2008－BIO－13 (15)
2008／6／27

近傍ハッシュ法によるエラー許容頻出パターン列挙

橋本 英樹[1], 小野 廣隆[2], 宇野 毅明[3],
漆原 秀子[4], 柳浦 睦憲[5]
[1] 京都大学, [2] 九州大学, [3] 情報学研究所,
[4] 筑波大学,[5] 名古屋大学

ゲノム配列から，ある機能発現に関わる遺伝子を発見したいという要望がある．このような遺伝子は対象となるゲノム配列集合においてエラーを含んだ形で頻出する配列パターンの形をとることがしばしばある．本研究では，エラーを含む文字列集合から，ある長さの頻出部分文字列パターンを高速に全列挙するアルゴリズムを提案する．エラーの影響から通常のパターン列挙と異なり，入力文字列には現れないパターンも列挙の対象となる．提案手法ではパターン頻出性の必要条件を利用して最小限の候補パターンをハッシュ格納することにより，高速な全列挙を実現する．

Neighborhood hashing for enumerating all frequent patterns allowing errors

Hideki Hashimoto[1], Hirotaka Ono[2], Takeaki Uno[3],
Hideko Urushihara[4], Mutsunori Yagiura[5]
[1] Kyoto University, [2] Kyushu University, [3] National Institute of Informatics
[4] University of Tsukuba, [5] Nagoya University

We propose a practically fast algorithm that enumerates all m-length substring patterns appearing in at least θ sequences among a given set of string sequences, where at most k errors are allowed for each appearance. The problem of enumerating such substring patterns is derived from the genome science, where frequent substring patterns allowing errors are candidates of a gene related to a certain function. From this context, some pattern should be enumerated even if it does not appear in any sequence, because it may match a sufficient number of sequences by allowing errors. In order to prevent overlooking such potential patterns, we propose a hash-based enumeration algorithm. The algorithm stores not only a substring pattern appearing in a sequence but also its neighboring patterns. By using several techniques/conditions to exclude non-frequent patterns, our algorithm achieves an efficient enumeration of frequent substring patterns allowing errors.

1 Introduction

We propose a practically fast algorithm that enumerates all fixed-length substring patterns appearing in many of given string sequences allowing a small size of errors.

In the field of genome science, identifying a gene related to a certain function from a set of genome sequences is a central issue but is computationally expensive at the same time, because the gene usually has an enormous number of candidates. A reasonable way to reduce the number of the candidates is to restrict them to frequent patterns [1]. In general, the term "frequent pattern" roughly means, for a given data set, a pattern that *matches* many members in the set. In the context of the gene identification, it is natural to consider that a substring pattern (a candidate of the target gene) matches data sequences (genome sequences) allowing a few characters mismatching, because genome sequences often include experimental/biological errors. This is a motivation to consider the problem of enumerating frequent substring patterns allowing errors.

In this paper, a pattern is considered to appear in a sequence if the mismatch between the pattern and a substring is at most k for a given parameter k. We consider the problem of enumerating all m-length substring patterns appearing in at least θ sequences among a given set of string sequences $S = \{S_1, S_2, \ldots, S_n\}$ where m and θ are the input parameters. It should be noted that some pattern should be enumerated even if it does not appear in any sequence, because it may match a sufficient number of sequences by allowing errors. One way to prevent overlooking such potential patterns is to prepare counters for all potential patterns, but the number of potential patterns can be quite large, which causes several problems: It may require too much memory, and the counting may be slow unless a suitable data structure is adopted. Taking these

into account, we propose a counting-type algorithm that utilizes a hash, which is a fundamental data structure for storing unordered objects systematically. The algorithm stores not only a substring pattern appearing in a sequence but also its neighboring patterns.

Here, we briefly explain the ideas of our algorithm for excluding as many non-frequent patterns as possible. We use the principle that each appearance of an m-length pattern allowing k errors must have a substring that has a smaller error rate than the average k/m. To apply the principle, our algorithm first finds r-length substrings with small error rate, where r is smaller than m. We call such r-length substrings *seed substrings*. It then constructs (and store into the hash) m-length substring patterns by expanding an r-length seed substring along the sequences from the original position. Note that patterns that do not appear in the sequence may be constructed, because the expansion is done with allowing k errors. We call this procedure *neighborhood expansion*. In the neighborhood expansion phase, we can exclude some patterns by two necessary conditions for the frequent patterns allowing k errors. Finally, the frequencies of the generated m-length patterns are counted.

A similar problem has been studied in the context of pattern mining from large databases. Uchida, Asai and Arimura [3] proposed a pattern-growth type enumeration algorithm that enumerates frequent patterns of all lengths allowing errors measured by the edit distance (insertion, deletion and exchange) from string sequences. In contrast to our problem, frequent patterns with any length are considered while ours considers patterns with a fixed length m. Moreover, in their problem, the error measure is defined as the edit distance instead of the number of mismatched characters between strings in our problem.

