CRIMSON PUBLISHERS
Wings to the Research

Open Access
Biostatistics & Bioinformatics

ISSN 2578-0247

Mini Review

String Matching in DNA Databases

Yangjun Chen*

Department of Applied Computer Science, University of Winnipeg, Canada

***Corresponding author:** Yangjun Chen, Department of Applied Computer Science, University of Winnipeg, Canada.

Submission: March 13, 2018; **Published:** May 11, 2018

Abstract

The recent development of next-generation sequencing has changed the way we carry out the molecular biology and genomic studies. It has allowed us to sequence a DNA (Deoxyribonucleic acid) sequence at a significantly increased base coverage, as well as at a much faster rate [1]. This facilitates building an excellent platform for a whole genome sequencing, and for a variety of sequencing-based analyses, including gene expressions, mapping DNA-protein interactions, whole-transcriptome sequencing, and RNA (Ribonucleic acid) splicing profiles. For example, the RNA-Seq protocol [2], in which processed mRNA is converted to cDNA and then sequenced, is enabling the identification of previously unknown genes and alternative splice variants. The whole-genome sequencing of tumour cells can uncover previously unidentified cancer-initiating mutations.

Recent Progress

The core and the first step to take advantage of the new sequencing technology is termed as read aligning, where a read is a short nucleotide sequence of 30-1000 base pairs (bp) [3] generated by a high throughput sequencing machine made by Illumina, Roche, ABI/Life Technologies [4], which is in fact a sequence fragment fetched from a longer DNA molecule present in a sample that is fed into the machine (https://github.com/lh3/wgsim/). Most of the next-generation sequencing projects begin with a reference sequence which is a previously well studied, known genome. The process of a read aligning is to find the meaning of reads, or in other words, to determine their positions within a reference sequence, which will then be used for an effective statistical analysis.

Compared to the traditional pattern matching problems, the new challenge from the read aligning is its enormous volume, and usually millions to billions of reads need to be aligned within a same genome sequence. For example, to sequence a human molecule sample with 15X coverage, one may need to align 1.5 billion reads of length about 100 characters (bps). Another challenge is the so-called inexact matching, by which we will find all the subsequences in a genome sequence having at most k positions different from a read or with k errors. Due to the polymorphisms or mutations among individuals or even sequencing errors, a read may disagree in some positions by an occurrence in the corresponding genome. Again, due to the huge number of reads, the existing methods have to be greatly modified to fit into such a new environment. In the past decade, a new indexing structure based on the [5] to speed up the matching of massive reads against different genome sequences, based on the so-called Burrows-Wheeler transformation (BWT-transformation for short) ([6] and also www.youtube.com/watch?v=4n7NPk5-lwbI) and trie structures [7] or automata over reads [8]. In terms of the test results reported in [9], an exact read matching can be done 40% faster using our new method than any existing strategies. The algorithm has been successfully used in a biological research project conducted in an RNA laboratory at University of Manitoba [10] (http://home.cc.umanitoba.ca/~xiej/). The BWT can also be utilized to expedite the string matching with k-mismatches [11,12]. However, this method cannot be easily extended to handle the massive string matching with k-mismatches or k-errors since when searching a trie or an automaton the mismatch information or the word periodicity cannot be directly employed in a BWT array scanning.

Literature Review

The matching of DNA sequences is just a special case of the general string matching problem, which has always been one of the main focuses in computer science. All the methods developed up to now can be roughly divided into two categories: exact matching and inexact matching. By the former, all the occurrences of a pattern string p in a target string s will be searched. By the latter, a best alignment between p and s (i.e., a correspondence with the highest score) is looked for in terms of a given score matrix M, which is established to indicate the relevance between characters [13] (more exactly, the meanings represented by them.)

Exact matching

Scanning-based: By this kind of algorithms, both pattern p and targets are scanned from left to right, but often with an auxiliary data structure used to speed up the search, which is typically constructed by a pre-processor. The first of them is the famous

Copyright © All rights are reserved by Yangjun Chen.

