In Proc. of 8th Symposium on Symposium on
Combinatorial Pattern Matching, Aarhus, Denmark,
June/July 1997, Lecture Notes in Computer Science 1264

Estimating the Probability of Approximate Matches

Stefan Kurtz[1⋆] and Gene Myers[2⋆⋆]

[1] Technische Fakultät, Universität Bielefeld, Postfach 100 131, D-33501 Bielefeld, Germany, E-mail: kurtz@techfak.uni-bielefeld.de
[2] Department of Computer Science, The University of Arizona, Tucson, Arizona 85721, USA, E-mail: gene@cs.arizona.edu

1 Introduction

While considerable effort and some progress has been made on developing an analytic formula for the probability of an approximate match, such work has not achieved fruition [4, 6, 2, 1]. Therefore, we consider here the development of an unbiased estimation procedure for determining said probability given a specific string $P \in \Sigma^*$ and a specific cost function δ for weighting edit operations. Problems of this type are of general interest, see for example a recent paper [5] giving an unbiased estimator for counting the words of a fixed length in a regular language. We were further motivated by a particular application arising in the pattern matching system *Anrep* designed by us for use in genomic sequence analysis [8, 11]. *Anrep* accomplishes a search for a complex pattern by backtracking over subprocedures that find approximate matches. The subpatterns are searched in an order that attempts to minimize the expected running time of the search. Determining this optimal backtrack order requires a reasonably accurate estimate of the probability with which one will find an approximate match to each subpattern. Given that the probabilities involved are frequently 10^{-6} or less, the simple expedient of measuring match frequency over a random text of several thousand characters has been less than satisfactory. The unbiased estimator herein is shown to give good results in a matter of a thousand samples even for small probability patterns. Thus it is expected to improve the performance of *Anrep* and may have utility in estimating the significance of similarity searches.

Proceeding formally, suppose we are given

- a pattern string $P = p_1 p_2 \ldots p_m \in \Sigma^*$,
- an integer cost function δ, and
- a match threshold $k \in \mathbb{N}_0$.

⋆ Research done while visiting the University of Arizona, partially supported by NLM grant LM-4960.
⋆⋆ Partially supported by NLM grant LM-4960.

Let $A_k(P,T)$ denote the event that P can be aligned to a *prefix* of text T with cost k or less. We seek $Pr\left[A_k(P,T)\right]$ under the distributional assumption that T is generated by independent, uniform Bernoulli trials over the alphabet Σ. Our results depend on δ satisfying the following two conditions:

$$\delta(a \to \varepsilon) + \delta(\varepsilon \to b) \geq \delta(a \to b) \geq 0 \tag{1}$$

$$\delta(\varepsilon \to b), \delta(a \to \varepsilon) > 0 \tag{2}$$

where $a \to \varepsilon$ denotes the deletion of the character a, $a \to b$ denotes the replacement of the character a by the character b, and $\varepsilon \to b$ denotes the insertion of the character b. These conditions are generally met by the scoring schemes required in most applications, and if not, one can usually transform the scoring scheme to one satisfying them. Note that δ may otherwise be any cost function over the integers.

Let the condensed k-neighborhood of P, $CN_k(P)$, be the set of all strings approximately matching P within threshold k, less those that have another such string as a prefix. Because the occurrence of each string in $CN_k(P)$ as a prefix of T is an independent event, it immediately follows that

$$Pr\left[A_k(P,T)\right] = \sum_{v \in CN_k(P)} \frac{1}{|\Sigma|^{|v|}}$$

Unfortunately, the size of $CN_k(P)$ grows exponentially with the threshold value, quickly rendering direct computation by enumerating $CN_k(P)$ hopelessly inefficient. Thus we turn to developing a Monte-Carlo algorithm that estimates $Pr\left[A_k(P,T)\right]$ by sampling a subset of $CN_k(P)$. The difficulty immediately encountered is that a direct procedure for sampling $CN_k(P)$ appears formidable if not impossible. So instead we turn to sampling a surrogate space, $S_k(P)$, of edit scripts of cost k or less that convert P into strings it approximately matches. The set $S_k(P)$ can be any set of edit scripts provided that it is efficiently enumerable and *complete* in that for every $v \in CN_k(P)$, there is at least one edit script $s \in S_k(P)$ that when applied to P generates $s(P) = v$.

The difficulty to be overcome in sampling the surrogate space $S_k(P)$ is that there is not a one-to-one correspondence between its edit scripts and the strings in $CN_k(P)$. Some strings in $CN_k(P)$ may be generated by several distinct edit scripts in $S_k(P)$ and some edit scripts in $S_k(P)$ may not even generate strings in $CN_k(P)$. This bias is removed by considering the random variable $X : S_k(P) \to \mathbb{R}_0^+$ defined by:

$$X(s) = \begin{cases} \frac{1}{|\Sigma|^{|v|}\, g(v)} & \text{if } v \in CN_k(P) \\ 0 & \text{otherwise} \end{cases}$$

where $v = s(P)$, and $g(v)$, called the cluster size of v, is the number of edit scripts in $S_k(P)$ that generate the string v. That X removes the bias is the claim of our central lemma:

