Discrete Applied Mathematics 200 (2016) 123–152

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Mining approximate patterns with frequent locally optimal occurrences[☆]

Atsuyoshi Nakamura[a,*], Ichigaku Takigawa[a], Hisashi Tosaka[b], Mineichi Kudo[a], Hiroshi Mamitsuka[c]

[a] Hokkaido University, Kita 14, Nishi 9, Kita-ku, Sapporo 060-0814, Japan
[b] NS Solutions Corporation, Tokyo, Japan
[c] Institute for Chemical Research, Kyoto University, Uji, Kyoto 611-0011, Japan

ARTICLE INFO

Article history:
Received 18 February 2014
Received in revised form 29 June 2015
Accepted 3 July 2015
Available online 30 July 2015

Keywords:
Alignment
Frequent pattern mining
String
Ordered tree
DNA

ABSTRACT

We consider a frequent approximate pattern mining problem, in which interspersed repetitive regions are extracted from a given string. That is, we enumerate substrings that frequently match substrings of a given string locally and optimally. For this problem, we propose a new algorithm, in which candidate patterns are generated without duplication using the suffix tree of a given string. We further define a k-gap-constrained setting, in which the number of gaps in the alignment between a pattern and an occurrence is limited to at most k. Under this setting, we present memory-efficient algorithms, particularly a candidate-based version, which runs fast enough even over human chromosome sequences with more than 10 million nucleotides. We note that our problem and algorithms for strings can be directly extended to ordered labeled trees. In our experiments we used both randomly synthesized strings, in which corrupted similar substrings are embedded, and real data of human chromosome. The synthetic data experiments show that our proposed approach extracted embedded patterns correctly and time-efficiently. In real data experiments, we examined the centers of 100 clusters computed after grouping the patterns obtained by our k-gap-constrained versions ($k = 0$, 1 and 2) and the results revealed that the regions of their occurrences coincided with around a half of the regions automatically annotated as Alu sequences by a manually curated repeat sequence database.

1. Introduction

A main study area in data mining is frequent pattern mining, in which major target objects are strings, such as texts, DNA sequences, sequences of musical notes and (discretized) time series data, which repeatedly appear and can be analyzed extensively. For example, chromosomes are strings with more than 100 million letters, containing similar substrings with length of more than 100 letters.

Frequent exact substring patterns, namely substrings which frequently appear exactly as it is, can be enumerated efficiently [3], while in real data such restricted patterns are too strict to find useful patterns. Flexibility can be then incorporated into the exact patterns by the following two manners: (1) removing 'contiguous' restriction, by matching with

[☆] The preliminary version of this paper has appeared in the Proceedings of the 2012 IIAI International Conference on Advanced Applied Informatics (IIAIAAI), pp. 54–59, under the title "Frequent Approximate Substring Pattern Mining Using Locally Optimal Occurrence Counting".

[*] Corresponding author. Tel.: +81 11 706 6806; fax: +81 11 706 7832.
E-mail address: atsu@main.ist.hokudai.ac.jp (A. Nakamura).

http://dx.doi.org/10.1016/j.dam.2015.07.002
0166-218X/© 2015 Elsevier B.V. All rights reserved.

gaps. Parts of a string that can be non-contiguous are called subsequences, and mining frequent subsequences has been studied actively as an extension of itemset mining [1]. (2) Removing 'exact' restriction, by matching approximate patterns, meaning that strings similar to a pattern string by some measure can be occurrences of the pattern. Typical examples of such a measure are the Hamming distance [35] and the edit distance.

We consider frequent substring pattern mining by using the above second approach. We note that the problem setting here is to find patterns frequently appearing in a single string instead of a set of multiple strings. Real applications of this setting include finding impressive parts from a musical piece by mining frequent contiguous subsequences in a musical note sequence, and finding repeated subsequences, so-called retrotransposons, from genome sequences. In both applications, we need to extract not only frequent patterns but also their occurrences.

Under our setting, the Hamming distance-based and edit distance-based approaches have been considered. The Hamming distance however is changed easily by even one insertion or deletion, resulting in that strings with different lengths cannot be similar. The edit distance can be changed simply by one when one insertion or deletion is done, meaning that this distance is more suitable. On the other hand the edit distance also has a problem: the same part may be counted many times because the edit distance is kept almost similar even when boundary positions of the part are slightly changed. We note that this is an intrinsic issue of mining from a given single string, while this would not be an issue for mining from a set of strings, where we can just count if each string has a pattern or not.

