Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Algorithms 57 (2005) 130–139

Journal of Algorithms

www.elsevier.com/locate/jalgor

FFT-based algorithms for the string matching with mismatches problem

Tor Schoenmeyr, David Yu Zhang [*]

California Institute of Technology, USA

Received 29 October 2002

Available online 21 February 2005

Abstract

The *string matching with mismatches* problem requires finding the Hamming distance between a pattern P of length m and every length m substring of text T with length n. Fischer and Paterson's FFT-based algorithm solves the problem without error in $O(\sigma n \log m)$, where σ is the size of the alphabet Σ [SIAM–AMS Proc. 7 (1973) 113–125]. However, this in the worst case reduces to $O(nm \log m)$. Atallah, Chyzak and Dumas used the idea of randomly mapping the letters of the alphabet to complex roots of unity to estimate the score vector in time $O(n \log m)$ [Algorithmica 29 (2001) 468–486]. We show that the algorithm's score variance can be substantially lowered by using a bijective mapping, and specifically to zero in the case of binary and ternary alphabets. This result is extended via alphabet remappings to deterministically solve the string matching with mismatches problem with a constant factor of 2 improvement over Fischer–Paterson's method.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Randomized algorithms; String matching with mismatches; Hamming distance; FFT

1. Introduction

In this paper, we address methods of solving the string matching with mismatches problem, also known as the Hamming distance problem. Given text T of length n and pattern P

[*] Corresponding author. Fax: (626) 584-0630.
E-mail address: dzhang@dna.caltech.edu (D.Y. Zhang).

0196-6774/$ – see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jalgor.2005.01.001

Position									i									
Text	...	a	p	p	r	o	x	i	m	a	t	e		s	t	r	i	...
Pattern									m	a	t	c	h	i	n	g		
Score vector	...	0	1	0	0	0	0	0	3									

Fig. 1. Calculation of the score vector. The score at the ith position is 3 because the letters m, a, and t match.

of length m, both constructed from the alphabet Σ, we wish to find the score vector c of matching P to every length m substring of text T (see Fig. 1). The vector c is defined as:

$$\forall i \in [0, n-m+1] \quad c_i = \sum_{k=0}^{m-1} \delta(P_i, T_{i+k}) \tag{1}$$

where

$$\delta(a,b) = \begin{cases} 1 & \text{if } a = b, \\ 0 & \text{otherwise.} \end{cases}$$

The naive algorithm is to compute each entry of the score vector separately in $O(m)$, resulting in total time complexity $O(nm)$, assuming constant time to compare two letters.

Fischer and Paterson introduced the idea of using the FFT and convolution to calculate many entries of the score vector simultaneously [2]. Each letter is masked, and the contributions to the score vector from each of σ letters is calculated independently in $O(n \log m)$ time and then summed, which leads to a total time complexity of $O(\sigma n \log m)$. This result is very good for small finite alphabets. For very large or unbounded alphabets (such as the set of natural numbers), the effective alphabet size becomes $O(m)$, and the worst case time complexity becomes $O(nm \log m)$, which is actually worse than the naive algorithm.

Fischer and Paterson also introduced a way to produce an estimate of the score vector $\hat{c}$ in time $O(n \log m \log \sigma)$ using binary encoding of the alphabet, which becomes $O(n \log^2 m)$ in the worst case.

Atallah, Chyzak and Dumas presented an algorithm based on mapping the alphabet to complex roots of unity, which allows for calculation of an estimate of the score vector $\hat{c}$ in $O(n \log m)$ time [1]. The other major achievement of the Atallah–Chyzak–Dumas algorithm is that the error of the estimate ε possesses a zero-centered symmetrical distribution.

We revisit Atallah, Chyzak and Dumas' algorithm in more detail in Section 2, and show that replacing the random mapping function with a bijective one (permutation) greatly decreases the variance on the error of the score vector estimate. The error is no longer zero-centered, but this can be quickly corrected by using a linear transformation.

In Section 3, we show a method to extend the bijectively mapped algorithm to solve the string matching with mismatches problem with zero error rate, with a constant factor of 2 in time reduction from the Fischer–Paterson deterministic algorithm. Although this gain is not asymptotic, it is significant for practical use in small alphabets (such as DNA, and amino acid sequences).

Finally, in Section 4, we discuss the results of the implementation of our permutation-based algorithm in practice.