Lemma 1. *Suppose $S_k(P)$ is uniformly distributed. Then the expected value $E[X]$ of X is $Pr\left[A_k(P,T)\right] / |S_k(P)|$.*

Proof.

$$\begin{aligned}
E[X] &= \sum_{s \in S_k(P)} \frac{X(s)}{|S_k(P)|} \\
&= \frac{1}{|S_k(P)|} \cdot \sum_{s \in S_k(P)} \left\{ \frac{1}{|\Sigma|^{|v|} g(v)} \mid s(P) \in CN_k(P), v = s(P) \right\} \\
&= \frac{1}{|S_k(P)|} \cdot \sum_{v \in CN_k(P)} \left\{ \frac{1}{|\Sigma|^{|v|} g(v)} \mid s \in S_k(P), s(P) = v \right\} \\
&= \frac{1}{|S_k(P)|} \cdot \sum_{v \in CN_k(P)} \frac{g(v)}{|\Sigma|^{|v|} g(v)} \\
&= \frac{1}{|S_k(P)|} \cdot \sum_{v \in CN_k(P)} \frac{1}{|\Sigma|^{|v|}} \\
&= \frac{Pr[A_k(P,T)]}{|S_k(P)|}
\end{aligned}$$

Our unbiased estimation procedure thus consists of

- selecting a suitably large set of samples from $S_k(P)$ with uniform probability,
- computing the average of $X(s)$ over the samples s, and
- multiplying the average by $|S_k(P)|$.

While the procedure, in outline, is quite simple, the remainder of this paper addresses how to define $S_k(P)$ and how to solve the algorithmic sub-problems involved in an efficient realization with respect to this definition. Section 2 introduces as our choice for $S_k(P)$ the set of what we call the *condensed*, *canonical edit scripts*. Our choice attempts to keep small, both (i) the number of edit scripts for which $X(s) = 0$, and (ii) the size of $g(v)$. Doing so improves the convergence of the estimator as it places $S_k(P)$ and $CN_k(P)$ in closer correspondence. The remaining sections present dynamic programming algorithms for the following subtasks:

Section 3: Determining the size of $S_k(P)$ in $O(|\Sigma|mk)$ time.
Section 4: Selecting an edit script from $S_k(P)$ with uniform probability in $O(|\Sigma|(m+k))$ time.
Section 5: Deciding if $s \in S_k(P)$ generates a string $s(P) \in CN_k(P)$ in $O(m \cdot \min(m,k))$ time.
Section 6: Computing the cluster size $g(v)$ in $O(\Delta \cdot m \cdot \min(m,k))$ time, where $\Delta \in [1, k+1]$ is depending on δ and P.

Altogether, our Monte-Carlo algorithm achieves a running time of $O(|\Sigma|mk + t(|\Sigma|m + |\Sigma|k + \Delta \cdot m \cdot \min(m,k)))$ and requires $O(mk)$ space where t is the number of samples collected. If δ is the unit cost function that scores all mismatches, insertions, and deletions as 1, the running time further improves to simply $O(tmk)$. The paper concludes with Section 7 that presents empirical results demonstrating the accuracy of the estimates, the rate of convergence of the sampling process, and the speed of the procedure.

In the context of scanning a text, our estimator above gives the probability that a match *begins* at a specific position in the text. However, these events are not independent. One match may significantly condition a match at the next

position. To estimate the number of such "clumps", one needs to compute the probability that a match begins at a position and there is no overlapping match to the left of it. Addressing this issue is beyond the scope of this paper. The current result is still of value as it overestimates this probability and in the context of our *Anrep* application these probabilities are small enough that the clump size is almost always one, so the overestimation is slight.

We close the introduction by noting that our treatment is extensible to patterns that are network expressions [11] (regular expression without Kleene closure), and to models of the text where characters are generated by a weighted Bernoulli process. We do not directly treat these extensions in this paper, as they complicate the treatment and obscure the basic ideas.

2 Condensed, Canonical Edit Scripts

Approximate matches are typically characterized as an alignment or trace between P and a string that it matches. In our context we must turn to the equivalent operational view of an *edit script* that transforms P into the string it matches as originally introduced, for example, in the seminal work of Wagner and Fischer [13]. There are three kinds of edit operations: $a \to \varepsilon$ denotes the *deletion* of the character a, $a \to b$ denotes the *replacement* of the character a by the character b, and $\varepsilon \to b$ denotes the *insertion* of the character b. An edit script for $P = p_1 p_2 \ldots p_m$ is a list $s = [\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \ldots, \alpha_r \to \beta_r]$ of edit operations for which $r \geq m$ and $P = \alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_r$ when one interprets ε as denoting the empty string. P is viewed as the *source* string and the application of the edit script s results in the *target* string $s(P) = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_r$. We further say that s *generates* the string $s(P)$. The correspondence between edit scripts and alignments is immediate given the observation that each edit operation corresponds to an alignment column.