Knuth & Morris [14] algorithm, which employs an auxiliary next-table (for p) containing the so-called shift information (or say, failure function values) to indicate how far to shift the pattern from right to left when the current character in p fails to match the current character in s. Its time complexity is bounded by O(m+n), where m=|p| and n=|s|. The Boyer & Moore [15] approach works a little bit better than the Knuth & Morris [14]. In addition to the next-table, a skip-table (also for p) is kept. For a large alphabet and small pattern, the expected number of character comparisons is about n/m, and is O(m+n) in the worst case. Although these two algorithms have never been used in practice [16], they sparked a series of research on this problem, and are improved by different researchers in different ways, such as the algorithms discussed in [17-21]. However, the worst-case time complexity remains unchanged. The idea of the 'shift information' has also been adopted indifferent approaches [8,22-24] for the multiple-string matching, by which s is searched for the occurrences of any one of a set of k patterns: $\{p_1,p_2,...,p_k\}$. Their algorithm needs only $O(\sum_{i=1}^{k} mi+n)$ time, where $m_i=|p_i|$ (i=1,...,k). However, this algorithm cannot simply be adapted to an index environment due to its working fashion to search the characters in some by one, which is totally unsuitable for indexes.

Index-based: In situations where a fixed string s is to be searched repeatedly, it is worthwhile constructing an index over s, such as suffix trees, suffix arrays, and more recently the BWT-transformation. A suffix tree is in fact a trie structure [7] over all the suffixes of s; and by using the Weiner's algorithm it can be built in O(n) time [25]. However, in comparison with suffix trees, the BWT-transformation is more suitable for DNA sequences due to its small alphabet Σ since the smaller Σ is, the smaller space will be occupied by the corresponding BWT array [26]. According to a survey done by Li & Homer [1] on sequence alignment algorithms for next-generation sequencing, the average space required for each character is 12-17 bytes for suffix trees while only 0.5-2 byte for the BWT. The experiments reported in [9] also confirm this distinction. For example, the file size of chromosome 1 of human is 270Mb. But its suffix tree is of 26Gb in size while its BWT needs only 390Mb-1Gb for different compression rates of auxiliary arrays, completely handle-able on PC or laptop machines. The huge size of a suffix tree may greatly increase the computation time. For example, for the Zebra fish and Rat genomes (sizes 1,464,443,456pb, and 2,909,701,677pb, respectively), one cannot finish the construction of their suffix trees within two days in a computer with 32GB RAM [27].

Hash-based: Intrinsically, all hash-table-based algorithms [28-30] extract short subsequences called 'seeds' from a pattern sequence p and create a signature (a bit string) for each of them. The search of a target sequence s is similar to that of the Brute Force searching, but rather than directly comparing the pattern at successive positions in s, their respective signatures are compared. Then, stick each matching seed together to form a complete alignment. Its expected time is O(m+n), but in the worst case, which is extremely unlikely, it takes O(m.n) time. The hash technique has also been extensively used in the DNA sequence research [30,31], and all experiments show that they are generally inferior to the suffix tree and the BWT index in both running time and space requirements [27,31,32].

Inexact string matching

K-mismatches and k-errors: The inexact matching [33] ranges from the score-based to the k-mismatching [34-36], as well as the k-error [37]. By the score-based method, a matrix M of size $|\Sigma|\times|\Sigma|$ is used to indicate the relevance between characters. The algorithm designed is to find the best alignment (or say, the alignment with the highest scores) between two given strings, which can be DNA sequences, protein sequences, or XML documents; and the dynamic programming paradigm is often utilized to solve the problem [38]. By the string matching with k-mismatching or k-errors, we will find all those subsequences q of s such that $d(p,q)\leq k$, where d() is a distance function. When it is the Hamming distance (defined to be the number of different positions), the problem is known as the sequence matching with k mismatches. When it is the Levenshtein distance, the problem is known as the sequence matching with k errors [8,26]. By the Levenshtein distance, the distance function is defined as follows:

