Implementing Approximate Regularities

Manolis Christodoulakis Costas S. Iliopoulos

Department of Computer Science
King's College London

Kunsoo Park

School of Computer Science and Engineering,
Seoul National University

Jeong Seop Sim

Electronics and Telecommunications Research Institute
Daejeon 305-350, Korea

Abstract

In this paper we study approximate regularities of strings, that is, approximate periods, approximate covers and approximate seeds. We explore their similarities and differences and we implement algorithms for solving the *smallest distance approximate period/cover/seed* problem and the *restricted smallest approximate period/cover/seed* problem in polynomial time, under a variety of *distance* rules (the Hamming distance, the edit distance, and the weighted edit distance). We then analyse our experimental results to find out the time complexity of the algorithms in practice.

Key words: approximate regularities, approximate period, approximate cover, approximate seed, Hamming distance, edit distance, weighted edit distance, smallest distance approximation, restricted smallest distance approximation

Email addresses: manolis@dcs.kcl.ac.uk (Manolis Christodoulakis), csi@dcs.kcl.ac.uk (Costas S. Iliopoulos), kpark@theory.snu.ac.kr (Kunsoo Park), simjs@etri.re.kr (Jeong Seop Sim).

1 Introduction

Finding *regularities* in strings is useful in a wide area of applications which involve string manipulations, such as molecular biology, data compression and computer-assisted music analysis. Typical regularities are repetitions, periods, covers and seeds.

In applications such as molecular biology and computer-assisted music analysis, finding exact repetitions is not always sufficient. A more appropriate notion is that of *approximate* repetitions [2,4,5], where errors are allowed. In this paper, we consider three different kinds of approximation: the *Hamming distance*, the *edit disctance* and the *weighted edit distance*.

Sim, Iliopoulos, Park and Smyth showed polynomial time algorithms for finding approximate periods [6] and, Sim, Park, Kim and Lee showed polynomial time algorithms for the approximate covers problem in [7]. More recently, Christodoulakis, Iliopoulos, Park and Sim showed polynomial time algorithms for the approximate seeds problem [1]. In this paper we implement and compare the algorithms given in [6,7,1].

2 Preliminaries

2.1 Distance functions

We call the *distance* $\delta(x, y)$ between two strings x and y, the minimum cost to transform one string x to the other string y. The special symbol Δ denotes the absence of a character (i.e. an insertion or a deletion occurs).

The *edit* or *Levenshtein distance* between two strings is the minimum number of *edit operations* that transform one string into another. The edit operations are *insertion*, *deletion* and *substitution*, each of cost 1.

The *Hamming distance* between two strings is the minimum number of *substitutions* that transform one string to the other. Note that the Hamming distance can be defined only when the two strings have the same length, because it does not allow insertions and deletions.

We also consider a generalized version of the edit distance model, the *weighted edit distance*, where each insertion, deletion, substitution has a different cost, stored in a *penalty matrix*.

2.2 Approximate Regularities

We give here the definitions of the approximate periods, covers and seeds. These definitions are expressed in a different way from the corresponding original definitions given in [6,7,1]. This is done in order to expose their similarities and provide us with the "background" common to all three of them.

Let x and s be strings over Σ^*, δ be a distance function, t be an integer and $s_1, s_2, \ldots, s_r$ $(s_i \neq \varepsilon)$ be strings such that $\delta(s, s_i) \leq t$, for $1 \leq i \leq r$.

Definition 1 *s is a t-*approximate period *of x if and only if there exists a superstring $y = xv$ (*right extension*) of x that can be constructed by concatenating copies of the strings $s_1, s_2, \ldots, s_r$.*

Definition 2 *s is a t-*approximate cover *of x if and only if x can be constructed by overlapping or concatenating copies of the strings $s_1, s_2, \ldots, s_r$.*

Definition 3 *s is a t-*approximate seed *of x if and only if there exists a superstring $y = uxv$ (*right and left extensions*) of x that can be constructed by overlapping or concatenating copies of the strings $s_1, s_2, \ldots, s_r$.*

Table 1
Comparison between periods, covers and seeds.

	what is covered	*how it is covered*
periods	right extension of the text	concatenations
covers	the text itself	concatenations and overlaps
seeds	left and right extension of the text	concatenations and overlaps

3 Problem Definitions and Solutions

3.1 Smallest Distance Approximate Period/Cover/Seed Problem

Definition 4 *Let x be a string of length n, s be a string of length m, and δ be a distance function. The* Smallest Distance Approximate Period/Cover/Seed *problem is to find the minimum integer t such that s is a t-approximate period/cover/seed of x.*

There are two steps involved to solve this problem:

(1) *Compute the distance between s and every substring of x.*

Let w_{ij} be the distance between s and $x[i..j]$, for $1 \leq i \leq j \leq n$, that is $w_{ij} = \delta(x[i..j], s)$. Section 3.1.1 explains in detail how these w_{ij}'s are computed.