The underlying *cost function* δ assigns a non-negative integer cost, $\delta(\alpha \to \beta)$, to each edit operation $\alpha \to \beta$. Recall that in addition to integrality, we are assuming that (1) and (2), given in the introduction, hold. The *cost* $\delta(s)$ of an edit script $s = [\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \ldots, \alpha_r \to \beta_r]$ is the sum of the costs of its edit operations, i.e., $\delta(s) = \sum_{i=1}^{r} \delta(\alpha_i \to \beta_i)$. We say that s is a (P, l) edit script if s is an edit script for P of cost l. The *edit distance* between P and $v \in \Sigma^*$, denoted by $\delta(P, v)$, is the minimal cost over all edit scripts for P that generate v. An edit script s for P is *optimal* if $\delta(s) = \delta(P, s(P))$.

Given P and $k \geq 0$, the *k-neighborhood* of P is the set of all strings matching P with cost k or less, i.e., $N_k(P) = \{v \in \Sigma^* \mid \delta(P, v) \leq k\}$. In order to remove dependent events, we restrict our attention to the *condensed k-neighborhood*, $CN_k(P)$, which is defined by

$$CN_k(P) = \{v \in N_k(P) \mid \forall \text{ proper prefixes } v' \text{ of } v, v' \notin N_k(P)\}$$

Note that $\varepsilon \in N_k(P)$ implies $Pr\,[A_k(P, T)] = 1$. Hence we do not need to estimate $Pr\,[A_k(P, T)]$ whenever k is greater or equal to the cost of deleting every

character in P. This implies that $k = O(m)$ in the cases of interest, assuming δ is a constant.

Recall that we will effectively be sampling $CN_k(P)$ by sampling the surrogate space $S_k(P)$ and applying the random variable X to each sample. We could simply choose to let $S_k(P)$ be the set of *all* (P, l) edit scripts for $l \leq k$. However, the convergence rate of the estimator improves, the tighter the correspondence between the set of sampled edit scripts and the condensed k-neighborhood. We develop a much smaller but still complete set of edit scripts in a progression of three observations below.

First, we remove easily detectable non-optimal or redundant edit scripts. An edit script is called *canonical* if it does not contain a sublist of the form: $[a \to \varepsilon, \varepsilon \to b]$, $[\varepsilon \to b, a \to \varepsilon]$, $[a \to \varepsilon, a \to b]$, or $[\varepsilon \to b, a \to b]$ for some $a, b \in \Sigma$. Such sublists are called *forbidden*.[1] Intuitively, canonical edit scripts can be characterized by the following properties:

- A replacement is preferred over a deletion/insertion or insertion/deletion combination.
- If the source string contains a substring of the same character, then only the rightmost instances in the substring are deleted (if any).
- If the edit script generates a substring of the same character in the target string, then only the rightmost instances are generated by an insertion (if any).

Let $C_l(P)$ be the set of canonical (P, l) edit scripts. Provided that δ satisfies (1), limiting our attention to canonical edit scripts is conservative as every non-canonical edit script s can be transformed into a canonical edit script s' such that $s(P) = s'(P)$ and $\delta(s') \leq \delta(s)$ holds. Thus we have $\bigcup_{l \leq k} \bigcup_{s \in C_l(P)} \{s(P)\} = N_k(P)$.

The second observation is that any edit script ending with an insertion, generates a shorter word at lesser cost if the final insertion is dropped from the edit script. Thus, any such edit script cannot generate a string in $CN_k(P)$ as it does not contain strings that have proper prefixes matching P at lesser cost. Let a *condensed, canonical edit script* be a canonical edit script that does not end with an insertion. Let $\mathcal{CC}_l(P)$ be the set of condensed, canonical (P, l) edit scripts. We may restrict our attention to $\bigcup_{l \leq k} \mathcal{CC}_l(P)$ as it is complete w.r.t. $CN_k(P)$, i.e., $\bigcup_{l \leq k} \bigcup_{s \in \mathcal{CC}_l(P)} \{s(P)\} \supseteq CN_k(P)$.

Finally, observe that while the edit distance between a string $v \in CN_k(P)$ and P may be less than k, there must be a lower bound as one could replace the last non-deletion operation in an edit script generating v optimally with a

[1] One could generalize the idea of a canonical edit script to those that do not contain forbidden sublists of length 3, or 4, etc. The marginal rate at which these higher-order schemes eliminate redundant edit scripts diminishes rapidly, while the cost of sampling them grows exponentially in time. Thus we present only a second order scheme here, both because it is most practical and also for the sake of keeping the exposition simple. However, it is worth noting that if one goes to an m-order scheme, then $CN_k(P)$ and canonical edit scripts correspond one-to-one.

deletion operation for some bounded increase in cost. Indeed, carrying this train of thinking to its furthest degree, one arrives at the following lemma:

Lemma 2. *For any $v \in CN_k(P)$, $\delta(P, v) > k - \Delta$ where*

$$\Delta = \max\Big\{\delta(p_i \to \varepsilon) - \delta(p_i \to b) \mid i \in [1, m], b \in \Sigma, \sum_{r=i+1}^{m} \delta(p_i \to \varepsilon) \leq k - \delta(p_i \to b)\Big\}$$