2 Problem

Let $S = \{S_1, S_2, \ldots, S_n\}$ be a set of sequences. Each sequence S_l is a string whose alphabet set is Σ (e.g., $\Sigma = \{\mathrm{A}, \mathrm{T}, \mathrm{G}, \mathrm{C}\}$). Let $p[i, j]$ be a substring of string p that starts at position i and ends at position j of p. We abbreviate $p[i, i]$ as $p[i]$. Let $|p|$ denote the length of string p. For given strings p and q with the same length (i.e., $|p| = |q|$), we say that p matches q allowing k errors if the number of mismatched characters between p and q is at most k (i.e., $|\{i \mid p[i] \neq q[i]\}| \leq k$). The objective is to enumerate all m-length patterns appearing in at least θ sequences of S allowing k errors.

3 r-length seed substrings

In this section, we discuss the properties of a frequent pattern allowing errors and how to utilize them in our algorithm.

Suppose that a pattern p of size m appears in a sequence S_l with at most k errors. We consider the partition of p into r-length blocks. Let $s = \lfloor |p|/r \rfloor$ and let $p^1, p^2, \ldots, p^s$ be the first s blocks of p; i.e., $p^1 = p[1, r], p^2 = p[r+1, 2r], \ldots, p^s = p[(s-1)r+1, sr]$. Then we note the following property.

Property 3.1 *If a pattern p of size m appears in a sequence allowing k errors, there exists a block p^μ that contains at most $\lfloor k/s \rfloor$ errors.*

Otherwise, each block contains more than $\lfloor k/s \rfloor$ errors, p contains $(\lfloor k/s \rfloor + 1)s > (k/s)s = k$ errors in total, and it is a contradiction. Furthermore, if p appears in at least θ sequences of S allowing k errors, then, each appearance at the θ sequences has a block allowing $\lfloor k/s \rfloor$ errors and hence, the following property holds.

Property 3.2 *If p appears in at least θ sequences of S allowing k errors, there exists a block p^μ that appears in at least $\lceil \theta/s \rceil$ sequences allowing $\lfloor k/s \rfloor$ errors.*

Let $\kappa = \lfloor k/s \rfloor$. If κ is sufficiently small, it is possible to enumerate all r-length patters appearing in S allowing κ errors since the total number of patterns is $O\left((\sum_{i=1}^{n} |S_i|)(\sum_{\epsilon=0}^{\kappa} \binom{r}{\epsilon} |\Sigma|^\epsilon)\right)$.

Our algorithm first enumerates all r-length patters appearing in S allowing κ errors, and patterns appearing in at least $\lceil \theta/s \rceil$ sequences among them are considered as candidates of p^μ. We call these r-length patterns seed substrings. The seed substrings and their appearance positions in sequences are stored in a hash data structure. In the experiments, we set r to be the maximum that satisfies $\kappa \leq 1$.

For each r-length seed substring, our algorithm considers the cases that the candidate is p^μ for $\mu = 1, \ldots, s$. If a seed substring is p^μ of a frequent pattern p, p also appears in at least $\lceil \theta/s \rceil$ sequences of the corresponding appearing positions of the seed substring. The algorithm conversely enumerates such m-length patterns as candidates of a frequent pattern p. We call this step the neighborhood expansion of the seed substring. The details are described in Section 4.

4 Neighborhood expansion for an r-length seed substring

In this section, as described in Section 3, we consider the neighborhood expansion of an r-length seed substring for the case that the seed substring is p^μ, i.e., the problem of enumerating m-length strings that match $\lceil \theta/s \rceil$ strings allowing k errors from different sequences of the set of m-length strings.

Let $\nu = \lceil \theta/s \rceil$ and let $\{T_l^u\}$ be the set of m-length substrings of S_l, where p^μ appears as the μth block of each T_l^u allowing κ errors, and T_l^u corresponds to the uth appearance in S_l. Let L denote the number of elements in $\{T_l^u\}$. In the following, we propose two necessary conditions, which we call the 0-degree and 1-degree conditions, for that there exists a pattern that matches ν strings allowing k errors from different sequences. Only if the necessary conditions are satisfied, for each T_l^u, the possible neighboring strings that match T_l^u allowing k errors are generated. The number of neighboring strings for an r-length seed substring is $O(\sum_{\epsilon=0}^{k} \binom{m}{\epsilon} |\Sigma|^\epsilon L)$.

4.1 The 0-degree condition

The 0-degree condition is derived from the total number of each character at each position for the strings T_l^u. Let

$$F^0(l,i,a) = \begin{cases} 1, & \text{if } \exists u,\ T_l^u[i] = a \\ 0, & \text{otherwise.} \end{cases}$$

For any pattern p, the ith characters of $\sum_{l=1}^{n} F^0(l,i,p[i])$ sequences exactly match $p[i]$. Then, from the total number of exact match characters for ν strings,

$$\sum_{i=1}^{m} \min \left\{ \nu, \max_{a \in \Sigma} \sum_{l=1}^{n} F^0(l,i,a) \right\} \geq (m-k)\nu$$

must hold. We call this inequality the 0-degree condition. It can be tested in $O(mL)$ time.

4.2 The 1-degree condition

The 1-degree condition is derived from the total number of character pairs at each position for the strings T_l^u.