(2) *Compute the minimum t such that s is a t-approximate period/cover/seed of x.*
Let t_i be the minimum value such that s is a t_i-approximate period/cover/seed of $x[1..i]$. Initially, $t_0 = 0$. For $i = 1$ to n, where n is the length of the text x, we compute

$$t_i = \min_{h_{min} \leq h \leq h_{max}} \{\max\{\min_{h \leq j \leq j_{max}} \{t_j\}, w_{h+1,i}\}\}$$

The value t_n is the minimum t such that s is a t-approximate period/cover/seed of x. The values of $h_{min}, h_{max}, j_{max}$ depend on the regularity (period, cover or seed) we are computing and on the distance function we are using, and will be explained in section 3.1.2.

3.1.1 Step 1: The distance between s and every substring of x.

This step resolves the matter of *what* is covered, as described in the definitions of approximate periods, covers and seeds.

The method we use to compute the distance between s and $x[i..j]$, for $1 \leq i \leq j \leq n$, depends on the distance function we are using. For the Hamming distance, that is the character-by-character comparison. For the edit and weighted edit distance we use dynamic programming; to compute the distance between two strings, x and y, a dynamic programming table, called the *D-table*, of size $(|x|+1) \times (|y|+1)$, is used. Each entry $D[i,j]$, $0 \leq i \leq |x|$ and $0 \leq j \leq |y|$, stores the minimum cost of transforming $x[1..i]$ to $y[1..j]$. Initially, $D[0,0] = 0$, $D[i,0] = D[i-1,0] + \delta(x[i], \Delta)$ and $D[0,j] = D[0,j-1] + \delta(\Delta, y[j])$. Then we can compute all the entries of the D table in $O(|x||y|)$ time by the following recurrence:

$$D[i,j] = \min \begin{cases} D[i-1,j] & + \delta(x[i], \Delta) \\ D[i,j-1] & + \delta(\Delta, y[j]) \\ D[i-1,j-1] & + \delta(x[i], y[j]) \end{cases}$$

where $\delta(a,b)$ is the cost of substituting character a with character b, $\delta(a,\Delta)$ is the cost of deleting a and $\delta(\Delta,a)$ is the cost of inserting a.

Covering a left extension of the text x, can be seen as aligning a suffix of the pattern s, $s[k..m]$, with a prefix of the text, $x[1..j]$, as shown in Figure 1. Therefore, in this case, we need to redefine w_{1j}, instead of being the distance

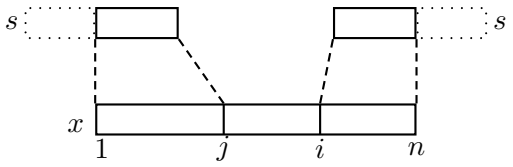


Fig. 1. Covering left and right extensions.

between $x[1..j]$ and s, to be the minimum distance between $x[1..j]$ and any suffix of the pattern, $s[k..m]$. Formally

(A) $w_{1j} = \min_{1 \le k \le m} \{\delta(x[1..j], s[k..m])\}$

Similarly, covering a right extension of the text x, can be seen as aligning a prefix of the pattern s, $s[1..k]$, with a suffix of the text, $x[i..n]$, as shown in Figure 1. Therefore, in this case, we redefine w_{in}, instead of being the distance between $x[i..n]$ and s, to be the minimum distance between $x[i..n]$ and any prefix of the pattern, $s[1..k]$. Formally

(B) $w_{in} = \min_{1 \le k \le m} \{\delta(x[i..n], s[1..k])\}$

Obviously, for approximate periods we use only rule (B), for approximate covers we don't use any of the rules, and for approximate seeds we use both of the rules.

Hamming Distance: Recall that the Hamming distance is defined only for strings of equal length. Therefore we compute those w_{ij}'s for which $j - i + 1 = m$, i.e. the substrings of the text that have size equal to that of the pattern, s. There are exactly $n - m + 1$ such substrings, and each can be compared with the pattern s in m units of time (comparison of m characters). Thus, it takes $(n - m + 1)m = O(mn)$ units of time.

If the rules (A) or (B) are used, then there is an extra cost of $O(m^2)$ units of time for each. Specifically, since again the substrings of x and s must have equal size, each of the rules computes only one distance δ, namely

(A) $w_{1j} = \delta(x[1..j], s[m - j + 1..m])$ and
(B) $w_{in} = \delta(x[i..n], s[1..n - i + 1])$

Therefore, the first time we compare strings of length 1 (cost 1 unit of time), the second time we compare strings of length 2 (cost 2 unit of time), ..., the m-th time we compare strings of length m (cost m unit of time), i.e. the total time consumption is $m(m-1)/2 = O(m^2)$.