Proof. *Let $v \in CN_k(P)$. We have $v = wb$ for some $w \in \Sigma^*$ and some $b \in \Sigma$. Let s be an optimal edit script for P generating v. Then $s \neq [\,]$, $\delta(s) = \delta(P, v) \leq k$, and the last edit operation in s is not an insertion. It is easy to show that there is an $i \in [1, m]$ and $b \in \Sigma$ such that $s = s' \cdot [p_i \to b] \cdot s''$ where s' is an edit script for $p_1 p_2 \ldots p_{i-1}$ generating w, and s'' is a maximal suffix of s consisting of deletions only. We have $\delta(s'') + \delta(p_i \to b) \leq \delta(s) \leq k$, and therefore $\sum_{r=i+1}^{m} \delta(p_r \to \varepsilon) = \delta(s'') \leq k - \delta(p_i \to b)$, which implies $\Delta \geq \delta(p_i \to \varepsilon) - \delta(p_i \to b)$. Since $s' \cdot [p_i \to \varepsilon] \cdot s''$ is an edit script for P generating w, we get $\delta(s') + \delta(p_i \to \varepsilon) + \delta(s'') \geq \delta(P, w)$. Hence*

$$\begin{aligned} \delta(P, v) &= \delta(s') + \delta(p_i \to b) + \delta(s'') \\ &\geq \delta(P, w) - \delta(p_i \to \varepsilon) + \delta(p_i \to b) \\ &> k - (\delta(p_i \to \varepsilon) - \delta(p_i \to b)) \\ &\geq k - \Delta \end{aligned}$$

To conclude, let

$$\mathcal{CC}(P) = \bigcup_{l=\Phi}^{k} \mathcal{CC}_l(P)$$

where $\Phi = k - \Delta + 1$. This set of edit scripts is our choice in this paper for $S_k(P)$. Note that $\mathcal{CC}(P)$ is an appropriate choice for the sampling surrogate as it is complete w.r.t. $CN_k(P)$, i.e., $\bigcup_{s \in \mathcal{CC}(P)} \{s(P)\} \supseteq CN_k(P)$.

3 Counting Edit Scripts

We first consider how to enumerate canonical edit scripts. To do so, we split the set of all canonical edit scripts into three classes according to the last edit operation in each edit script. This gives us the control we need to avoid enumerating edit scripts containing a forbidden sublist.

Definition 1. *For each $i \in [0, m]$, each $l \in [0, k]$, and each $b \in \Sigma$ define:*

$$\begin{aligned} D(l, i) &= \{s \in C_l(p_1 p_2 \ldots p_i) \mid s \text{ ends with a deletion}\} \\ R(l, i) &= \{s \in C_l(p_1 p_2 \ldots p_i) \mid s = [\,] \text{ or } s \text{ ends with a replacement}\} \\ I_b(l, i) &= \{s \in C_l(p_1 p_2 \ldots p_i) \mid s \text{ ends with the insertion } \varepsilon \to b\} \\ I(l, i) &= \{s \in C_l(p_1 p_2 \ldots p_i) \mid s \text{ ends with an insertion}\} \end{aligned}$$

If $l < 0$, we define $D(l, i) = R(l, i) = I_b(l, i) = I(l, i) = \emptyset$.

From the definitions it follows that $\mathcal{CC}(P) = \bigcup_{l=\Phi}^{k} D(l,m) \cup R(l,m)$. It remains to develop recurrences describing each class of edit scripts. Note that the class of edit scripts ending with an insertion are further decomposed according to the character b inserted. This is required so that, in the recurrences below, we can avoid composing edit scripts that follow the insertion of b with a substitution of b (see the equation for $R(l,i)$ below).

Lemma 3. *For $i, l \geq 0$, the following recurrences hold:*

$$\begin{aligned}
D(l,i) &= (D(l', i-1) \cup R(l', i-1)) \cdot [p_i \rightarrow \varepsilon] \text{ where } l' = l - \delta(p_i \rightarrow \varepsilon) \\
R(l,i) &= \bigcup_{b \in \Sigma} ((D(l^b, i-1) \text{ if } i > 1 \text{ and } p_{i-1} \neq p_i) \cup \\
&\qquad (I(l^b, i-1) - I_b(l^b, i-1)) \cup \\
&\qquad R(l^b, i-1)) \cdot [p_i \rightarrow b] \text{ where } l^b = l - \delta(p_i \rightarrow b) \\
I_b(l,i) &= (I(l', i) \cup R(l', i)) \cdot [\varepsilon \rightarrow b] \text{ where } l' = l - \delta(\varepsilon \rightarrow b) \\
I(l,i) &= \bigcup_{b \in \Sigma} I_b(l,i).
\end{aligned}$$

subject to the boundary conditions: $D(l,0) = \emptyset$ and $R(l,0) = (\text{if } l = 0 \text{ then } \{[\,]\} \text{ else } \emptyset)$. The notation $X \cdot a$ where X is a set of edit scripts and a is an edit operation, denotes the set $\{x \cdot a \mid x \in X\}$. The symbol "$\cdot$" denotes the concatenation of edit scripts.