$$d_{i,j} = min\{d_{i-1,j} + w(p_i,\Phi),\ d_{i,j-1} + w(\Phi,\ q_j),\ d_{i-1,j-1} + w(p_i,\ q_j)\}$$

where $d_{i,j}$ represents the distance between p[1..i] and q[1..j], $p_i(q_j)$ the ith character in p(j^{th} character in q), ϕ an empty character, and w(pi,qj) the cost to transform pi into qj.

There is a bunch of algorithms proposed for this problem, such as for the k-mismatch; and [39,40] for the k-error. By the best method for the k-mismatch a time complexity $O(n\sqrt{k}\log k)$ can be achieved. Especially, for small k and large Σ, the search requires a sublinear time on average. In addition, the BWT and suffix trees can also be used as indexes for this problem. For the k-error, the worst case time complexity is bounded by O(mn). But the expected time can reach O(k·n) by an algorithm discussed in [33].

Don't care symbols: As a different kind of inexact matching, the string matching with Don't-Cares (or wild-cards) [41] has also been an active research topic for decades, by which we may have wild-cards in p, in s, or in both of them. A wild card matches any character. Due to this property, the 'match' relation is no longer transitive, which precludes straightforward adaption of the shift information used by Knuth & Morris [14]. All the methods proposed to solve this problem also require quadratic time [42]. Using a suffix array as the index, however, the searching time can be reduced to O(logn) for some patterns, which contain only a sequence of consecutive Don't Cares [43].

Methodology

Massive string pattern mapping

By the massive pattern matching, we mean to find all the occurrences of all the patterns in a target string s. The number of

How to cite this article: Yangjun C. String Matching in DNA Databases. Open Acc Biostat Bioinform. 1(4). OABB.000523. 2018.
DOI: 10.31031/OABB.2018.01.000523

patterns can be millions to billions. In [9], a method is proposed to build a BWT-array for s, denoted as BWT(s), and to organize the patterns into a trie structure T, which is in fact a tree such that all those patterns with the same prefixes will be clustered together. Then, exploring a path in T, a group of patterns will be searched, instead of a single one, against BWT(s). Especially, the so-called multi-character checking is used, by which multiple characters from different patterns will be checked at each step when a segment in BWT(s) is scanned. In this way, high efficiency can be achieved. This method can be further improved by organizing all the patterns into an automaton [44], which is constructed by adding some links between nodes to trie T with each from a node v to another node u such that the string represented by the path from the root to u is the same as a suffix of the string represented by the path from the root to v. One objective of this method is to extend this kind of trie-based methods to an automaton-based method. For this, two issues should be investigated:

A. How to use links to speed up a search of BWT(s)?

B. Whether the multi-character checking and links can be simultaneously used to speed up a searching?

Inexact matching

As mentioned in Section 2, by the inexact string matching, we distinguish between k-mismatches and k-errors, as well as the string matching with don't-care symbols. What needs to be done is to shift all the problems to an environment of massive patterns and very long targets.

String matching with k-mismatches

By the string matching with k mismatches, we mean a problem to find all the occurrences of a pattern string pin a target string s with each occurrence having up to k positions different between p and s. This problem is important for DNA databases to support the biological research, where we need to locate all the appearances of a read in a genome sequence for disease diagnosis or some other purposes. Due to possible mistakes or mutations among different individuals, the occurrence of p in target s with some differences may need to be considered. As an example, consider a target s=ccacacagaagcc, and a pattern p=aaaaacaaac. Assume that k=4. Let us see whether there is an occurrence of pwith $\leq$k mismatches that starts at the third location in s.

```
a a a a a c a a a c
c c a c a c a g a a g c c
      |   |   |     |
```

At only four locations s and p have different characters, implying an occurrence of p starting at the third location of s. Note that the case k=0 is the extensively studied string matching problem.