To overcome this problem of the edit distance, for each pattern, we consider a *locally optimal occurrence*, which is the locally optimal substring among substrings of a given string [10], when comparing with a pattern. More specifically, a locally optimal occurrence of a pattern has boundaries, which are optimized in the sense that boundaries are the most fitted to the pattern in alignment score when the other side of the boundaries is fixed. We then count locally optimal occurrences only. That is, we count only the occurrences with optimal boundaries among all occurrences with different boundaries. We reasonably restrict patterns to substrings in a given string. We can use the suffix tree of a given string, by which all substrings can be generated without duplication, and approximate patterns with frequent locally optimal occurrences can be enumerated in $O(n^3)$ time and $O(n^3)$ space for a given string with length n. All locally optimal occurrences can be calculated by running a local alignment algorithm twice, as forward and backward directions [23]. Thus our algorithm, which we call ESFLOO(Enumerator of subStrings with Frequent Locally Optimal Occurrences), is a harmonic combination of this algorithm and pattern generation using a suffix tree.

Algorithm ESFLOO can be applied to a string with 100,000 letters, while it is very slow and spends huge memory for a string with more than one million letters, according to our experiments. This algorithm needs $O(n^3)$ space to keep all candidate occurrences generated in the forward stages. The space can be reduced if the local optimality can be checked in both the forward and backward directions simultaneously. This is possible if the total number of gaps in an alignment is constrained to be a small number, say k. We thus consider a gap-constraint version of ESFLOO, which we call ESFLOO-kG, which runs in $O(k^2n^3)$ time and (k^2n^2) space. Algorithm ESFLOO-kG computes the local alignment scores between a whole pattern and substrings of a given string by filling the entries of all columns in the score table for dynamic programming. For longer patterns, however, the number of occurrence candidates is smaller, by which computation time for non-candidate substrings is larger and useless. We thus developed a candidate-based version of ESFLOO-kG, which we call ESFLOO-kG.C. Algorithm ESFLOO-kG.C keeps, for each candidate, an alignment score table with the width of $(2k + 1)$, and computes its all entries. This algorithm needs to compute one entry value more than once when entries are overlapped, while this computational redundancy can be compensated by the computational speed-up obtained by avoiding non-candidates, especially when enumerating long frequent substring patterns from a long string. We note that our algorithms can be extended to those for labeled ordered trees, where labeled ordered tress can be a hierarchical set of strings, regarding each bottom-up subtree as a letter.

First we examined the performance of our algorithms using synthetic data, where we generated ten random strings, in each of which one randomly generated pattern (with 100 letters) and 99 strings similar to it, which were generated by random editing of the pattern, were embedded. We checked whether the embedded pattern strings can be extracted as the most frequent pattern, comparing with two other methods: PS (Positive score), which simply counts the number of right end positions of substrings that have positive alignment scores with a pattern, and PPS (Positive Prefix Score), which counts the number of left end positions of substrings that satisfy the condition that any non-null prefix of them has a positive alignment score with some prefix of a pattern. Out of ten embedded patterns, ESFLOO extracted seven and ESFLOO-kG ($k = 3, 4, 5$) could extract all ten, while both PS and PPS could not extract any pattern. We then checked space efficiency of ESFLOO-1G and ESFLOO-1G.C, showing that the empirical space complexity was $O(n^{1.14})$ for ESFLOO-1G and $O(n^{0.877})$ for ESFLOO-1G.C, being smaller than $O(n^{2.07})$ for ESFLOO by a factor of more than $\Theta(n)$. We further confirmed that the empirical time complexity of ESFLOO-1G.C is also smaller than that of ESFLOO by a factor of $\Theta(n^{0.32})$. Also ESFLOO-1G.C was significantly faster than PS and PPS.

Secondly we applied algorithms ESFLOO-kG.C ($k = 0, 1, 2$) to real data of human chromosome 21 with 47 million nucleotides (35.1 million nucleotides are already sequenced). We first run ESFLOO-kG.C ($k = 0, 1, 2$) to enumerate substrings under the condition of at least 30 occurrences and at least 100 nucleotides, resulting more than 50 million substrings for each of k. (For this task, ESFLOO-kG.C of $k = 0$, 1 and 2 needed 43 min, 9 h 17 min and around 46 h, respectively, implying that ESFLOO, PS and PPS will need more than one year.) We then selected substrings, being consistent with 0.5-tolerance closed frequent patterns [6,26], and run normalized spectral clustering over them to obtain 100 cluster centers. For the 100 patterns, we further identified all locally optimal occurrences and compared them with the repeat regions