Conservative String Covering of Indeterminate Strings

Pavlos Antoniou[1], Maxime Crochemore[1], Costas S. Iliopoulos[1], Inuka Jayasekera[1], and Gad M. Landau[2]

[1] Dept. of Computer Science, King's College London, London WC2R 2LS, England, UK
[2] Department of Computer Science, University of Haifa, Mount Carmel, Haifa 31905, Israel

Abstract. We study the problem of finding local and global covers as well as seeds in conservative indeterminate strings. An indeterminate string is a sequence $T = T[1]T[2]\dots T[n]$, where $T[i] \subseteq \Sigma$ for each i, and Σ is a given alphabet of fixed size. A conservative indeterminate string, is an indeterminate string where the number of indeterminate symbols in the positions of the string, i.e. the non-solid symbols, is bounded by a constant κ. We present an algorithm for finding a conservative indeterminate pattern p in an indeterminate string t. Furthermore, we present algorithms for computing conservative covers and seeds of the string t.

1 Introduction

Covers are considered as common regularities in a string along with repetitions and periods. They are periodically repetitive. A substring w of a string x is called a cover of x if and only if x can be constructed by concatenations and superpositions of w. A seed is an extended cover in the sense of a cover of a superstring of x.

Finding the regularities present in strings is not only interesting in string algorithms but it is also useful in many applications. These applications include molecular biology, data compression and computational music analysis. Regularities in strings have been studied widely the last 20 years. There are several $O(n \log n)$- time algorithms for finding repetitions ([4],[7]), in a string x, where n is the length of x. Apostolico and Breslauer [2] gave an optimal $O(\log \log n)$-time parallel algorithm for finding all the repetitions. The preprocessing of the Knuth-Morris-Pratt algorithm [11] finds all periods of every prefix of x in linear time.

In many cases, it is desirable to relax the meaning of repetition. For instance, if we allow overlapping and concatenations of periods in a string we get the notion of *covers*. The notion of covers was introduced by Apostolico, Farach and Iliopoulos in [3], where a linear-time algorithm to test superprimitivity, was given. Moore and Smyth in [12] gave linear-time algorithms for finding all covers of a string x.

An extension of the notion of covers, is that of seeds; that is, covers of a superstring of x. The notion of seeds was introduced by Iliopoulos, Moore and Park [10] and an $O(nlogn)$-time algorithm was given for computing all seeds of x. A parallel algorithm for finding all seeds was presented by Berkman, liopoulos and Park [6], that requires $O(\log n)$ time and $O(n \log n)$ work.

In this work, we study and design algorithms for these string regularities in indeterminate strings. An indeterminate string is a sequence $T = T[1]T[2]\dots T[n]$, where $T[i] \subseteq \Sigma$ for each i, and Σ is a given alphabet of potentially large size. The simplest form of indeterminate string is one in which indeterminate positions can contain only

Pavlos Antoniou, Maxime Crochemore, Costas S. Iliopoulos, Inuka Jayasekera, Gad M. Landau: Conservative String Covering of Indeterminate Strings, pp. 108–115.
Proceedings of PSC 2008, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04145-1

a don't care letter, that is, a letter $*$ that matches any letter of the alphabet Σ on which X is defined.

In biology, usually, the number of indeterminate positions in a sequence is naturally bounded by a constant value. Otherwise, we would have a cover of length 1 with just a don't care symbol that corresponds to all the letters of the alphabet Σ. Therefore, we impose a constraint on the strings, which requires that the number of indeterminate positions in a cover c is less than the constant, that is a "conservative" cover. An example of a sequence containing indeterminate positions is shown in Figure 1 which depicts a sequence logo of an indeterminate sequence.The bottom logo is the consensus sequence derived by the 12 sequences on top of it. If we look at the logo we can see that position 1 is indeterminate as we can have $[TCAG]$ occurring, position 2 is indeterminate also having possible occurrence of $[TCA]$,position 3 is solid, non indeterminate, as in that position only A occurs.