2. Bijective mapping

2.1. *The Atallah–Chyzak–Dumas algorithm in detail*

Since this paper is based on the Atallah–Chyzak–Dumas algorithm, we shall first describe it in more detail.

The key idea is that of first mapping every letter X of the alphabet Σ randomly to an integer in $[0, \sigma)$ with a function $\vartheta(\cdot)$, and then remapping to $\omega_\sigma^{\vartheta(X)}$, the σth root of unity defined as $e^{i2\pi\vartheta(X)/\sigma}$. In the text T, the letters are replaced accordingly, while in the pattern P, the letters are replaced with the geometric inverse of the root of unity defined for the letter. In this way, when the inner product is taken between the letter in the text and the letter in the pattern, a correct match will contribute score 1, while a mismatch would contribute a quantity with expected value 0. Note that this mapping used by Atallah, Chyzak and Dumas is not injective (one-to-one), so that mapping to nth roots of unity, where $n > \sigma$, would yield similar results.

The convolution is done between a part of the text versus the reverse of the pattern padded by αm zeros. To simplify convolution calculation, both the text and pattern are padded with additional zeros up to the closest power of two. After a tiny bit of rearranging, we achieve αm correct Hermitian products. The algorithm is summarized in Fig. 2.

```
1. Randomly construct a mapping function $\vartheta : X \in \Sigma \to i \in [0, \sigma)$.
2. Map the text characters to roots of unity to form $\tilde{t}$
       $\tilde{t}_i \leftarrow \omega_\sigma^{\vartheta(t_i)}$.
3. Map the pattern character to roots of unity to form $\tilde{p}$
       $\tilde{p}_i \leftarrow \omega_\sigma^{-\vartheta(p_i)}$.
4. Define $\alpha$, the chunk size parameter.
   (a) Define chunk size $\zeta$ as the closest power of 2:
           $\zeta = 2^{\lceil log_2(\alpha+1)m \rceil}$.
5. For $a = 0, 1, \ldots, \lceil \frac{n}{\alpha m} \rceil - 1$:
   (a) Define text chunk: for $b = 0, 1, \ldots, \zeta - 1$,
           $\tau_b = \tilde{t}_{a\alpha m+b}$.
   (b) Define pattern chunk as reverse of the pattern: for $b = 0, 1, \ldots, \zeta - 1$,
           $\rho_b = \tilde{p}_{\zeta-1-b}$,
           $\forall c \notin [0, m-1], \quad \tilde{p}_c = 0$.
   (c) Perform the fast convolution
           $\gamma \leftarrow \mathrm{IFFT}(\mathrm{FFT}(\tau) * \mathrm{FFT}(\rho))$.
   (d) Save $\alpha m$ terms of the convolution as score vector components: for $b = 0, 1, \ldots, \alpha m - 1$,
           $c_{a\alpha \hat{m}+b} = \gamma_{b-1}$     where $\gamma_{-1} = \gamma_{\zeta-1}$.
```

Fig. 2. Atallah–Chyzak–Dumas algorithm.

The time complexity of processing each chunk is $O((\alpha+1)m\log((\alpha+1)m)) = O(\alpha m(\log\alpha+\log m)) = O(\alpha m\log m)$ for $\alpha \ll m$. There are $\frac{n}{\alpha m}$ chunks to process, so the runtime of each iteration is $\frac{n}{\alpha m}O(\alpha m\log m) = O(n\log m)$. This can be repeated on z iterations to reduce error of the score estimates, bringing the total time to $O(zn\log m)$.

At this point, the calculated scores in vector $\hat{c}$ are complex, with both imaginary parts and real parts. The expected value of the $\mathrm{Re}(\hat{c_i})$ is c_i, the actual score at position i, and thus it can be taken as an estimate. Since character matches contribute zero to the imaginary part of $\hat{c}$, while mismatches contribute a zero-centered random quantity, $\mathrm{Im}(c_i)$ is similar to the result of an unbiased one-dimensional random walk. As such, it correlates positively with the score error $\varepsilon_i = |c_i - \hat{c_i}|$.

However, since we are using a static mapping, the steps of the random walk are not independent, and thus the correlation between $\mathrm{Im}(\hat{c_i})$ and ε_i is only very weak. In the rest of this paper, we choose to ignore this information, and focus on $\mathrm{Re}(\hat{c_i})$.