However, when computing approxiamte periods (where overlaps are not allowed and there is no left extension), this step takes only $O(n)$ time, since x must be partitioned in n/m blocks of size m (the last block can be of length

Table 2
Time complexity of Step 1: Computing weights.

	cover x	*left extension*	*right extension*	*Overall time*
	Hamming Distance			
periods	$O(n)$	-	-	$O(n)$
covers	$O(mn)$	-	-	$O(mn)$
seeds	$O(mn)$	$O(m^2)$	$O(m^2)$	$O(mn+m^2)$
	Edit Distance			
periods	$O(mn+m^2)$	-	$O(m^2)$	$O(mn+m^2)$
covers	$O(mn+m^2)$	-	-	$O(mn+m^2)$
seeds	$O(mn+m^2)$	$O(m^2)$	$O(m^2)$	$O(mn+m^2)$
	Weighted Edit Distance			
periods	$O(mn^2)$	-	$O(mn)$	$O(mn^2+mn)$
covers	$O(mn^2)$	-	-	$O(mn^2)$
seeds	$O(mn^2)$	$O(mn^2)$	$O(mn)$	$O(mn^2+mn)$

less than m), and for each of these blocks it takes $O(m)$ time to compute its distance with s. □

Edit Distance: When using the edit distance function, the distance between two characters, a and b, is always 1 if $a \neq b$, and 0 if $a = b$. This implies that it is not necessary to compute the edit distances between s and the substrings of x whose lengths are larger than $2m$ because their edit distances with s will exceed m. Therefore, we compute those w_{ij}'s for which $j - i + 1 \leq 2m$ as follows: For each position i of the text x we create a dynamic table D of size $(m+1) \times (2m+1)$, in $2m^2$ units of time. The last row of this table, gives us the values of $w_{ii}, w_{i,i+1}, \ldots, w_{i,i+2m-1}$. Thus, the overall time to cover x is $2m^2n$ units of time.

However, by using the algorithm presented in [3], having computed the D-table for the distance between $x[i..i+2m-1]$ and s, we can update the D-table, for the distance between $x[i+1..i+2m]$ and s, in only $3m$ units of time. Thus, it takes $O(m^2)$ time to create the first D-table and $O(mn)$ time to get the rest n D-tables.

When rule (B) is used, the computation of each w_{in} increases the time complexity by $O(m)$. Since we have already computed the D-table between $x[i..n]$ and s, as described above, it only takes m units of time to find the minimum of the last column of this table, which gives us the required value. There are $2m$ w_{in}'s to be computed, so the overall time complexity for rule (B) is $O(m^2)$.

Rule (A), although a little bit more complicated than (B), also increases the time complexity by $O(m^2)$. The procedure we follow is: we reverse the substring of the text, $x[1..j]$, as well as the pattern, s, and compute the D-table between the reversed strings; the minimum of the last column of this table (computed in $O(m)$ time) is the minimum distance between $x[1..j]$ and any suffix of s. Again, we can make use of the algorithm presented in [3] so that it only takes $2m^2$ units of time to compute the first table and $O(m)$ to update it at each step afterwards. Thus, the time required to compute the $2m$ w_{1j}'s is $O(m^2)$. □

Weighted Edit Distance: In this case, we compute the distance between s and *every* substring of x. The fact that the distance between two characters is arbitrary (given in a penalty matrix), prevents us from using the incremental algorithm in [3], that we used in the edit distance.

Thus, for each position i of the text x we create a new dynamic table D of size $(m+1) \times (n-i)$, in $m(n-i)$ units of time. The last row of this table, gives us the values of $w_{ii}, w_{i,i+1}, \ldots, w_{i,n}$. In total, the time complexity is $mn(n-1)/2 = O(mn^2)$.

Rule (B) is computed similarly with the case of edit distance, with the difference that now there are n w_{in}'s, and so it takes time $O(mn)$.

Rule (A), however, now takes time $O(mn^2)$ since in this case a new D-table has to be computed for the distance between each prefix of x (reversed) and the pattern (also reversed). □

3.1.2 Step 2: The minimum error t

This step resolves the matter of *how* the text (or an extension of it) is covered, as described in the definitions of approximate periods, covers and seeds.

As explained before, for $i = 1$ to n, we compute

$$t_i = \min_{h_{min} \le h \le h_{max}} \{\max\{\min_{h \le j \le j_{max}} \{t_j\}, w_{h+1,i}\}\}$$

Each t_i denotes the cost of "covering" $x[1..i]$. The value t_n, where n is the size of the text x, is the minimum t such that s is a t-approximate period/cover/seed of x.

