1999

Optimal Discovery of Subword Associations in Strings

Alberto Apostolico

Giorgio Satta

Report Number:
99-042

OPTIMAL DISCOVERY OF SUBWORD ASSOCIATIONS IN STRINGS

Alberto Apostolico
Giorgio Satta

CSD TR #99-042
November 1999

Optimal Discovery of Subword Associations in Strings

Alberto Apostolico* Giorgio Satta†
Purdue University and Università di Padova Università di Padova

(May 1999 - Revised Oct. 1999)
Purdue CS TR 99-042

Abstract

Given a textstring x of n symbols and an integer constant d, we consider the problem of finding, for any pair (y, z) of subwords of x the number of times that y and z occur in tandem (i.e., with no intermediate occurrence of either one of them) within a distance of d symbols of x. Although in principle there might be n^4 distinct subword pairs in x, we show that it suffices to consider a family of only n^2 such pairs, with the property that for any neglected pair (w', z'), there is a corresponding pair (w, z) contained in our family and such that: (i) w' is a prefix of w and z' is a prefix of z, and (ii) the tandem index of (w', z') equals that of (w, z). We show that an algorithm for the construction of the table of all such tandem indices can be built to run in optimal $O(n^2)$ time and space.

Keywords: Pattern Matching, String Searching, Subword Tree, Substring Statistics, Tandem Index, Association Rule.

---

*Department of Computer Sciences, Purdue University, Computer Sciences Building, West Lafayette, IN 47907, USA and Dipartimento di Elettronica e Informatica, Università di Padova, Padova, Italy. axa@cs.purdue.edu. Work supported in part by NSF Grant CCR-9700276, and by the Italian Ministry of Research.

†Dipartimento di Elettronica e Informatica, Università di Padova, Padova, Italy. satta@dei.unipd.it

1 Introduction

The problem of characterizing and detecting unusual events such as recurrent subsequences and other streams or over/under-represented words in sequences arises ubiquitously in diverse applications and is the subject of much study and interest in fields ranging from Computer and Network Security to Data Mining, from Speech and Natural Language Processing to Computational Molecular Biology. It also gives rise to interesting modeling and algorithmic questions, some of which have displayed independent interest.

Among the problems in this class we find, for instance, the detection of all squares or palindromes in a string, for which optimal $O(n \log n)$ algorithms have long been known (refer, e.g., [3, 5] and references therein). It is not difficult to extend those treatments to germane problems such as the discovery of pairs of occurrences, within a given distance, of a same string, or of a string and its reverse, and os on.

In this paper, we concentrate on the problem of detecting repetitive phenomena that consist of unusually frequent *tandem* occurrences, within a preassigned distance in a string, of two distinct but otherwise unspecified substrings. By the two strings occurring in tandem, we mean that there is no intermediate occurrence of either one in between.

Specifically, we consider the following problem. Let x be a string of n symbols over some alphabet Σ and d some fixed non-negative integer. For any pair (y, z) of subwords of x, their *tandem index* $I(y, z)$ relative to x is the number of times that z has a closest occurrence in x within a distance of d from a corresponding, closest occurrence of y. We are interested in finding pairs of subwords with surprisingly high tandem index.

The problem can be cast in the emerging contexts of *data mining* and *information extraction*. As is well known, while traditional data base queries aim at retrieving records based on their isolated contents, these contexts focus on the identification of patterns occurring across records, and aim at the retrieval of information based on the discovery of interesting rules present in large collection of data. Central to these developments is the notion of *association rule*, which is an expression of the form $S_1 \rightarrow S_2$ where S_1 and S_2 are sets of data attributes endowed with sufficient *confidence* and *support*. Sufficient support for a rule is achieved if the number of records whose attributes include $S_1 \cup S_2$ is at least equal to some pre-set minimum value. Confidence is measured instead in terms of the ratio of records having $S_1 \cup S_2$ over those

having S_1, and is considered sufficient if this ratio meets or exceeds a pre-set minimum. Clearly, a statistic of the number of records endowed with the given attributes must be computed as a preliminary step, and this is often a bottleneck for the process of information extraction. We refer to [1] and [7] for a broader discussion of these concepts.

Back to our problem, we observe that, in principle, there might be n^4 distinct pairings of subwords of in x. We show, however, that it suffices to restrict attention to a family containing only only n^2 pairs, after which for any neglected pair (w', z'), there is a pair (y, z) in the family such that: (i) w' is a prefix of w and z' is a prefix of z, and (ii) the tandem index of (w', z') equals that of (w, z). We show that an algorithm for the construction of the table of all such tandem indices can be built to run in optimal $O(n^2)$ time and space for a string x of n symbols.