2.2. Bijective mapping of the alphabet to primitive roots of unity

The mapping function from the alphabet to primitive roots of unity can be replaced with a bijective one, $\theta(\cdot)$. We sometimes refer to the bijective mapping as permutation in this paper, because in the actual implementation, we start with a fixed bijective function, and then “shuffle” the mappings. The bijective mapping function can be randomly constructed in $O(\sigma)$ time [4].

In the original Atallah–Chyzak–Dumas algorithm, mismatches are mapped to random roots of unity, uniformly distributed; this is no longer the case when using a bijective mapping function, because a mismatch will never be mapped to 1. Thus, the score estimates produced no longer possess expected value equal to that of the true score.

However, mismatches are mapped to all $(\sigma-1)$ other roots of unity with equal probability.

$$\begin{aligned}
\mathrm{E}\bigl(\omega_\sigma^{\theta(t_i)-\theta(p_j)}\bigr) &= \delta(t_i,p_j) + \bigl(1-\delta(t_i,p_j)\bigr)\frac{1}{\sigma(\sigma-1)}\sum_{k\neq l}\omega_\sigma^{k-l} \\
&= \delta(t_i,p_j) + \bigl(1-\delta(t_i,p_j)\bigr)\frac{1}{\sigma(\sigma-1)}\left(\sum_{k,l}\omega_\sigma^{k-l} - \sum_k \omega_\sigma^{k-k}\right) \\
&= \delta(t_i,p_j) + \bigl(1-\delta(t_i,p_j)\bigr)\frac{1}{\sigma(\sigma-1)}(0-\sigma) \\
&= \delta(t_i,p_j) - \frac{(1-\delta(t_i,p_j))}{(\sigma-1)}.
\end{aligned}$$

We calculate the expected value of $\tilde{c}$, the score obtained from the FFT convolution after using a bijective mapping function, relative to the true score c.

$$\mathrm{E}(\tilde{c}_i) = \mathrm{E}\left(\sum_{k=0}^{m-1}\omega_\sigma^{\theta(t_{i+k})-\theta(p_k)}\right) = \sum_{k=0}^{m-1}\mathrm{E}\bigl(\omega_\sigma^{\theta(t_{i+k})-\theta(p_k)}\bigr) = c_i - \frac{(m-c_i)}{(\sigma-1)}.$$

An algebraic rearrangement thus leads us to:

$$\mathrm{E}\left(\tilde{c_i}+\frac{m-\tilde{c_i}}{\sigma}\right)=c_i. \tag{2}$$

We can thus adjust our score estimates to recenter for the negative expected contributions of the mismatches. Our estimated score vector may have non-zero imaginary parts (even though the expected value is zero); we simply remove the imaginary part when interpreting the score vector. The computational complexity is the same as when mappings were used, $O(n\log m)$.

2.3. Variance comparisons

In investigating the variance of the bijectively-mapped method with that of the original, it is important to note that both algorithms' error increases as the true score decreases. Both methods contribute 1 to the score vector for each character match, so in the case of perfect matches, neither algorithm will produce any error.

We consider the variance contribution of a single mismatch to a score vector. Define x_ϑ to be the random variable that denotes the real part of the score contribution of a single mismatch, using the random mapping function $\vartheta(\cdot)$: $x_\vartheta=\mathrm{Re}(\omega^{\vartheta(l)-\vartheta(k)})$, with $l\neq k$.

$$\begin{aligned}\mathrm{Var}(x_\vartheta)&=\mathrm{E}\big(x_\vartheta^2\big)-\mathrm{E}(x_\vartheta)^2\\&=\frac{1}{\sigma(\sigma-1)}\left(\sum_{k=0}^{\sigma-1}\sum_{l=0}^{\sigma-1}\cos^2\left(\frac{2\pi(\vartheta(k)-\vartheta(l))}{\sigma}\right)-\sum_{k=0}^{\sigma-1}\cos^2 0\right)\\&\quad-\left(\frac{1}{\sigma(\sigma-1)}\left(\sum_{k=0}^{\sigma-1}\sum_{l=0}^{\sigma-1}\cos\left(\frac{2\pi(\vartheta(k)-\vartheta(l))}{\sigma}\right)-\sum_{k=0}^{\sigma-1}\cos 0\right)\right)^2\\&=\frac{1}{\sigma}\sum_{k=0}^{\sigma-1}\cos^2\left(\frac{2\pi k}{\sigma}\right)-0=\frac{1}{\sigma}\sum_{k=0}^{\sigma-1}\left(\frac{1}{2}+\frac{1}{2}\cos\left(\frac{4\pi k}{\sigma}\right)\right)\\&=\begin{cases}\frac{1}{2} & \text{if } \sigma\neq 2,\\ 1 & \text{if } \sigma=2.\end{cases}\end{aligned}$$