It works as follows. Consider that we have already computed the values $t_1 \ldots t_{i-1}$, that is, we have already computed the best way to "cover" $x[1..i-1]$. We are now looking for a way to cover $x[1..i]$, with the smallest possible error. We align ("cover") a suffix of $x[1..i]$, namely $x[h+1..i]$, with the pattern s, with

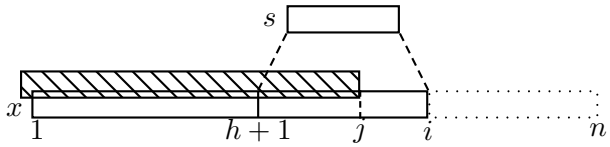


Fig. 2. The second step of the algorithm: The minimum error t.

error $w_{h+1,i}$. What is left to be "covered" is $x[1..h]$. There are two possibilities regarding "covering" $x[1..h]$:

- *Concatenations only*: If overlaps are not allowed (i.e., when computing approximate periods), there is only one way to cover $x[1..h]$, and its cost is t_h. This is achieved by setting $j_{max} = h$.
- *Concatenations and overlaps*: When overlaps are allowed ($j_{max} = i - 1$) there are several (in particular, $i - h - 1$) ways to "cover" $x[1..h]$: cover either $x[1..h]$ (error t_h), or $x[1..h + 1]$ (error t_{h+1}), ... or $x[1..i - 1]$ (error t_{i-1}), (in general $x[1..j]$, with error t_j); we choose the $x[1..j]$ (the shaded box in Fig. 2) that gives the smallest error.

  Allowing overlaps adds an extra cost $O(n)$ (in the worst case). However, by using $O(n)$ additional memory, we can compute the $\min_{h \le j < i} \{t_j\}$ in $O(1)$ time, as follows: For each i we store $min\_t[h] = \min_{h \le j < i} \{t_j\}$, for $0 \le h < i$; in the next round, $i + 1$, $\min_{h \le j < i+1} \{t_j\} = \min\{min\_t[h], t_{i+1}\}$. Note, however, that this trick only works for the edit and weighted edit distance. When the Hamming distance is used, h takes only one value, as we will show later on, and so the min_t vector is not updated correctly.

Next we investigate, how many such suffixes of $x[1..i]$ we should consider depending on the distance function we are using.

Table 3
Time complexity of Step 2: Computing the minimum error.

	Hamming Distance	*Edit Distance*	*Weighted Edit Distance*
periods	$O(n)$	$O(mn)$	$O(n^2)$
covers	$O(mn)$	$O(mn)$	$O(n^2)$
seeds	$O(mn)$	$O(mn)$	$O(n^2)$

Hamming Distance: Recall that when using the Hamming distance, the distance between two strings is defined only when the two strings have the same length. In other words, there is only one suffix of $x[1..i]$—namely, the suffix $x[i - m + 1..i]$— (except, of course, from the case at the beginning and end of the text) that can be "covered" by s, i.e. $h_{min} = h_{max} = i - m$.

Therefore if overlaps are not allowed (i.e. in the case of approximate periods) the time complexity is $O(n)$, since now at each iteration, t_i is computed in $O(1)$ time. In fact, we need not compute all the values t_i, for $1 \le i \le n$,

because, as we explained before, x is actually partitioned in blocks of size m, and thus only the values t_i, $i = m, 2m, \ldots$ are necessary.

If, on the other hand, overlaps are allowed, computing the inner *min* loop requires m units of time, and thus the total time complexity is $O(mn)$. □

Edit Distance: As we mentioned earlier, when we were computing the weights between substrings of the text and the pattern, we do not align copies of the pattern with substrings of the text of length more than $2m$, because their distance is guaranteed to be greater than m. Thus, we consider only the $2m$ suffixes of $x[1..i]$ of length less than or equal to $2m$ ($h_{max} = i - 1$ and $h_{min} = i - 2m$). And since, with or without overlaps, we need $O(1)$ time to compute the inner *min* loop, we need $2mn$ units of time, in this case. □

Weighted Edit Distance: When using the weighted edit distance function, it takes time $O(n^2)$, since at each position i of the text we consider all the suffixes of $x[1..i]$ ($h_{min} = 0$ and $h_{max} = i - 1$). □

Table 4
Time complexity of Problem 1.