An algorithm was described [8] for computing all occurrences of a pattern p in a text string x, but although efficient in theory, the algorithm was not useful in practice. Indeterminate string pattern matching has mainly been handled by bit mapping techniques (ShiftOr method) [5],[15]. These techniques have been used to find matches for an indeterminate pattern p in a string x [9] and the agrep utility [14] has been one of the few practical algorithms available for indeterminate pattern-matching.

In [9], the authors extended the notion of indeterminate strings by distinguishing two distinct forms of indeterminate match: "quantum" and "deterministic". Roughly speaking, a "quantum" match allows an indeterminate letter to match two or more distinct letters during a single matching process; a "determinate" match restricts each indeterminate letter to a single match[9].

In this paper, we describe algorithms for finding string regularities in constrained indeterminate strings. The next section introduces the basic definition, Section 3 describes the algorithm for conservative pattern matching. Additionally, Section 4 and Section 5 describe the algorithms for computing the covers and seeds of a string respectively.

2 Basic definitions

A *string* is a sequence of zero or more symbols from an alphabet Σ. The set of all strings over Σ is denoted by Σ^* . The length of a string x is denoted by $|x|$. The *empty* string, the string of length zero, is denoted by ϵ. The i-th symbol of a string x is denoted by $x[i]$.

A string w is a substring of x if $x = uwv$, where $u, v \in \Sigma^*$. We denote by $x[i \dots j]$ the substring of x that starts at position i and ends at position j. Conversely, x is called a *superstring* of w. A string w is a *prefix* of x if $x = wy$, for $y \in \Sigma^*$. Similarly, w is a *suffix* of x if $x = yw$, for $y \in \Sigma^*$.

We call a string w a *subsequence* of x (or x is a supersequence of w) if w is obtained by deleting zero or more symbols at any positions from x. For example, ace is a subsequence of $aabcdef$. For a given set S of strings, a string w is called a common supersequence of S if s is a supersequence of every string in S.

The string xy is the *concatenation* of the strings x and y. The concatenation of k copies of x is denoted by x^k. For two strings $x = x[1 \dots n]$ and $y = y[1 \dots m]$ such that $x[n-i+1 \dots n] = y[1 \dots i]$ for some $i \geq 1$ (that is, such that x has a suffix equal to a prefix of y), the string $x[1 \dots n]y[i+1 \dots m]$ is said to be a *superposition* of x