Similarly, define x_θ to be the random variable that denotes the real part of the contribution of a mismatch, using the bijective mapping function $\theta(\cdot)$: $x_\theta=\mathrm{Re}(\omega^{\theta(l)-\theta(k)})$, with $l\neq k$.

$$\begin{aligned}\mathrm{Var}(x_\theta)&=\mathrm{E}\big(x_\theta^2\big)-\mathrm{E}(x_\theta)^2\\&=\frac{1}{\sigma(\sigma-1)}\left(\sum_{k=0}^{\sigma-1}\sum_{l=0}^{\sigma-1}\cos^2\left(\frac{2\pi(\theta(k)-\theta(l))}{\sigma}\right)-\sum_{k=0}^{\sigma-1}\cos^2 0\right)\\&\quad-\left(\frac{1}{\sigma(\sigma-1)}\left(\sum_{k=0}^{\sigma-1}\sum_{l=0}^{\sigma-1}\cos\left(\frac{2\pi(\theta(k)-\theta(l))}{\sigma}\right)-\sum_{k=0}^{\sigma-1}\cos 0\right)\right)^2\end{aligned}$$

$$= \frac{1}{\sigma - 1}\sum_{k=1}^{\sigma-1}\left(\cos^2\left(\frac{2\pi k}{\sigma}\right)\right) - \frac{1}{(\sigma-1)^2}$$
$$= \frac{1}{\sigma - 1}\sum_{k=1}^{\sigma-1}\left(\frac{1}{2} + \frac{1}{2}\cos\left(\frac{4\pi k}{\sigma}\right)\right) - \frac{1}{(\sigma-1)^2}$$
$$= \frac{1}{2} - \frac{1}{(\sigma-1)^2} + \frac{1}{2(\sigma-1)}\left(\sum_{k=0}^{\sigma-1}\left(\cos\left(\frac{4\pi k}{\sigma}\right)\right) - 1\right)$$
$$= \begin{cases} \frac{\sigma(\sigma-3)}{2(\sigma-1)^2} & \text{if } \sigma \neq 2, \\ 0 & \text{if } \sigma = 2. \end{cases}$$

Recalling the statistical inequality,

$$\mathrm{Var}(X+Y) \leqslant \left(\sqrt{\mathrm{Var}(X)} + \sqrt{\mathrm{Var}(Y)}\right)^2.$$

We recursively derive the upper bound on the variance of the score estimate to be

$$\mathrm{Var}(\hat{c_i}) = \mathrm{Var}\left(\sum_{k=0}^{m-1} \omega_\sigma^{\theta(t_{i+k})-\theta(p_k)}\right) \leqslant (m-c_i)^2\,\mathrm{Var}(x). \tag{3}$$

This can be repeated, and results averaged for z iterations to reduce the variance to:

$$\mathrm{Var}(\bar{c_i}) \leqslant \frac{(m-c_i)^2\,\mathrm{Var}(x)}{z}. \tag{4}$$

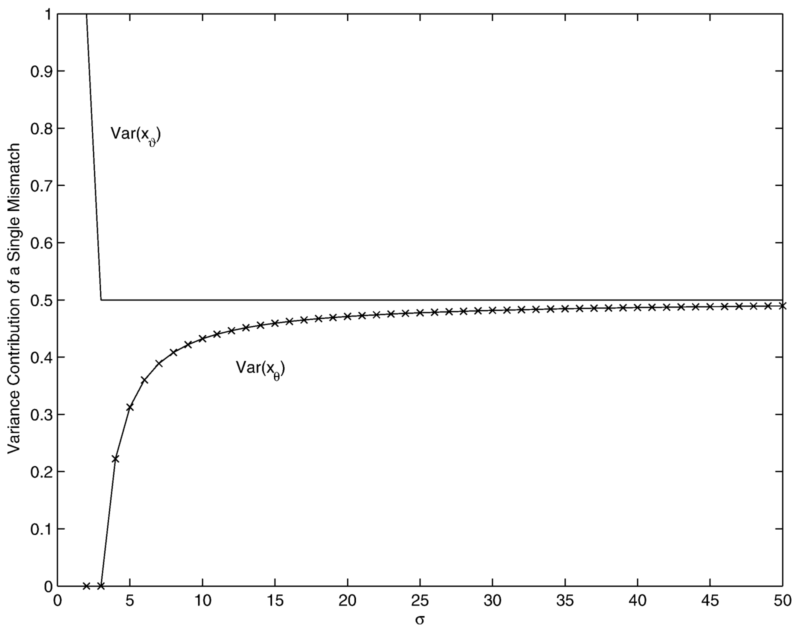

Fig. 3. Variance comparison between random and bijective mappings.