With the recurrences above it is now a simple exercise to reformulate them to *count* the number of edit scripts, as opposed to *enumerating* them. We compute three $(k+1) \times (m+1)$-tables: $ND(l,i) = |D(l,i)|$, $NR(l,i) = |R(l,i)|$, and $NI(l,i) = |I(l,i)|$ for $l \in [0,k]$ and $i \in [0,m]$. Note that we make the small optimization of not *storing* the number of edit scripts in $I_b(l,i)$ for each $b \in \Sigma$, as it requires only constant time to compute these numbers on demand given the other entries. This saves us a factor of $|\Sigma|$ space without any penalty in time. We now have

$$|\mathcal{CC}(P)| = \sum_{l=\Phi}^{k} ND(l,m) + NR(l,m)$$

Thus it follows:

Lemma 4. *$|\mathcal{CC}(P)|$ can be evaluated in $O(|\Sigma|mk)$ time and $O(mk)$ space. This further improves to $O(mk)$ time when δ is the unit cost function.*

4 Uniformly Sampling Edit Scripts

We begin by noting that each edit script in $\mathcal{CC}(P)$ can be obtained by tracing back through the recurrences given in the previous section, thereby producing an edit script from *right to left*. In order to sample $\mathcal{CC}(P)$ uniformly, one must select a traceback branch with probability proportional to the number of possibilities the chosen branch can generate. That is, if there are r branch possibilities and

branch i leads to C_i possible finishes, then branch i should be chosen with probability $C_i / \sum_{l=1}^{r} C_l$.

As a concrete example, suppose that we have thus far chosen a suffix s for which $D(l,i) \cdot s \subseteq \mathcal{CC}(P)$. We wish to uniformly sample a finishing prefix from $D(l,i)$. Following the recurrence for $D(l,i)$, we prepend $[p_i \rightarrow \varepsilon]$ to s and decide to either complete the result with an edit script from $D(l', i-1)$ with probability $ND(l', i-1)/ND(l,i)$, or with an edit script from $R(l', i-1)$ with probability $NR(l', i-1)/ND(l,i)$ where $l' = l - \delta(p_i \rightarrow \varepsilon)$. As one further example, the sampling process starts with a decision to either generate the edit script from $D(l,m)$ with probability $ND(l,m)/|\mathcal{CC}(P)|$, or from $R(l,m)$ with probability $NR(l,m)/|\mathcal{CC}(P)|$ where $l \in [\Phi, k]$.

Recall that insertions have cost 1 or more (see (2)). Thus the longest possible edit script in $\mathcal{CC}(P)$ is of length not greater than $m+k$. Moreover, from the structure of the recurrences above it is easy to see that at each traceback point there are never more than $O(|\Sigma|)$ branches. Thus it follows:

Lemma 5. *Given the dynamic programming tables for counting edit scripts described in Section 3, an edit script $s \in \mathcal{CC}(P)$ can be selected with uniform probability in time $O(|\Sigma|(m+k))$. This further improves to $O(m+k)$ time when δ is the unit cost function.*

5 Deciding Condensed Neighborhood Membership

Suppose we have selected s from $\mathcal{CC}(P)$. We next wish to know if $v = s(P)$ is in $\mathcal{CN}_k(P)$. To do so, consider performing a standard sequence comparison (cf. [13]) between P and v. That is, consider computing the $(m+1) \times (n+1)$ table $E(i,j) = \delta(p_1 p_2 \ldots p_i, v_1 v_2 \ldots v_j)$ where $n = |v|$. Observe that $E(m,n) = \delta(P,v) = \delta(P, s(P)) \leq \delta(s) \leq k$, confirming that v is in $N_k(P)$. If v is in $\mathcal{CN}_k(P)$, then it must further be true that no prefix of v is in the k-neighborhood. That is, for all $j \in [0, n-1]$, $\delta(P, v_1 v_2 \ldots v_j) = E(m,j) > k$ must hold. Checking this condition requires just $O(n)$ additional time beyond the $O(mn)$ time spent computing E.

Whenever $k < m$, the time complexity can be further improved by observing that we need only consider the band of E of width k about the main diagonal. This follows because all entries outside this band must be greater than k as more than k insertions or deletions are required to edit the associated prefix of P into that of v. The computation of E restricted to this band consumes only $O(mk)$ time. Furthermore, only $O(k)$ space is required as each row can be computed from the preceding row. See [12, 9] for more details. Recalling that $n \leq m + k \in O(m)$, we obtain the following result:

Lemma 6. *$v \in \mathcal{CN}_k(P)$ can be decided in $O(m \cdot \min(k,m))$ time and in $O(\min(k,m))$ space.*

6 Computing Cluster Sizes

To this point an edit script s has been chosen uniformly from $\mathcal{CC}(P)$ and we have determined that $v = s(P)$ is in $\mathcal{CN}_k(P)$. It remains to compute the cluster size $g(v)$, i.e., the number of edit scripts in $\mathcal{CC}(P)$ that generate v. As in Section 3, we first consider how to construct the set of canonical edit scripts for P generating v. Throughout this section, let $r = k - E(m,n)$.