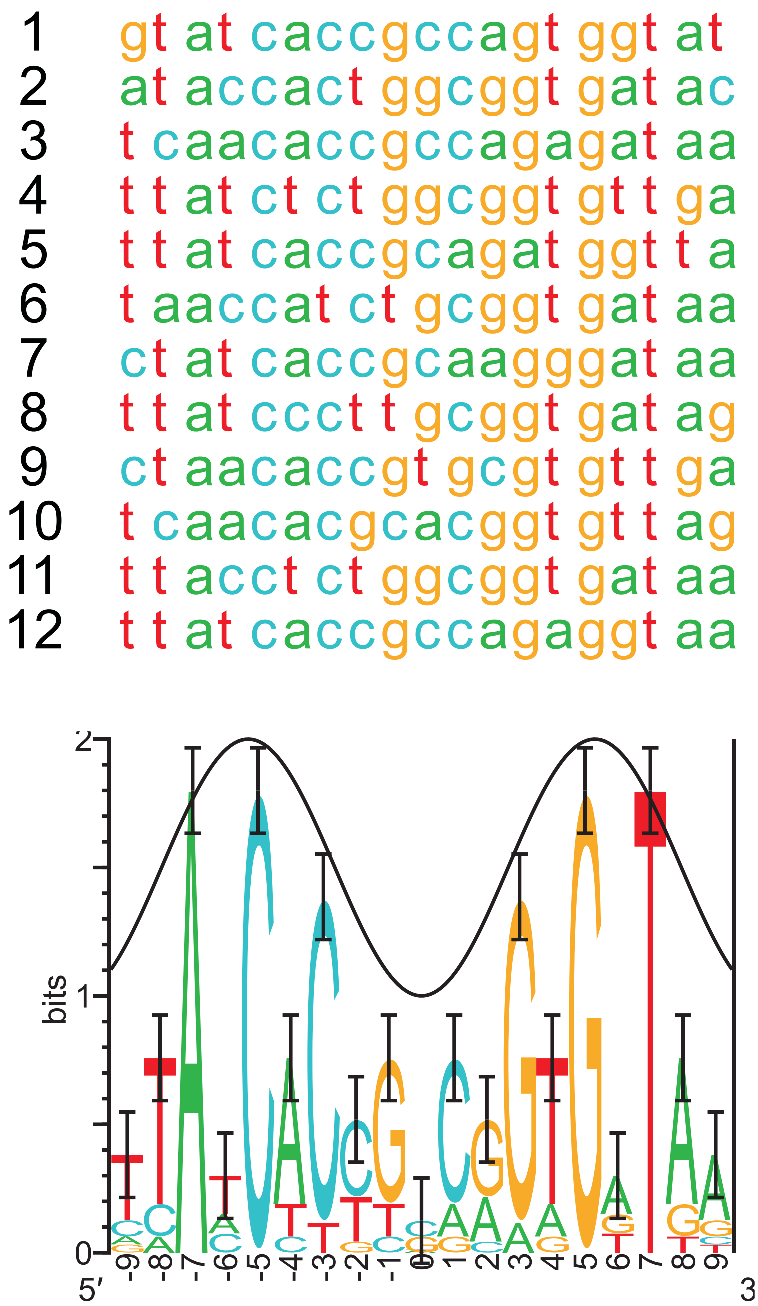


Figure 1. A sequence logo of a biological indeterminate sequence. Picture taken from [13]

and y. We also say that x overlaps with y. A substring y of x is called a *repetition* in x, if $x = uy^kv$, where u, y, v are substrings of x and $k \geq 2$, $|y| \neq 0$. For example, if $x = aababab$, then a (appearing in positions 1 and 2) and ab (appearing in positions 2, 4 and 6) are repetitions in x; in particular $a^2 = aa$ is called a *square* and $(ab)^3 = ababab$ is called a *cube*.

A non-empty substring w is called a *period* of a string x, if x can be written as $x = w^kw^r$ where $k \leq 1$ and w' is a prefix of w. The shortest period of x is called the period of x. For example, if $x = abcabcab$, then abc, $abcabc$ and the string x itself are periods of x, while abc is the period of x.

A substring w of x is called a *cover* of x, if x can be constructed by concatenating or overlapping copies of w. We also say that w *covers* x. For example, if $x = ababaaba$, then aba and x are covers of x. If x has a cover $w \neq x$, x is said to be *quasiperiodic*; otherwise, x is *superprimitive*.

A substring w of x is called a seed of x, if w covers one superstring of x (this can be any superstring of x, including x itself). For example, aba and $ababa$ are some seeds of $x = ababaab$.

An *indeterminate string* is a sequence $T = T[1]T[2]\dots T[n]$, where $T[i] \subseteq \Sigma$ for each i, and Σ is a given alphabet of potentially large size. When a position of the string is indeterminate, and it can match more than one element from the alphabet Σ, we say that this position has *non-solid* symbol. If in a position we have only one element of the alphabet Σ present, then we refer to this symbol as *solid*. A *conservative* indeterminate string, is an indeterminate string where its number of indeterminate symbols is bounded by a constant k.

Building the Aho-Corasick Automaton [1]. The Aho-Corasick Automaton for a given finite set P of patterns is a Deterministic Finite Automaton G accepting the sets of all words containing a word of P as a suffix.

$G = (Q, \Sigma, g, f, q_0, F)$, where function Q is the set of states, Σ is the alphabet, g is the forward transition, f is the failure link i.e. $f(q_i) = q_j$, if and only if S_j is the longest suffix of S_i that is also a prefix of any pattern, q_0 is the initial state and F is the set of final (terminal) states [1]. The construction of the AC automaton can be done in $O(d)$-time and space complexity, where d is the size of the dictionary, i.e. the sum of the lengths of the patterns which the AC automata will match.