	Step 1	*Step 2*	*Overall time*
	Hamming Distance		
periods	$O(n)$	$O(n)$	$O(n)$
covers	$O(mn)$	$O(mn)$	$O(mn)$
seeds	$O(mn + m^2)$	$O(mn)$	$O(mn + m^2)$
	Edit Distance		
periods	$O(mn + m^2)$	$O(mn)$	$O(mn + m^2)$
covers	$O(mn + m^2)$	$O(mn)$	$O(mn + m^2)$
seeds	$O(mn + m^2)$	$O(mn)$	$O(mn + m^2)$
	Weighted Edit Distance		
periods	$O(mn^2 + mn)$	$O(n^2)$	$O(mn^2 + n^2 + mn)$
covers	$O(mn^2)$	$O(n^2)$	$O(mn^2 + n^2)$
seeds	$O(mn^2 + mn)$	$O(n^2)$	$O(mn^2 + n^2 + mn)$

3.2 Restricted Smallest Approximate Period/Cover/Seed Problem

Definition 5 *Given a string x of length n, the* Restricted Smallest Approximate Period/Cover/Seed *problem is to find a substring s of x such that: s is*

a t-approximate period/cover/seed of x and there is no substring of x that is a k-approximate period/cover/seed of x for all $k < t$.

The approach we are using to solve this problem is to consider every substring s of x, of length less than $|x|/2$, as a candidate period/cover/seed and run the algorithm described in the previous section for each s.[1] The reason for choosing the candidate period/cover/seed s to be of length less than $|x|/2$ is because otherwise the longest proper prefix of x (or any long prefix of x) can easily become an approximate period/cover/seed of x with a small distance.

There are $O(n^2)$ such substrings of x to be used as candidate period, cover, or seed, so the time complexities for this problem are as shown in table 5.

Table 5
Time complexity of Problem 2.

	Hamming Distance	*Edit Distance*	*Weighted Edit Distance*
periods	$O(n^3)$	$O(n^3)$	$O(n^4)$
covers	$O(n^3)$	$O(n^3)$	$O(n^4)$
seeds	$O(n^3)$	$O(n^3)$	$O(n^4)$

Since the length of s is not fixed in this case, we use a relative distance function (rather than an absolute distance function); that is, an error ratio, in the case of the Hamming and edit distance, or a weighted edit distance.

To understand why, consider s to be any substring of x of length 1; then, s is always an approximate period/cover/seed of x with error at most 1. The error ratio we used in our experiments is the ratio of the (absolute) minimum error for covering x with s, over the length of the candidate period/cover/seed s.

4 Experimental Results

4.1 Experimental Environment

The algorithms were implemented in C++ using the Standard Template Library (STL), and the tests were run on a Pentium-4M 1.7GHz system, with 256MB of RAM, under the Madrake Linux operating system (v8.0). The dataset we used to test the algorithms is the nucleotide sequence of Saccharomyces cerevisiae chromosome IV.

[1] There also exist other ways to solve this problem (see [6,7,1]), but since they do not offer any improvement in the time or space complexity, we prefer to use this simpler method.

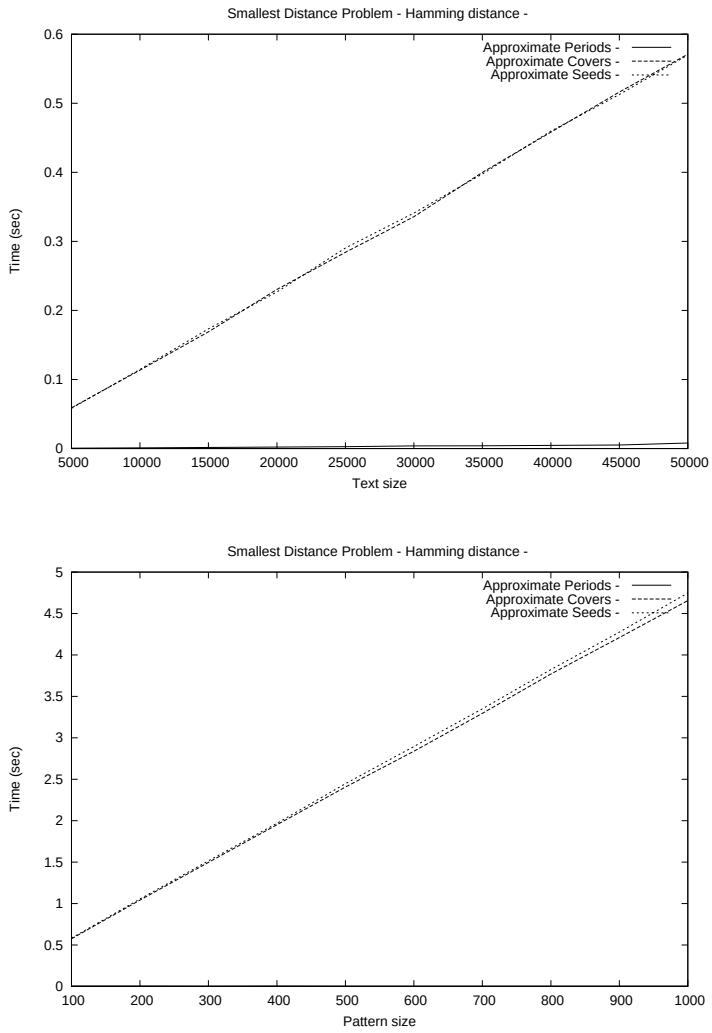


Fig. 3. The Smallest Distance Problem when Hamming Distance is used. Time consumption with respect to the text size (top) and the pattern size (bottom).