2 Preliminaries

We begin by recalling an important "right-context" property, which is conveniently adapted from [4]. Given two words x and y, the *start-set* of y in x is the set of *occurences* of y in x, i.e., $pos_x(y) = \{i : y = x_i...x_j\}$ for some i and j, $1 \leq i \leq j \leq n$. Two strings y and z are equivalent on x if $pos_x(y) = pos_x(z)$. The equivalence relation instituted in this way is denoted by $\equiv_x$ and partitions the set of all strings over Σ into equivalence classes. Thus, $[y]$ is the set of all strings that have occurences in x beginning at the same set of positions as y. In the example of the string $abaababaabaababaababa$, for instance, $\{ab, aba\}$ forms one such equivalence class and so does $\{abaa, abaab, abaaba\}$. Recall that the *index* of an equivalence relation is the number of equivalence classes in it.

Fact 2.1 *The index k of the equivalence relation $\equiv_x$ obeys $k < 2n$.*

Fact 2.1 suggests that we might only need to look among $O(n^2)$ substring pairs of a string of n symbols in order to find unusually frequent pairs. The following considerations show that this statement can be made precise giving an indirect proof of it. We recall the notion of the *suffix tree* T_x associated with x. This is essentially a compact trie with $n + 1$ leaves and at most n internal nodes that collects all suffixes of $x\$$, where $\$$ a symbol not in Σ. We assume familiarity of the reader with the structure and its clever $O(n \log |\Sigma|)$

time and linear space constructions such as in [6, 9, 10]. The word ending precisely at vertex α of T_x is denoted by $w(\alpha)$. The vertex α is called the *proper locus* of $w(\alpha)$. The *locus* of w is the unique vertex of T_x such that w is a prefix of $w(\alpha)$ and $w(\text{FATHER}(\alpha))$ is a proper prefix of w.

Having built the tree, some simple additional manipulations make it possible to count and locate the distinct (possibly overlapping) instances of any pattern w in x in $O(|w|)$ steps. For instance, listing all occurrences of w in x is done in time proportional to $|w|$ plus the total number of such occurrences, by reaching the locus of w and then visiting the subtree of T_x rooted at that locus. Alternatively, a trivial bottom-up computation on T_x can weigh each node of T_x with the number of leaves in the subtree rooted at that node. This weighted version serves then as a statistical index for x, in the sense that, for any w, we can find the frequency of w in x in $O(|w|)$ time. Note that the counter associated with the locus of a string reports its correct frequency even when the string terminates in the middle of an arc. This is, indeed, nothing but a re-statement and a proof of Fact 2.1. From now on, we assume that a tree with weighted nodes has been produced and is available for our constructions.

3 Algorithms

For simplicity of exposition we assume that Σ is the binary alphabet $\Sigma = \{a, b\}$, but it should be clear that this assumption can be removed without penalty on our constructions.

In view of Fact 2.1 we may restrict attention to the $O(n)$ equivalence classes of $\equiv_x$ represented by strings that end precisely at the internal nodes and leaves of T_x. Even these latter, each being formed by some consecutive prefixes of a singleton string, may be neglected as uninteresting.

As a warmup for the discussion, consider the problem of computing the following relaxed notion of a tandem index, which we denote by $\bar{I}(y, z)$ and consists of the number of instances of z that fall within d positions of a closest occurrence of y. The difference with respect to $I(y, z)$ is that we still forbid intervening occurrences of y, but now allow possibly intervening occurrences of z.

It is easy to compute $\bar{I}(y, z)$ for a fixed word y and all z's with a proper locus in T_x in overall linear time. This is done as follows. We can assume

that, as a trivial by-product of the construction of T_x, there is access from each leaf of T_x to the corresponding position of x and *vice versa*. Now, let ν be the node of T_x such that $w(\nu) = y$. Visit the subtree of T_x rooted at ν and place a special mark on the positions of x that correspond to leaves in this subtree. This marks all the starting positions of occurrences of y in x. Next, scan the positions of x in ascending order: tag those positions that fall within d positions of an occurrence of y, and assign a weight of 1 to every leaf that corresponds to such a position. Visit now T_x bottom up, and assign to each node a weight equal to the sum of the weights of its children. Clearly, this accomplishes computation of $\bar{I}(y, z)$ for all z's. Iterating the process for all y's fills the table of $\bar{I}(y, z)$ values for all pairs of words with a proper locus in T_x, and this takes (optimal) $O(n^2)$ time and space.