3 Finding constrained pattern p in indeterminate string T

As a building step, here, we study the constrained pattern matching problem on indeterminate strings. The problem of constrained indeterminate pattern matching is defined as follows:

INPUT: We are given a pattern p of length m with at most κ non solid symbols, where κ is a constant. We are given an indeterminate string T, the text, of length n.

QUERY: Find all the occurrences of the pattern p in the text T, i.e find the positions in T where the intersection of the pattern and the text is non-empty.

Example 1. We consider a pattern, $p = A[CG]TA[AG]$ and text, $T = GA[CG][CT]A$ $G[AT]A[AG][CT][AT]AG$. Figure 2 shows the result of searching for p in t. It can be seen from the figure that p occurs in t starting at positions 2, 5, 8 and 9.

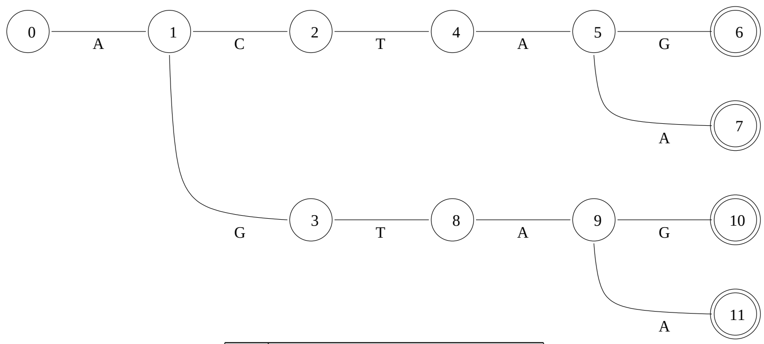
i	0	1	2	3	4	5	6	7	8	9	10	11	12
t	G	A	[CG]	[CT]	A	G	[AT]	A	[AG]	[CT]	[AT]	A	G
Matches		A	[CG]	T	A	[AG]							
					A	[CG]	T	A	[AG]				
								A	[CG]	T	A	[AG]	
									A	[CG]	T	A	[AG]

Figure 2. Pattern matching with p and t

The algorithm works in two steps:

Step 1

Let the pattern p be $p = P_1P_2 \dots P_m$. We built the Aho-Corasick automaton for the dictionary of the prefixes of the pattern $D = \{\pi_1\pi_2 \dots \pi_m, \forall \pi_i \in P_i, 1 \leq i \leq m\}$. Note that $|D| = \prod_{i=1}^{m} |P_i| < 2^\kappa$ as there are at most κ non-solid symbols.



i	0	1	2	3	4	5	6	7	8	9	10	11
f(i)	0	0	0	0	0	1	3	1	0	1	3	1

Figure 3. Aho-Corasick automata and its failure function for p

Step 2

Assume that we have processed $T[1, i]$. At this point we have a set, P, of prefixes of the strings in the dictionary in the Aho-Corasick automaton. We will now perform iteration $i + 1$. For each symbol τ occurring at $T[i + 1]$, we try to extend each prefix in P by that symbol τ, or we follow its failure link provided by the Aho-Corasick automaton. Figures 3 and 4 present a part of the matching process for the previous example.

Note that $|P|$ is bounded by the maximum number of possible prefixes, which in turn is bounded by the size of the automaton, therefore this is constant. Thus, this method is linear.

i	0	1	2	3	4	5	6	
t	G	A	[CG]	[CT]	A	G	[AT]	...
P	0	{1}	{2,3}	{4,8}	{5,9}	{6, 10}	{8}	...

Figure 4. Matches of prefixes of P in text t

4 Computing λ-conservative covers of indeterminate strings

Here, we study another string regularity, conservative covering of an indeterminate string with a fixed length cover. The λ-conservative cover problem is defined as follows:

Input: We are given a conservative indeterminate string t, of length n, a constant κ, which is the maximum number of non-solid symbols allowed in a cover and an integer λ, which is the length of the cover.