In comparison of the performance of the two mappings in terms of variance, it is seen that the bijective mapping holds great advantage over the random mapping in the case of small alphabets, reaching 0 for binary and ternary alphabets. On the other hand, as σ increases to more than 20, the difference between the two becomes less than 6%.

2.4. *Generalized string matching*

The method of using bijective mappings can be easily extended to allow for all the variants of “generalized string matching” that Atallah, Chyzak and Dumas mentioned in their paper, including specifically weighted string matching, “always match” and “never match” symbols, and higher-dimensional array matching. As the application of the replacement of a random mapping function with a bijective one is relatively straightfoward, details concerning the exact implementation are omitted.

3. A deterministic solution

The previous section proved that there is no error when using the bijective-mapping method on alphabets of size three. This feature is used in the design of a deterministic algorithm to solve string matching with mismatches problems for arbitrary alphabets in a way which is faster than the algorithm due to Fischer–Paterson by a constant factor of 2.

Fischer and Paterson’s method for solving the string matching with mismatches problem deterministically (with no error) involves σ iterations of a “masked” string comparison [2]. In each iteration, a different letter X is picked, and all instances of X in both text and pattern are replaced by 1, while all other letters are replaced by 0. The number of matches from the letter X is the calculated using the inner product, calculated quickly using the FFT and convolution. Finally, the score contributions from all letters are summed (see Fig. 4).

Our method is as follows: First we group the letters of the alphabet into arbitrary pairs P, composed of letters P_1 and P_2. We performed $\sigma/2$ iterations, one on each pair. On pair P’s iteration, all instances of the letter P_1 in the text are replaced with ω_3^1 $(-1/2 + i\sqrt{3}/2)$, all instances of letter P_2 are replaced with ω_3^2 $(-1/2 - i\sqrt{3}/2)$, and all other letters are replaced with 0. In the pattern, all instances of P_1 are replaced with ω_3^2, all instances of P_2 are replaced with ω_3^1, and all other letters are replaced with ω_3^0. A score for the pair is then calculated by taking the inner product and then the scores of all pairs are summed, just as in Fischer–Paterson (see Fig. 5).

Text	…	A	C	A	B	…
Pattern		A	B	D	B	

	A				B				C				D				
Text	1	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	
Pattern	1	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	
Score	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	= 2

Fig. 4. Fischer–Paterson deterministic algorithm.

Text	...	A	C	A	B	...	
Pattern		A	B	D	B		

$P_1 = \{A, B\}$ $\quad P_2 = \{C, D\}$

	P_1				P_2				
Text	ω_3^1	0	ω_3^1	ω_3^2	0	ω_3^1	0	0	
Pattern	ω_3^2	ω_3^1	ω_3^0	ω_3^1	ω_3^0	ω_3^0	ω_3^1	ω_3^0	
Score	ω_3^3	0	ω_3^1	ω_3^3	0	ω_3^1	0	0	
Re(Score)	1	0	$-\frac{1}{2}$	1	0	$-\frac{1}{2}$	0	0	$=1$

Recentering $\quad \text{Score} = \hat{c} + \frac{m-\hat{c}}{\sigma} = 1 + \frac{4-1}{3} = 1 + 1 = 2$

Fig. 5. Our deterministic algorithm.

This score is not accurate yet, as we have not adjusted for the bijective mapping. Each letter of the pattern contributes score 1 in the case of a match and $-1/2$ in the case of a mismatch, since we have reduced the alphabet size to 3, and thus the recentering adjustment at the end, as according to Eq. (2), is done with $\sigma = 3$, as opposed to the original alphabet size.

As a practical note, since the Fast Fourier Transform and the Inverse Fast Fourier Transform both require floating point precision, there are no time advantages for the Fischer–Paterson algorithm having only the terms 1 and 0, and thus our algorithm is truly a factor of two faster. This non-asymptotic gain may yield practical gains for string matching purposes.

