferas, Correcting counter automaton recognizable languages, *SIAM J. Comput.* **7** (3) (1978) 357–375.
[13] M.S. Waterman, General methods of sequence comparison, *Bull. Math. Biol.* **46** (1984) 473–501.
[14] D.H. Younger, Recognition and parsing of context-free languages in time n^3, *Inform. and Control* **10** (2) (1967) 189–208.