Query: Is there a conservative cover, c, of t of length λ?

STEP 1
We consider the prefix, $\hat{T}$, of t of length λ,

$$\hat{T} = T_1 \dots T_\lambda$$

and the suffix, $\tilde{T}$ of t of length λ,

$$\tilde{T} = T_{n-\lambda+1}, \dots T_n$$

We build the Aho-Corasick automaton for the dictionary

$$D = \{t_1 \dots t_\lambda \mid \forall\ t_i \in T_i \cap T_{i+n-\lambda},\ 1 \leq i \leq \lambda\}$$

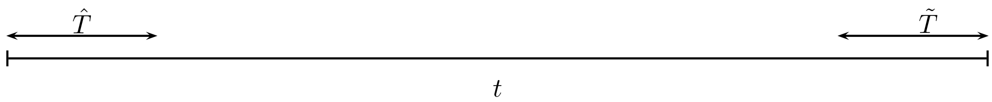


Figure 5. The cover, c, covers the beginning and the end of T. Thus $\hat{T}$ and $\tilde{T}$ provide the set of potential candidates.

STEP 2
For each $d \in D$ we find all of its occurrences in T, parsing the text T through the Aho-Corasick Automaton built in STEP 1. If a word d occurs at position i then we set a flag $L(i) = true$. If the distance $|i - j|$ of any two consecutive flags is less than λ, then we have a cover

$$C_1 C_2 \dots C_\lambda, \text{where}$$

$$C_i = \{d_i, \text{ is the } i - th \text{ letter of every word in } D,\ 1 \leq i \leq \lambda\}$$

The overall complexity of the above two steps is linear.

5 Computing λ-conservative seeds of indeterminate strings

Here we study yet another regularity, covering an indeterminate string with seed of a given length. The λ-constrained seed problem is defined as follows:

INPUT: We are given an indeterminate string t, of length n, a constant κ, which is the maximum number of non-solid symbols allowed in a seed and an integer λ, which is the length of the seed.
QUERY: Is there a conservative seed, s, of t of length λ?
STEP 1
The first occurrence of the seed can be in any of the positions $\{1 \dots \lambda\}$. Thus we consider the following strings of length λ :

$$L_1 = \{T[1..\lambda], T[2..\lambda+1], \dots T[\lambda..2\lambda-1]\}$$

and all the suffixes of string t of length λ :

$$L_2 = \{T[n-\lambda..n], T[n-\lambda-1..n-1] \dots T[n-2\lambda-1]\}$$

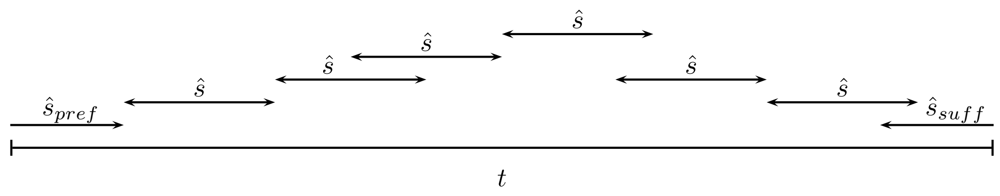


Figure 6. Above, $\hat{s}$ is a seed of the string t, where each $\hat{s}$ contains at most κ non-solid symbols and is of length λ. Also, $\hat{s}_{pref}$ and $\hat{s}_{suff}$ are a prefix and suffix of $\hat{s}$ respectively.

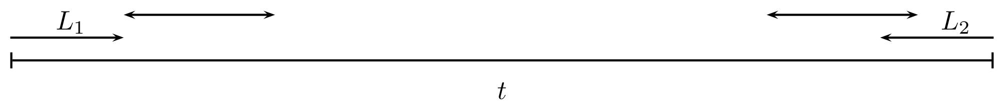


Figure 7. The positions of candidate seeds from lists L_1 and L_2 are shown above.