An initial strategy for this problem is to build a BWT array A for s, and for a pattern p search A against p repeatedly to find all the possible string matching with k mismatches. This requires O(mn) time in the worst case, even worse than some existing on-line algorithms. To avoid redundant work done in the above process, a special tree structure, called an S-tree, needs to be utilized to record information on the mismatches between p and each path P made up of some entries of A, which are searched to find some possible string matchings with k mismatches. Concretely, each path in the S-tree can be considered as a vector V of length k+1 such that V[i]=j if and only if p[j]≠P[j] and it is the ith mismatch between p and P. More importantly, this kind of information can be used to derive mismatches between p and some other parts of s in terms of the mismatching information among different parts of p, instead of searching them in a normal way. Using the S-tree, the time complexity can be reduced to O(kn′), where n′≪n. This is better than $O(n\sqrt{k}\log k)$, the best time complexity achievable by an on-line algorithm, when $n' < \frac{\log k}{\sqrt{k}} n$. In extensive experiments have also been conducted, showing that the method with the S-tree uniformly outperforms any existing on-line approach. In a probabilistic analysis of n′ is given. In a next step, it will be investigated how to use the periodicity of pto speed up the computation, which has been used by the on-line algorithm discussed in [34]. However, it is not clear whether the periodicity of p can also be employed in an indexing environment. So, a thorough research is required on all different indexing mechanisms: suffix trees, suffix arrays, hashing and BWT arrays to see how the periodicity of p can be integrated.

String matching with k-errors

The string matching with k-errors is quite different from the string matching with k-mismatches. Hence, the methods developed for solving k mismatches cannot be directly used for them. In fact, almost all the algorithms proposed for k errors are based on the dynamical programming paradigm, using an n×m matrix to store differences between p[1..i] and s[1..j] for each pair i, j(i=1,...,m; j=1,...,n). However, for the string matching with k errors, the matrix can be created column by column and thus only O(m) space is required for the task. In addition, by using the so-called diagonal wise monotocinity of the Levenshtein distance table recognized by Ukkonen [40], many entries in the matrix needn't be generated, leading to an average time complexity O(kn) [42], which has been further improved by involving the use of mismatching statistics, and suffix trees built for patterns, as well as the partition of columns and the so-called shift-add strategy proposed by Baeza & Manber [43]. However, no effort has been made to index the target string in a way like the exact matching. As for the k mismatches, a BWT array for s needs to be constructed and searched, but combined with the dynamical programming computation. Specifically, the following investigation will be conducted.

A. Check whether it is possible to create a dynamical programming matrix columnal wise during a BWT array search.

B. Check whether the diagonal wise monotocinity of the Levenshtein distance table can be used during a BWT search.

C. Finally, we will also try to integrate the partition of columns and the use of matching statistics into a BWT array search.

How to cite this article: Yangjun C. String Matching in DNA Databases. Open Acc Biostat Bioinform. 1(4). OABB.000523. 2018.
DOI: 10.31031/OABB.2018.01.000523

String matching with don't cares

We distinguish between two kinds of string matching with don't cares: only the pattern containing don't care symbols, and both the pattern and the target containing don't care symbols. For the former problem, by establishing a suffix array for s, in which the positions of all the suffixes in s are stored according to the lexicographic order of the respective suffixes, the time complexity is bounded by $O(O^{1/4}+r)$, where r is the number of p's occurrences in s. For the latter, by using the shift-add algorithm [45], only $O(\lceil\frac{m}{w}\rceil n)$ time is required, where w is the word size in bits of a computer.

For this research, the focus will be mainly on the latter problem to integrate the shift-add technique into a BWT array search. More specifically, the algorithm discussed in [45] should be modified by replacing the scan of s with a scan of BWS[s]. Since by BWS[s] we can check a character in p against multiple positions with the same character in s at each step, the algorithm discussed in can be possibly greatly improved.