Definition 2. *For each $i \in [0,m]$, $j \in [0,n]$, and $l \in [E(i,j), E(i,j)+r]$ let $C(l,i,j)$ be the set of all canonical $(p_1p_2\ldots p_i, l)$ edit scripts generating $v_1v_2\ldots v_j$. Define*

$$\begin{aligned} D(l,i,j) &= \{s \in C(l,i,j) \mid s \text{ ends with a deletion}\} \\ R(l,i,j) &= \{s \in C(l,i,j) \mid s = [\,] \text{ or } s \text{ ends with a replacement}\} \\ I(l,i,j) &= \{s \in C(l,i,j) \mid s \text{ ends with an insertion}\} \end{aligned}$$

For $l < E(i,j)$, we define $D(l,i,j) = R(l,i,j) = I(l,i,j) = \emptyset$.

If we compare Definition 2 with Definition 1, we recognize an additional argument j, accounting for the fact that we want the edit scripts generating a *fixed* string v.

Lemma 7. *For $l \in [E(i,j), E(i,j)+r]$, the following recurrences hold:*

$$\begin{aligned} D(l,i,j) &= (D(l',i-1,j) \cup R(l',i-1,j)) \cdot [p_i \to \varepsilon] \text{ where } l' = l - \delta(p_i \to \varepsilon) \\ R(l,i,j) &= ((D(l',i-1,j-1) \quad \text{if } i > 1 \text{ and } p_{i-1} \neq p_i) \cup \\ &\qquad (I(l',i-1,j-1) \quad \text{if } j > 1 \text{ and } v_{j-1} \neq v_j) \cup \\ &\qquad R(l',i-1,j-1)) \cdot [p_i \to v_j] \text{ where } l' = l - \delta(p_i \to v_j) \\ I(l,i,j) &= (I(l',i,j-1) \cup R(l',i,j-1)) \cdot [\varepsilon \to v_j] \text{ where } l' = l - \delta(\varepsilon \to v_j) \end{aligned}$$

subject to the boundary conditions:

$$\begin{aligned} D(l,0,j) &= \emptyset \\ R(l,0,j) &= \text{if } l = 0 \text{ and } j = 0 \text{ then } \{[\,]\} \text{ else } \emptyset \\ R(l,i,0) &= \text{if } l = 0 \text{ and } i = 0 \text{ then } \{[\,]\} \text{ else } \emptyset \\ I(l,i,0) &= \emptyset \end{aligned}$$

To compute the cluster size of v, one evaluates three $(r+1) \times (m+1) \times (n+1)$-tables: $ND(l,i,j) = |D(l,i,j)|$, $NR(l,i,j) = |R(l,i,j)|$, and $NI(l,i,j) = |I(l,i,j)|$, for $i \in [0,m]$, $j \in [0,n]$, and $l \in [E(i,j), E(i,j)+r]$. It immediately follows that $g(v) = \sum_{l=E(m,n)}^{k} ND(l,m,n) + NR(l,m,n)$.

Recurrences for computing each entry in the three tables in constant time can simply be derived from the recurrences of Lemma 7. Note that whenever $k < m$, it suffices to evaluate the table entries in the band of width k around the main diagonal, as described in Section 5. Thus $O(r \cdot \min(m,k) \cdot n)$ values are to be computed. At any time, only $O(r \cdot \min(m,k))$ of these need to be stored. Recall that $n \leq m + k \in O(m)$ and $r < \Delta$. Hence we can conclude with:

Lemma 8. *The cluster size $g(v)$ can be computed in $O(\Delta \cdot \min(m,k) \cdot m)$ time and $O(\Delta \cdot \min(m,k))$ space.*

Note that for the unit cost function we have $k < m$ and $\Delta = 1$, i.e., the cluster size can be computed in $O(km)$ time and $O(k)$ space.

7 Experimental Results

We implemented our estimator in C, and performed experiments to demonstrate its accuracy, its convergence, and its speed. In the first three experiments we used an alphabet of size four and the unit cost function. For this cost function our implementation runs in $O(tmk)$ time and $O(mk)$ space. We achieved a considerable speedup in practice by using the failure function of the Knuth-Morris-Pratt algorithm [7]: whenever a column in table E is computed such that the minimal entry is k, the remaining columns need not be evaluated. Instead, using the precomputed failure function, one can decide $v \in CN_k(P)$ and compute $g(v)$ in $O(k)$ additional time. For the details of this technique, see [10, page 352].

In the first experiment, we employed the neighborhood construction algorithm of [10], to compute the "real" probability $Pr = Pr[A_k(P,T)]$ for $k = 2$ and 10,000 random patterns of length $m = 20$. We applied our procedure to the same threshold value and the same random patterns, and compared the resulting estimation Pr_e^t with Pr after each trial $t \in [1, 2000]$, by evaluating the deviation $a(t) = 100 \cdot |Pr - Pr_e^t|/Pr$. Figure 1 shows the probability that $a(t) > d$ for $d = 10\%, 20\%, \ldots, 50\%$ and $t \in [1, 2000]$. It reveals that after 1,000 trials we can expect our procedure to compute a very good estimation of the real probability. The average of $a(t)$ over all 10,000 random patterns was 6.99% after 1,000 trials, the median was 8.97%, and the standard deviation was 4.93%.