4.2 Performance

In all our tests, the main string x consists of the first n characters of the chromosome. In the case of the Smallest Distance Approximate Period/Cover/Seed we choose s, the candidate period/cover/seed, to be a random substring of the main string x.

Figure 3 presents the running times of the algorithms for the Smallest Distance Approximate Period/Cover/Seed problem when the Hamming distance is used. It is clear that, for the Hamming distance, approximate periods are computed in $O(n)$ time: the time increases linearly on size of the text and is constant (does not increase) on the size of the pattern —in fact, in the graph below (in Fig. 3), the line representing approximate periods is hardly visible

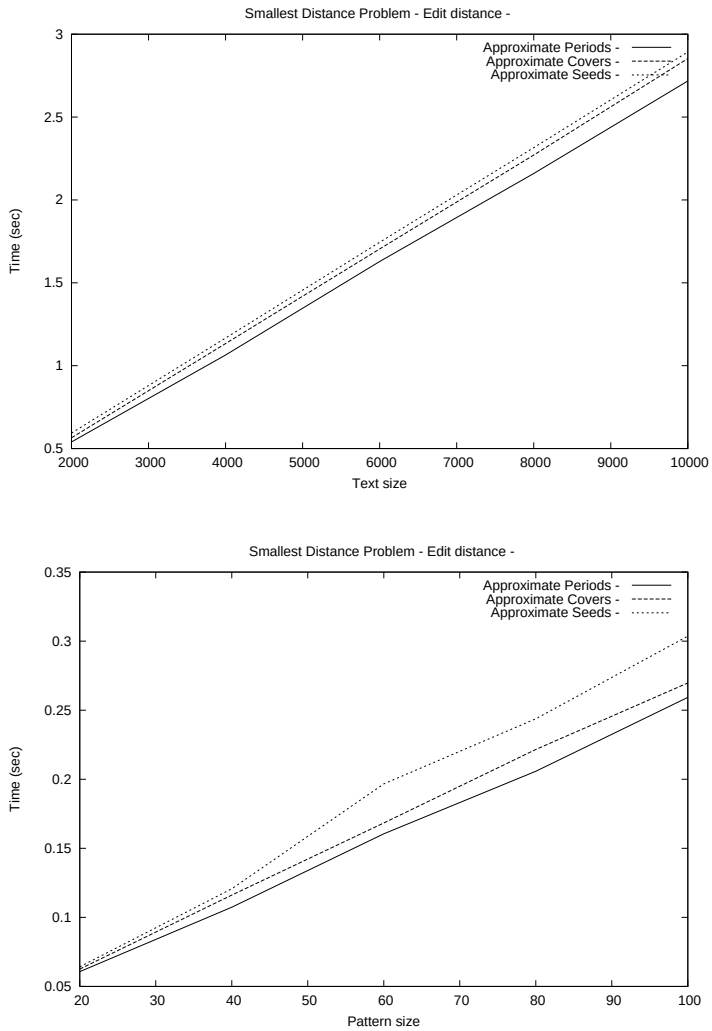


Fig. 4. The Smallest Distance Problem when Edit Distance is used. Time consumption with respect to the text size (top) and the pattern size (bottom).

because the constant is very small. Approximate covers and seeds take $O(mn)$ time: the increases linearly when either the text size or the pattern size are being increased.

Note also, that although all periods, covers, and seeds depend linearly on the size of the text, periods are computed much faster (the line representing periods in the graph above in Fig. 3 is almost invisible). This, of course was expected, since, first, for approximate covers and seeds the time also depends on m (time complexity $O(mn)$ as explained in Section 3.1), and, second, because in the first step of the algorithm, approximate periods require a single pass over the text to find the distances, while for covers and seeds more passes are necessary.