4. Implementation

A computer program was written in C to compare the performance of our random and deterministic algorithms as well as that of Atallah et al. We performed analysis on texts ranging in size up to 4.7 million (the E. coli genome [3]) and pattern ranging in size up to 87,000. The alphabet size for this experiment was 4 (DNA). The y-axis of Fig. 6 represents the average error in the average of score vectors produced by the number of recursions run denoted on the x-axis. The average error is defined as follows:

$$\varepsilon = \frac{1}{n-m+1} \sum_{i=1}^{n-m+1} \varepsilon_i = \frac{1}{n-m+1} \sum_{i=1}^{n-m+1} \left|\bar{c_i} - c_i\right|. \tag{5}$$

$\bar{c_i}$ is the score at position i computed from the average of the recursions, and c_i is the real score at that position. The real scores were calculated using our deterministic algorithm.

Note that Fig. 6 is plotted on a semi-log graph. As can be seen, permuting/bijective mapping yields stabilization of error near 200 errors, while random mapping yields stabilization of errors near 3000 errors. Both algorithms have occasional fluctuations in average error count. Under the conditions of total pattern size of about 90,000 while the alpha-

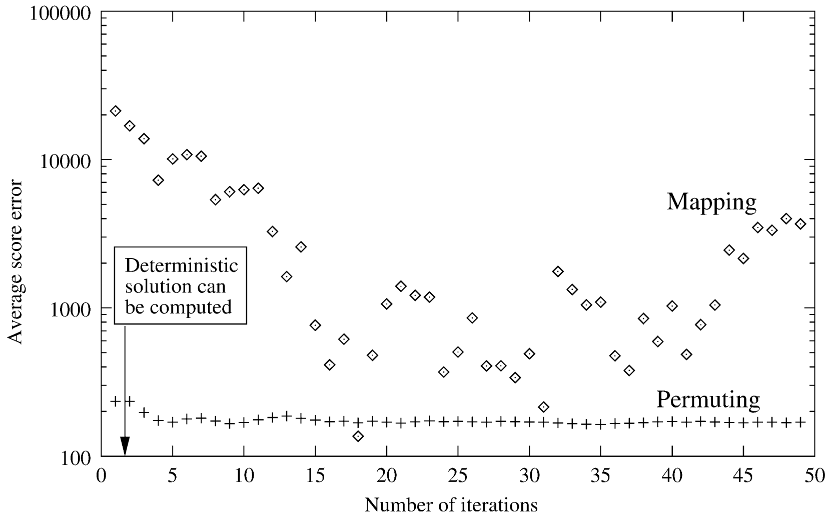

Fig. 6. Sample runs of the algorithms on the E. coli genome. The text was the full E. coli genome, and the pattern was an arbitrarily selected portion of the genome. $\sigma = 4$, $n = 4639220$, $m = 86239$.

bet size is 4, permuting is about 15 times better. This is actually substantially better than predicted by theory, which prognosticates about a ratio of about 2.25 ($\frac{1/2}{2/9}$).

Both algorithms exhibited strong improvement over the first few repetitions—from more than 20,000 to under 10,000 for mapping and from more than 220 to under 180 for permuting in the first 4 rounds.

As expected, the benefit of using our non-deterministic algorithm diminishes with increasing σ. In the case of $\sigma = 97$ where the abstract to this paper was compared to the book “Hard Times” by Charles Dickens (not shown), performance was similar to Atallah’s algorithm, as expected from our theoretical calculations.

Also shown is the time equivalent number of iterations required for the deterministic algorithm to run. Since the deterministic algorithm never produces any error in the score vector, the graphs are useful for showing the threshold error requirement below which it is beneficial to run the deterministic algorithm.

The source code is available in C upon request.

Acknowledgment

The authors would like to heartily thank Paul Rothemund for assistance in proofreading this paper.

References

[1] M.J. Atallah, F. Chyzak, P. Dumas, A randomized algorithm for approximate string matching, Algorithmica 29 (2001) 468–486.

[2] M.J. Fischer, M.S. Paterson, String matching and other products, SIAM–AMS Proc. 7 (1973) 113–125.

[3] T. Hayashi, et al., Complete genome sequence of enterohemorrhagic Escherichia coli O157:H7 and genomic comparison with a laboratory strain K-12, DNA Research 8 (1) (2001) 11–22.
[4] D.E. Knuth, The Art of Computer Programing, vol. 2, Addison–Wesley, 1998, pp. 145–147.