We build the Aho-Corasick automaton for the dictionary

$$D = \{t_{i_1} \ldots t_{i_\lambda} \mid \forall t_{i_j}, \text{where } \mathrm{t_{i_j}} \text{ is the } \mathrm{j-th} \text{ symbol of } \mathrm{T \in L_1 \cup L_2}\}.$$

Step 2
For each $d \in D$ we find all of its occurrences in T, parsing the text T through the Aho-Corasick Automaton built in Step 1. If a word d occurs at position i then we set a flag $L_d(i) = true$. If the distance $|i - j|$ of any two consecutive flags in L_d is less than λ, then we d is a candidate for a seed. Let i_1 and i_2 be the first and last occurrences of d in T. We check if $T[1, i_1]$ is a suffix of d and if $T[i_2, n]$ is a prefix of d, if that is the case then d is a suffix. The overall complexity is $O(\lambda n)$.

6 Conclusion

In conclusion, we have shown $O(n)$ algorithms for finding the smallest conservative cover, λ-conservative local covers. We have also presented a $O(\lambda n)$ algorithm for finding the λ-conservative seeds of a string. All the algorithms which we have used are easily adaptable to allow the bit-matching technique to be used, in order to allow efficient implementations.

References

1. A. V. Aho and M. J. Corasick: *Efficient string matching: an aid to bibliographic search.* Commun. ACM, 18(6) 1975, pp. 333–340.
2. A. Apostolico and D. Breslauer: *An optimal o(loglog n)-time parallel algorithm for detecting all squares in a string.* SIAM J. Comput., 25(6) 1996, pp. 1318–1331.
3. A. Apostolico, M. Farach, and C. S. Iliopoulos: *Optimal superprimitivity testing for strings.* Information Processing Letters, 39 1991, pp. 17–20.
4. A. Apostolico and F. P. Preparata: *Optimal off-line detection of repetitions in a string.* Theor. Comput. Sci., 22 1983, pp. 297–315.
5. R. Baeza-Yates and G. H. Gonnet: *A new approach to text searching.* Commun. ACM, 35(10) 1992, pp. 74–82.

6. O. Berkman, C. S. Iliopoulos, and K. Park: *The subtree max gap problem with application to parallel string covering*. Information and Computation, 123(1) 1995, pp. 127–137.
7. M. Crochemore: *An optimal algorithm for computing the repetitions in a word*. Inf. Process. Lett., 12(5) 1981, pp. 244–250.
8. M. J. Fischer and M. S. Paterson: *String-matching and other products*, tech. rep., Massachusetts Institute of Technology, Cambridge, MA, USA, 1974.
9. J. Holub, W. F. Smyth, and S. Wang: *Fast pattern-matching on indeterminate strings*. J. of Discrete Algorithms, 6(1) 2008, pp. 37–50.
10. C. S. Iliopoulos, D. W. G. Moore, and K. Park: *Covering a string*, in Proceedings of the 4-th Symposium on Combinatorial Pattern Matching, vol. 684 of Lecture Notes in Computer Science, Berlin, 1993, Springer-Verlag, pp. 54–62.
11. D. E. Knuth, J. Morris, and V. R. Pratt: *Fast pattern matching in strings*. SIAM Journal of Computing, 6(2) 1977, pp. 323–350.
12. D. Moore and W. F. Smyth: *An optimal algorithm to compute all the covers of a string*. Inf. Process. Lett., 50(5) 1994, pp. 239–246.
13. M. C. Shaner, I. M. Blair, and T. D. Schneider: *Sequence logos: A powerful, yet simple, tool*, in Proceedings of the Twenty-Sixth Annual Hawaii International Conference on System Sciences, Volume 1: Architecture and Biotechnology Computing, T. N. Mudge, V. Milutinovic, and L. Hunter, eds., IEEE Computer Society Press, 1993, pp. 813–821.
14. S. Wu and U. Manber: *Agrep – a fast approximate pattern-matching tool*, in Proceedings USENIX Winter 1992 Technical Conference, San Francisco, CA, 1992, pp. 153–162.
15. S. Wu and U. Manber: *Fast text searching: allowing errors*. Commun. ACM, 35(10) 1992, pp. 83–91.