Figure 4 presents the running times of the algorithms for the Smallest Distance

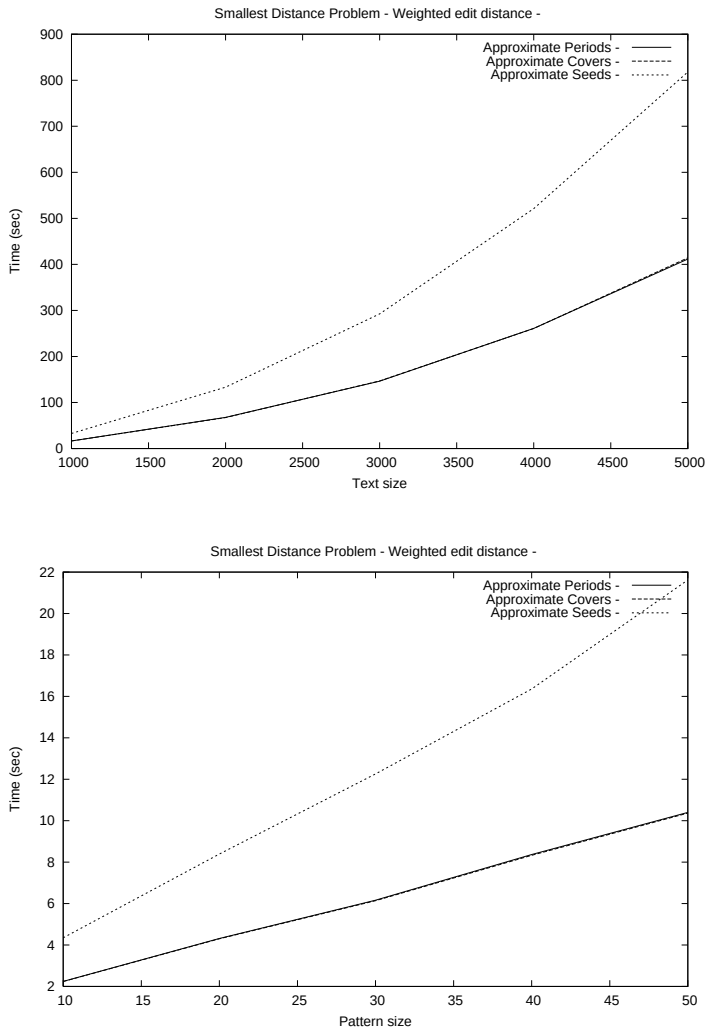


Fig. 5. The Smallest Distance Problem when Weighted Edit Distance is used. Time consumption with respect to the text size (top) and the pattern size (bottom).

Approximate Period/Cover/Seed problem when the Edit distance is used. For the edit distance, all approximate regularities take time $O(mn)$. Moreover, the constants hidden in the time complexities of approximate periods, covers, and seeds, turn out to be close to each other, and thus all of them takes almost the same amount of time, with periods being the winner (but with only a small difference from covers and seeds).

In Figure 5 the running times of the algorithms for the Smallest Distance Approximate Period/Cover/Seed problem when the Weigthed Edit distance is used, are presented. All approximate periods, covers, and seeds are computed in $O(mn^2)$ —the time increases in a quadratic manner when the text size is increased, and linearly when the pattern size is increased.

Note that, in the case of weighted edit distance, approximate periods and

covers take exactly the same amount of time, while approximate seeds are computed in almost double the time needed for the other two kinds of regularity. According to Table 4, one would expect that approximate covers are the ones to be computed faster. However, a close look at the detailed analysis in Section 3.1 reveals that this is not the case: although step 2 takes the same amount of time for all of the regularites, in step 1 the dominant term $O(mn^2)$ is doubled (due to the left extension) for approximate seeds (see Table 2).

Finally, Figure 6 presents the running times of the algorithm for the Restricted Smallest Approximate Period/Cover/Seed problem. The time complexities are $O(n^3)$, $O(n^3)$ and $O(n^4)$ for the Hamming, edit, and weighted edit distances, respectively. For all the types of distance, approximate seeds are computed slower than approximate periods and covers. Recall that the Restricted Smallest Approximate Regularity problem is being tackled by repeatedly solving the Smallest Distance Approximate Regularity problem. Thus, the invisible (in the latter algorithm) extra cost of computing the left extension for the seeds, is now being magnified and results in a slowdown (by a constant factor) of the computation of approximate seeds. It is worth noting also that, in the case of Hamming distance, approximate periods are computed significantly faster, as was expected, since the Smallest Distance Approximate Regularity problem is solved significantly faster for approximate periods, under the Hamming distance model.