In the second experiment, we chose a fixed random pattern of length $m = 50$ and $k = 10$. For $r \in [8, 16]$ we evaluated $c(2^r) = 100 \cdot |\log_2(Pr_e^{2^r}/Pr_e^{2^{r-1}})|$, where Pr_e^t is the estimated probability after t trials. Figure 2 shows the probability that $c(2^r) > d$ for $d = 10\%, 20\%, \ldots, 50\%$ in 1,000 runs of our procedure. (In each run, the random number generator used for selecting random edit scripts, was started with a different seed.) One recognizes that the oscillation of the estimated probability becomes smaller, the larger the number of trials. For instance, the average of $c(2^{16})$ was 29.6% over all 1,000 runs, the median was 21.6%, and the standard deviation was 28.0%. For $C(2^{15})$ the corresponding values were 35.7%, 26.8%, and 35.7%. To get an idea of the size of the set we sampled from, we computed the estimation $1.23 \cdot 10^{18}$ for $|CN_k(P)|$, by dividing $|CC(P)|$ by the average cluster size obtained in the successful trials. These numbers show that 2^{16} trials are not a large effort compared to the alternative of enumerating the entire condensed k-neighborhood.

In the third experiment, we ran 10,000 trials with 100 random patterns of length $m \in \{10, 20, \ldots, 50\}$ and error rates $m/k \in \{10\%, 20\%\}$. The average running times (in seconds) on a DEC Alpha $200^{4/233}$ are shown in the following table:

m	10	20	30	40	50
10%	0.2	0.6	1.2	1.9	2.8
20%	0.3	1.0	2.0	3.2	4.7

In the fourth and fifth experiment, we used the 20-character alphabet of amino-acids and the PAM120 [3] scoring function σ with score -8 for insertions

and deletions. We estimated the probability $Pr_\sigma[A_k(P,T)]$ that a pattern string P of length m approximately matches some prefix v of a random string T such that the length-relative score $\sigma(P,v)/|s|$ is greater or equal to k', where $k' = 0.8$ and $|s|$ is the length of the shortest optimal edit script for P generating v. Recall that PAM120-scores are to be maximized, i.e., $\sigma(P,v)$ is the *maximal* score of any edit script for P generating v. Since σ has negative replacement scores, we transformed it into a cost function δ defined by $\delta(\alpha \to \beta) = -\sigma(\alpha \to \beta) + x$ where $x = \max_{a,b \in \Sigma} \sigma(a \to b) = 12$. δ is an integer cost function satisfying (1) and (2). Moreover, one can show that $\sigma(P,v)/|s| \geq k'$ if and only if $\delta(P,v) \leq (x - k') \cdot |s|$. δ assigns cost 20 to deletions and insertions. The maximal cost for replacements is 20, and the average is 14.01. Hence, in an optimal edit script deletions and insertions are rare, i.e., m is a good approximation for $|s|$. Therefore, we can expect a good estimation for $Pr_\sigma[A_k(P,T)]$ if we run our procedure with δ and $k = (x - k') \cdot m = 11.2 \cdot m$.

In the fourth experiment, we again evaluated $c(2^r)$ for $r \in [8, 16]$. This time we chose a random pattern of length $m = 20$ and $k = 11.2 \cdot m = 224$. The size of the sampled set $CN_k(P)$ was estimated to be $2.03 \cdot 10^{22}$. Figure 3 shows the probability that $c(2^r) > d$ for $d = 10\%, 20\%, \ldots, 50\%$ in 1,000 runs of our procedure. As in Figure 2, the oscillation of the estimated probability decreases, with the number of trials becoming larger. After about 2^{14} trials one can recognize the convergence of the estimation value. For instance, the average of $C(2^{16})$ was 12.1% over all 1,000 runs, the median was 8.4%, and the standard deviation was 15.8%. For $C(2^{14})$ the corresponding values are 14.2%, 9.8%, and 15.5%.

In the fifth experiment, we ran 10,000 trials with 100 different random patterns of length $m \in \{10, 20, \ldots, 50\}$ and $k = 11.2 \cdot m$. The average running times (in seconds) on a DEC Alpha $200^{4/233}$ are shown in the following table:

m	10	20	30	40	50
	2.7	6.7	12.2	18.7	26.8

We did not perform any experiments verifying the accuracy of our estimator for the 20-character alphabet and σ. This is because for interesting values of m and k the condensed k-neighborhood was too large to be enumerated completely.

Acknowledgements

We wish to thank Will Evans for discussions on the convergence properties of our estimator.

References

1. L. Allison and C.S. Wallace. The Posterior Probability Distribution of Alignments and Its Application to Parameter Estimation of Evolutionary Trees and to Optimization of Multiple Alignments. *J. Mol. Evol.*, **39**:418–430, 1994.