Summary

In this article, a mini-review on the string matching in DNA databases is delivered and some potential researches on this issue are briefly discussed. It seems that by using the BWT-transformation as part of indexes, new algorithms can be achieved, which possibly work better than the traditional on-line strategies.

References

1. Li H, Homer N (2010) A survey of sequence alignment algorithms for next-generation sequencing. Briefings in Bioinformatics 11(5): 473-483.
2. Jiang H, Wong WH (2008) Seq Map: mapping massive amount of oligo nucleotides to the genome. Bioinformatics 24(20): 2395-2396.
3. Schatz M (2009) Cloudburst: highly sensitive read mapping with map reduce. Bioinformatics 25(11): 1363-1369.
4. Bolger AM, Lohse M, Usadel B (2014) Trimmomatic bolger: A flexible trimmer for illumina sequence data. Bioinformatics 30(15): 2114-2120.
5. Chen Y, Wu Y (2018) On the string matching with k-mismatches. Theoretical Computer Science 726: 5-29.
6. Burrows M, Wheeler DJ (1994) A block-sorting lossless data compression algorithm. pp. 1-24.
7. Knuth DE (1975) The art of computer programming. Addison-Wesley Publish Com, Massachusetts, USA, Vol. 3.
8. Aho AV, Corasick MJ (1975) Efficient string matching: an aid to bibliographic search. Communication of the ACM 23(1): 333-340.
9. Chen Y, Wu Y, Xie J (2016) An efficient algorithm for read matching in DNA databases. Proc Int Conf DBKDA'2016, Lisbon, Portugal, pp. 23-34.
10. Wu S, Manber U (1994) A fast algorithm for multi-pattern searching. Technical Report TR-94-17, Dept Computer Science, Chung-Cheng University, Taiwan.
11. Chen Y, Wu Y (2018) BWT: An index structures to speed-up both exact and inexact string matching. In: Samui, Ntalampiras R (Eds.), Springer Verlag, Germany.
12. Chen Y, Wu Y (2017) BWT arrays and mismatching trees: A new way for string matching with k mismatches. Proc Intl Conf on Data Engineering (ICDE2017), IEEE, San Diego, USA, pp. 339-410.
13. Smith AD, Xuan Z, Zhang MQ (2008) Using quality scores and longer reads improves accuracy of Solexa read mapping. BMC Bioinformatics 9: 128.
14. Knuth DE, Morris JH, Pratt VR (1977) Fast pattern matching in strings. SIAM Journal on Computing 6(2): 323-350.
15. Boyer RS, Moore JS (1977) A fast string searching algorithm. Communications of the ACM 20(10): 762-772.
16. Navarro G, Raffinot M (2002) Pattern matching in strings. Cambridge University Press, USA.
17. Apostolico A, Giancarlo R (1986) The Boyer-Moore-Galil string searching strategies revisited. SIAM Journal on Computing 15(1): 98-105.
18. Colussi L, Galil Z, Giancarlo R (1990) On the exact complexity of string matching. Proc. 31st Annual IEEE Symposium of Foundation of Computer Science 1: 135-144.
19. Galil Z (1977) On improving the worst case running time of the Boyer-Moore string searching algorithm. Communication of the ACM 22(9): 505-508.
20. Lecroq T (1992) A variation on the Boyer-Moore algorithm. Theoretical Computer Science 92(1): 119-144.
21. Tarhio J, Ukkonen E (1993) Approximate boyer-moore string matching. SIAM Journal on Computing 22(2): 243-260.
22. Commentz B (1979) A string matching algorithm fast on the average. Proc 6th Colloquium on Automata, Languages and Programming, pp. 118-132.
23. Crochemore M, Czumaj A, Gasieniec L, Jarominek S, Lecroq T, et al. (1999) Fast practical multi-pattern matching. Information Processing Letters 71: 107-113.
24. Hon W, Tak WL, Kunihiko S, Wing-KS, Siu MY (1973) A space and time efficient algorithm for constructing compressed suffix arrays. Algorithmica 48(1): 23-36.
25. Weiner P (2006) Linear pattern matching algorithm. Proc 14th IEEE Symposium on Switching and Automata Theory pp. 1-11.
26. Li H, Durbin R (2010) Fast and accurate long-read alignment with Burrows-Wheeler transform. Bioinformatics 26(5): 589-595.
27. Harrison MC (1971) Implementation of the substring test by hashing. Communication of the ACM 14(12): 777-779.
28. Karp RL, Rabin MO (1987) Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development 31(2): 249-260.
29. Lin H, Zhang Z, Zhang MQ, Ma B, Li M (2008) ZOOM! Zillions of oligos mapped. Bioinformatics 24(21): 2431-2437.
30. Li H, Ruan J, Durbin R (2008) Mapping short DNA sequencing reads and calling variants using mapping quality scores. Genome Res 18: 1851-1858.
31. Li R, Li Y, Kristiansen K, Wang J (2008) SOAP: Short oligonucleotide alignment program. Bioinformatics 24(5): 713-714.
32. Chang WL, Lampe J (2005) Theoretical and empirical comparisons of approximate string matching algorithms. In: Apostolico A, Crocchemore M, Galil Z, Manber U (Eds.), Combinatorial pattern matching, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany, 644: 175-184.
33. Amir A, Lewenstein M, Porat E (2004) Faster algorithms for string matching with k mismatches. Journal of Algorithms 50(2): 257-275.
34. Galil Z, Giancarlo R (1986) Improved string matching with k mismatches. ACM SIGACT News 17(4): 52-54.