References

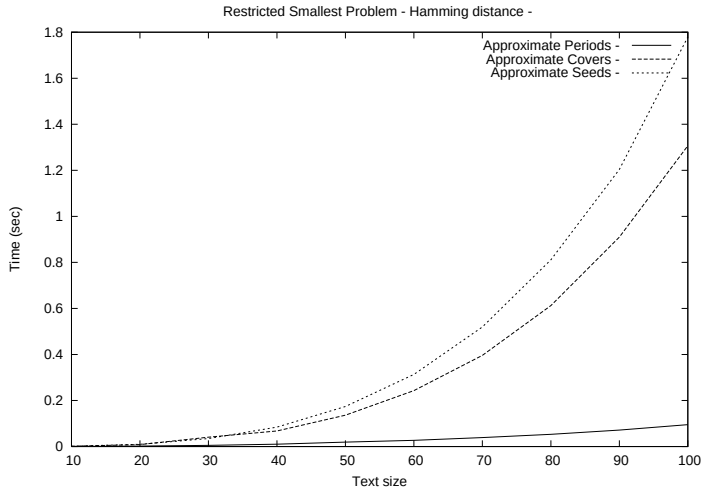
[1] M. Christodoulakis, C. S. Iliopoulos, K. Park, and J. S. Sim. Approximate seeds of strings. In *Proceedings of the Prague Stringology Conference*, pages 25–36, 2003.

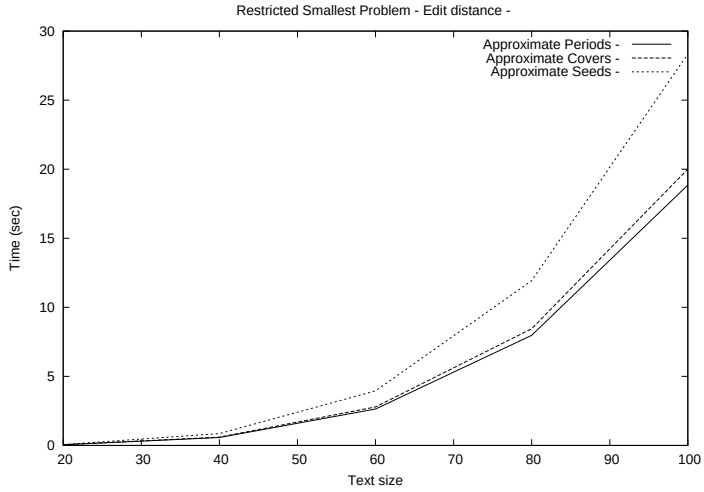
[2] M. Crochemore, C. S. Iliopoulos, and H. Yu. Algorithms for computing evolutionary chains in molecular and musical sequences. In *Proc. 9th Australasian Workshop on Combinatorial Algorithms*, pages 172–185, 1998.

[3] S. Kim and K. Park. A dynamic edit distance table. In *Proc. 11th Symp. Combinatorial Pattern Matching*, volume 1848, pages 60–68. Springer, Berlin, 2000.

[4] G. M. Landau and J. P. Schmidt. An algorithm for approximate tandem repeats. In *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*, number 684, pages 120–133, Padova, Italy, 1993. Springer-Verlag, Berlin.

[5] J. P. Schmidt. All highest scoring paths in weighted grid graphs and its application to finding all approximate repeats in strings. *SIAM Journal on Computing*, 27(4):972–992, 1998.





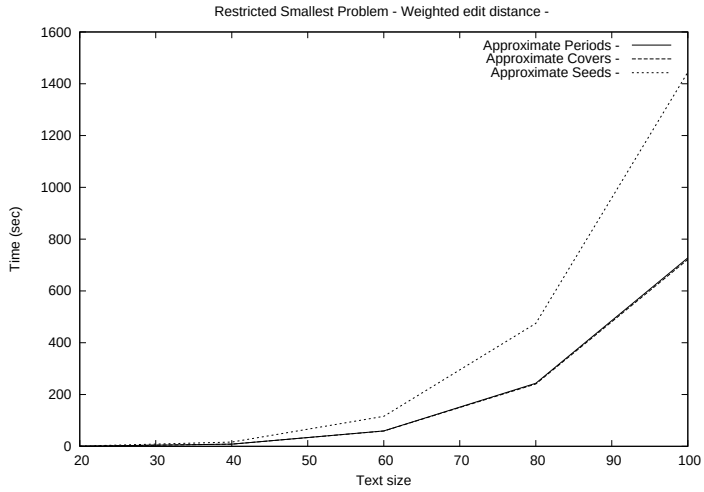


Fig. 6. Restricted Smallest Problem. Time consumption with respect to the text size, when the Hamming, the Edit, and the Weighted edit distance measures are used, respectively.

[6] J. S. Sim, C. S. Iliopoulos, K. Park, and W. F. Smyth. Approximate periods of strings. *Theoretical Computer Science*, 262:557–568, 2001.

[7] J. S. Sim, K. Park, S. Kim, and J. Lee. Finding approximate covers of strings. *Journal of Korea Information Science Society*, 29(1):16–21, 2002.