2. W.I. Chang and J. Lampe. Theoretical and Empirical Comparisons of Approximate String Matching Algorithms. In *Proc. of* CPM 92*,* LNCS 644, pages 175–184, 1992.
3. M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. A Model of Evolutionary Change in Proteins. Matrices for Detecting Distant Relationships. *Atlas of Protein Sequence and Structure*, **5**:345–358, 1978.
4. W.M. Fitch. Random Sequences. *J. Mol. Biol.*, **163**:171–176, 1983.
5. S. Kannan, Z. Sweedyk, and S. Mahaney. Counting and random generation of strings in regular languages. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 551–557, 1995.
6. S. Karlin and S.F. Altschul. Methods for Assessing the Statistical Significance of Molecular Sequence Features by using General Scoring Schemes. *Proc. Nat. Acad. Sci. U.S.A.*, **87**:2264–2268, 1990.
7. D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast Pattern Matching in Strings. *SIAM Journal on Computing*, **6**(2):323–350, 1977.
8. G. Mehldau and E.W. Myers. A System for Pattern Matching Applications on Biosequences. *Comp. Appl. Biosci.*, **9**(3):299–314, 1993.
9. E.W. Myers. An $O(ND)$ Differences Algorithm. *Algorithmica*, **2**(1):251–266, 1986.
10. E.W. Myers. A Sublinear Algorithm for Approximate Keyword Searching. *Algorithmica*, **12**(4/5):345–374, 1994.
11. E.W. Myers. Approximate Matching of Network Expressions with Spacers. *J. Comp. Biol.*, **3**(1):33–51, 1996.
12. E. Ukkonen. Algorithms for Approximate String Matching. *Information and Control*, **64**:100–118, 1985.
13. R.A. Wagner and M.J. Fischer. The String to String Correction Problem. *Journal of the ACM*, **21**(1):168–173, 1974.

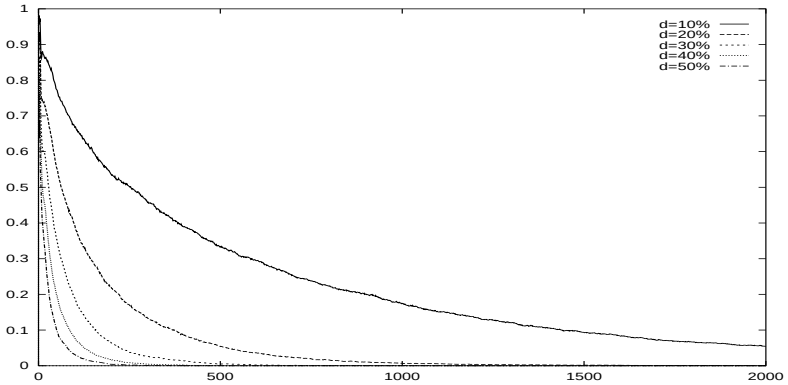

Fig. 1. $Pr[a(t) > d]$ for 10,000 patterns ($|\Sigma| = 4, m = 20, k = 2$)

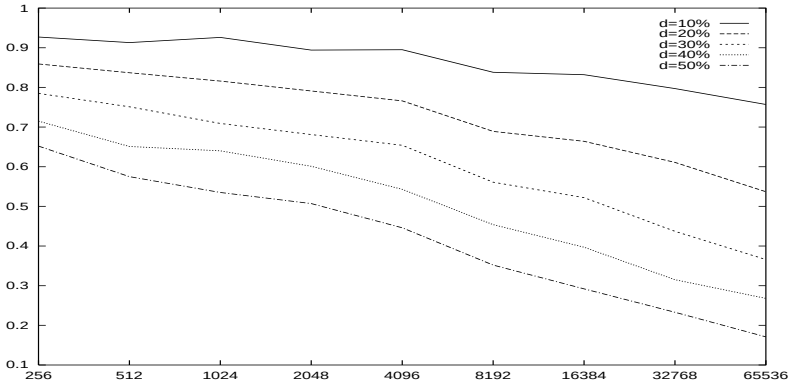

Fig. 2. $Pr[c(2^r) > d]$ for a fixed pattern and 1,000 runs ($|\Sigma| = 4, m = 50, k = 10$)

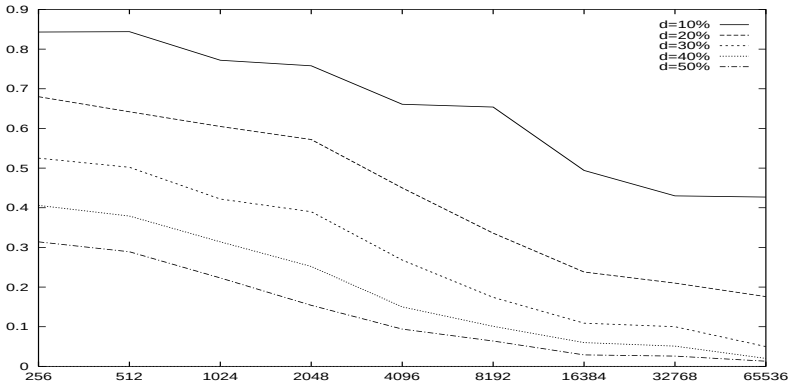

Fig. 3. $Pr[c(2^r) > d]$ for a fixed pattern and 1,000 runs ($|\Sigma| = 20, m = 20, k = 224$)