Let

$$F^1(l,i,j,a,b) = \begin{cases} 1, & \text{if } \exists u,\ T_l^u[i] = a \text{ and } T_l^u[j] = b \\ 0, & \text{otherwise.} \end{cases}$$

For any pattern p, characters $p[i]$ and $p[j]$ exactly match the corresponding characters of $\sum_{l=1}^{n} F^1(l,i,p[i],p[j])$ sequences. Let σ be a permutation of size m and $\sigma(i)$ be the ith element of σ. It is easy to see that

$$\sum_{i=1}^{m} \min \left\{ \nu, \max_{a,b \in \Sigma} \sum_{l=1}^{n} F^1(l,i,\sigma(i),a,b) \right\} \geq (m-2k)\nu$$

must hold for any σ. We call this inequality the 1-degree condition. The problem of finding a σ which minimizes the left-hand side can be formulated as the assignment problem, where the cost of assigning i to j is

$$c(i,j) = \min \left\{ \nu, \max_{a,b \in \Sigma} \sum_{l=1}^{n} F^1(l,i,j,a,b) \right\}.$$

The assignment problem can be solved efficiently [2]. The computation time for preparing all $c(i,j)$ is $O(\binom{m}{2} L)$ time.

5 Computational experiments

We conducted computational experiments to evaluate the proposed algorithm. The algorithm was coded in C and run on a PC (Xeon, 2.8 GHz, 1 GB memory). We used first 1100 characters from each 100 genome sequences (i.e., $n = 100$, $|S_l| = 1100$) and $\Sigma = \{\mathrm{A}, \mathrm{T}, \mathrm{G}, \mathrm{C}, \mathrm{N}\}$ where "N" means an unknown amino acid code in FASTA format.

We compared our results with that of Uchida, Asai and Arimura [3]. Their algorithm enumerates all patterns of any length which appear in at least θ sequences allowing k edit distances. It can be applied to our problem by setting the insertion and deletion costs to a large value. Since their algorithm is a pattern-growth type enumeration, it cannot be applied directly to the enumeration with fixed m.

Table 1 shows our computational results for several settings of m and k with fixed $\theta = 90$. Column N_{succ} (resp., N_{fail}) denotes the number of times when the necessary condition cuts, i.e., the 0-degree and 1-degree conditions, succeed (resp., fail). Column N_{posi} denotes the number of r-length seed substrings which generate an m-length candidate pattern in Section 4. Such a subpattern always satisfies any necessary conditions including the 0-degree and 1-degree conditions, and $N_{\mathrm{posi}} \leq N_{\mathrm{fail}}$ holds. In the experiments, the neighborhood expansions are not executed when $k \leq 1$, because we set $\kappa \leq 1$. In such cases, marks "-" are shown in the columns of N_{succ}, N_{fail} and N_{posi}. Column P_{gen} denotes the number of generated m-length patterns and column P_{freq} denotes the number of frequent patterns.

Table 2 shows the results of Uchida, Asai and Arimura [3] with $\theta = 90$ where the experiments were conducted in our environment. Column $m_{\max}$ denotes the maximum length of frequent patterns.

From the experimental results, we can observe, for a given length m, our algorithm can enumerate all m-length frequent patterns much faster than that of Uchida, Asai and Arimura [3].

Table 1: Our results for several settings

m	k	N_{succ}	N_{fail}	N_{posi}	P_{gen}	P_{freq}	time(s)
12	0	-	-	-	3	3	0.39
12	1	-	-	-	413	413	12.59
12	2	2908	7566	2305	260,420	23,737	897.27
12	3	930	9544	4619	4,077,770	775,558	10437.22
20	0	-	-	-	0	0	0.50
20	1	-	-	-	7	7	29.80
20	2	6636	6410	1690	84,706	1285	436.87
20	3	4354	8692	2612	2,121,061	91,093	8737.76
28	0	-	-	-	0	0	0.58
28	1	-	-	-	0	0	52.03
28	2	2164	3110	304	9658	0	215.60
28	3	1291	3983	1178	451,322	16	4269.34

Table 2: Computational results of Uchida, Asai and Arimura [3]

k	$m_{\max}$	time(s)
0	13	4.45
1	20	140.04
2	25	3477.97
3	29	70100.74

6 Conclusion

In this paper, we proposed an algorithm that enumerates all m-length substring patterns appearing in at least θ sequences allowing k errors from a given set of string sequences S. Our hash-based enumeration algorithm achieves an efficient enumeration by using several techniques/conditions to exclude non-frequent patterns. The experimental results show that our algorithm runs practically fast for a reasonable size of examples.

Acknowledgment

The authors would like to thank Hiroki Arimura for providing us with their source code [3].

References

[1] N. Kobayashi, M. Marin, Y. Tanaka, and H. Urushihara. On the development of an analysis system for upstream sequences in dictyostelium discoideum genome. *Computer Software*, 22(3):167–172, 2005.

[2] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms.* Springer, 3 edition, 2005.

[3] Y. Uchida, T. Asai, and H. Arimura. Discovering frequent approximate episodes in large sequence databases (in Japanese). In *Proceedings of the 15th Data Engineering Workshop (DEWS2004)*, 2004.