35. Landau GM, Vishkin U (1986) Efficient string matching with k mismatches. Theoretical Computer Science 43: 239-249.

36. Landau GM, Vishkin U (1985) Efficient string matching in the presence of errors. Proc 26th Annual IEEE Symposium on Foundations of Computer Science 126-136.

37. Eddy SR (2004) What is dynamic programming? Nat Biotechnol 22(7): 909-910.

38. Tarhio J, Ukkonen E (1993) Approximate boyer-moore string matching. SIAM Journal on Computing 22(2): 243-260.

39. Ukkonen E (1992) Approximate string-matching with q-grams and maximal matches. Theoretical Computer Science 92(1): 191-211.

40. Manber U, Myers EW (1990) Suffix arrays: A new method for on-line string searches. Proc the 1st Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, USA, pp. 319-327.

41. Pinter RY (1985) Efficient string matching with don't' care patterns. In: Apostolico A, Galil Z (Eds.), Combinatorial Algorithms on Words, NATO ASI Series, Springer-Verlag, Berlin, Germany, 12: 11-29.

42. Manber U, Baeza RA (1991) An algorithm for string matching with a sequence of don't cares. Information Processing Letters 37(3): 133-136.

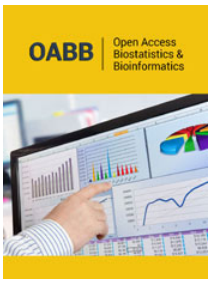
43. Chen Y, Wu Y (2017) Searching BWT against pattern matching machine to find multiple string matches. Proc Intl Conf on Cyber-Enabled Distributed Computing and Knowledge Discovery, IEEE, Nanjing, China, pp. 167-176.

44. Seward J (2007) Bzip2 and libbzip2, version 1.0. 5: A program and library for data compression.

45. Xie J, Chen Y. Safeguarding transcriptome integrity and hormone production by hnRNP L. Cell Research.

Creative Commons Attribution 4.0 International License

For possible submissions Click Here Submit Article



Open Access Biostatistics & Bioinformatics

Benefits of Publishing with us

- High-level peer review and editorial services
- Freely accessible online immediately upon publication
- Authors retain the copyright to their work
- Licensing it under a Creative Commons license
- Visibility through different online platforms