Consider now computing the $I(y, z)$ values for a given y and all strings z with a proper locus in T_x. If, in the bottom up computation above, we wanted to compute $I(y, z)$ instead of $\bar{I}(y, z)$, then we should prevent the weighting process from counting more than one occurrence of z within the d-range of each closest occurrence of y.

Let $\mathcal{L}$ be the sorted list of positions of x, with occurrences of y suitably marked and the positions falling within a distance of d from such occurrences tagged as above. Define a *clump* in $\mathcal{L}$ as a maximal run of tagged positions falling between two consecutive occurrences of y (or following the last occurence of y). Let us say further that a clump is *represented* at a node μ of T_x if at least one element of the clump is found in the subtree of T_x rooted at μ. Clearly, we want each clump to contribute precisely one unit weight at all nodes where the clump itself is represented.

It is possible to achieve this by the following preprocessing of T_x. With k the total number of clumps, initialize k empty *clump lists*. Visit the (leaves of the) tree from left to right. For each leaf encountered, check on its corresponding entry in $\mathcal{L}$ whether this leaf belongs to a clump. In this case, assign to the leaf a *rank* equal to the ordinal number of arrival of the leaf in its clump list according to the visit. At the end, the concatenation of the clump lists constitutes a sorted list of clumps such that leaves within each clump are met in order of ascending rank. This means that scanning each clump list now meets the leaves in the clump in the same order as they would be encountered when visiting T_x's yield from left to right. At this point we invoke as a subroutine the following well known Lowest Common Ancestor (l.c.a.) Algorithm (see, e.g., [8]): given a tree T with n leaves, it is possible

to preprocess T in time linear in the number of nodes and in such a way that, after preprocessing, for any two leaves i and j it is possible to give in constant time the lowest common ancestor of i and j.

For a given y, we pre-process all clumps in succession as follows. Singleton clumps are not touched. With reference now to the generic non-singleton clump, consider its leaves in order of ascending rank. For each leaf, find its l.c.a.'s relative to the leaf and its successor and give the weight of this node a -1 handicap. At the end of the process, weigh the tree by the bottom up weighting procedure as before.

We claim that the weights thus assigned to any node μ represent $I(y, w(\mu))$. This is seen by induction on the sparsification $\bar{T}$ of T_x intercepted by the leaves of some arbitrary clump and the associated l.c.a. nodes. The assertion is true on any deepest internal node of $\bar{T}$. In fact, since T_x is binary then so is $\bar{T}$. Considering any deepest internal node of $\bar{T}$, the leftmost one of its two leaves will have caused the algorithm to give the node a handicap of -1. This would combine with the weight of 2 resulting from the bottom up weighting to yield a weight of 1. Assume now the assertion true for all descendants of a node μ of $\bar{T}$, we have that the two childern α and β of μ will be assigned a weight of 1 in the bottom up process. Considering the rightmost leaf of the subtree rooted at α and the leftmost one in the subtree rooted at β, we have that μ, their l.c.a., has been given a handicap of -1 during pre-processing of the clump. Therefore, the sum of weights at μ will again be 1. The following theorem summarizes our discussion.

Theorem 3.1 *Given a string x of n symbols, the tandem indices for all pairs of subwords of x can be computed in $O(n^2)$ time and space.*

References

[1] Agrawal, R., T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD*, pp.207–216, Washington DC May 1993.

[2] Aho, A.V., J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass. (1974).

[3] Apostolico, A., and Z. Galil (Eds.), *Pattern Matching Algorithms*, Oxford University Press, New York (1997).

[4] Blumer, A., J. Blumer, A. Ehrenfeucht, D. Haussler, M.T. Chen and J. Seiferas, The Smallest Automaton Recognizing the Subwords of a Text, *Theoretical Computer Science* , **40**, 31-55 (1985).

[5] Crochemore, M., and W. Rytter, *Text Algorithms*, Oxford University Press, New York (1994).

[6] McCreight, E.M., A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, April 1976.

[7] Piatesky-Shapiro, G. and W.J. Frawley, Eds., *Knowledge Discovery in Databases*. AAAI Press/MIT Press, 1991.

[8] Schieber, B. and U. Vishkin, On finding lowest common ancestors: simplifications and parallelizations. *SIAM Journal on Computing* **17**:1253–1262, 1988.

[9] Ukkonen, E., On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[10] Weiner, P., Linear pattern matching algorithm. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, Washington, DC, 1973.