BIOINFORMATICS

Vol. 26 ECCB 2010, pages i414–i419
doi:10.1093/bioinformatics/btq364

A fast algorithm for exact sequence search in biological sequences using polyphase decomposition

Abhilash Srikantha[1,*], Ajit S. Bopardikar[1,*], Kalyan Kumar Kaipa[1], Parthasarathy Venkataraman[1], Kyusang Lee[2], TaeJin Ahn[2] and Rangavittal Narayanan[1]

[1]Samsung Advanced Institute of Technology India Lab, Bangalore, Karnataka, India and [2]Genetic Analysis Group, Emerging Center, Samsung Advanced Institute of Technology, Suwon, South Korea

ABSTRACT

Motivation: Exact sequence search allows a user to search for a specific DNA subsequence in a larger DNA sequence or database. It serves as a vital block in many areas such as Pharmacogenetics, Phylogenetics and Personal Genomics. As sequencing of genomic data becomes increasingly affordable, the amount of sequence data that must be processed will also increase exponentially. In this context, fast sequence search algorithms will play an important role in exploiting the information contained in the newly sequenced data. Many existing algorithms do not scale up well for large sequences or databases because of their high-computational costs. This article describes an efficient algorithm for performing fast searches on large DNA sequences. It makes use of hash tables of Q-grams that are constructed after downsampling the database, to enable efficient search and memory use. Time complexity for pattern search is reduced using beam pruning techniques. Theoretical complexity calculations and performance figures are presented to indicate the potential of the proposed algorithm.

Contact: s.abhilash@samsung.com; ajit.b@samsung.com

1 INTRODUCTION

Decreasing costs of sequencing personal genome have opened up many avenues of research. Several efforts related to Personal Healthcare and Pharmacogenetics are attempting to use the information in an individual's genomic data towards personalization of healthcare. An important component of these solutions is the search for specific subsequences in a given genome. For example, Cetuximab (Eric *et al.*, 2009)—an Epidermal Growth Factor Receptor (EFGR) inhibitor used to treat various types of cancer is ineffective if certain mutations in the KRAS gene (which lies in Exon 2 of Chromosome 12) exist. Thus, a search for appropriate mutations conducted on genetic data would be vital in prescribing the most effective treatment. Exact sequence search also finds application in fields such as Evolutionary Biology and Phylogenetics where certain subsequences of DNA are mined from genomic data of various species of organisms to understand their origin, relatedness and descent.

In the above context, the challenge is to be able to perform fast pattern searches in whole genomes (a complete human genome contains 3 billion base pairs) and databases spanning Giga to Tera Bytes or more. Prevalent search methods (Altschul *et al.*, 1990; Charras and Lecroq, 2004; Gusfield 1997; Kurtz *et al.*, 2004; Li and Durbin, 2010; Lipman and Pearson, 1985; Ma *et al.*, 2002; Ning *et al.*, 2001) use techniques that have proven to be efficient for existing genomic sequences and databases, but do not scale up well for large datasets such as those of whole human genomes. Fast and efficient search methods that scale up well for large databases are therefore of great value.

In this article, we address the problem of searching for all occurrences of pattern P in a text T, where T is the reference sequence whose length can vary from several million (in case of individual chromosomes) to several billion (in case of complete genomes) bases. P is the pattern which is a few tens to a few hundred bases in length.

2 RELATED WORK

Many methods have been developed for the task of pattern search. These include FASTA (Lipman and Pearson, 1985; Pearson and Lipman, 1988), BLAST (Altschul *et al.*, 1990), PatternHunter (Ma *et al.*, 2002), MUMMER (Kurtz *et al.*, 2004), SSAHA (Sequence Search and Alignment using Hashing Algorithm) (Ning *et al.*, 2001), Fast String Matching Algorithms (Lecroq, 2007) and BWA-SW (Burrows Wheeler Alignment—Smith Waterman) (Li and Durbin, 2010).

Though Smith Waterman-based methods such as FASTA mines approximate matches by employing dynamic programming techniques, they are computationally very intensive. BLAST and its variants are an improvement over FASTA in that they use certain seeds for basic anchoring, which are then extended to exact or approximate matches. However, apart from being probabilistic in nature, BLAST type algorithms require large amounts of memory and computing time for pattern search in large sequences such as whole genomes. PatternHunter (Ma *et al.*, 2002) is a similar seed-based technique but is still inefficient for applications that involve whole genomes or large databases.

Recent suffix tree-based methods (Gusfield, 1997) such as Mummer (Kurtz *et al.*, 2004) that yield exact matches have a very low-search time complexity. They represent all suffixes of the text as a plurality of inter-mingled linked lists. At times when the knowledge about genomes gets updated frequently, updating the suffix tree in place becomes tedious as the inter-mingling of linked lists is very sensitive to changes in the text data. Also, as every node in the tree is required to hold tree-related information such as pointers to their parents and children apart from text-based information, even the best implementation of suffix trees require ~15.4 bytes per base (Kurtz *et al.*, 2004), which scales up to 46 GB of memory to store the preprocessed Human Genome.

Deterministic Finite Automaton (DFA)-based methods (Charras and Lecroq, 2004; Gusfield 1997) such as BWA-SW (Li and Durbin, 2010) combine DFA and dynamic-programming-based

*To whom correspondence should be addressed.

alignment methods. As this method involves Finite Automaton, the method does not scale well for large sequences, even for the best case of exact matches. Also, because they use dynamic programming, the memory requirements of the method are huge.

Hashing-based methods such as SSAHA (Ning *et al.*, 2001) and those proposed by Lecroq (2007) propose substring matching and Q-gram hashing method to greatly improve the time complexity.

To summarize, efficient biological pattern-search algorithms must mitigate two problems. First, that of random access into text, without which the time complexity of the algorithm shoots up to an unacceptable $O(L_T)$ (Charras and Lecroq, 2004; Knuth *et al.*, 1977; Melichar, 1995), where T is of the order of several billion bases. This can be mitigated by employing mechanisms such as suffix trees and hash tables. Hashing methods are considered because changing data locally is an easy task when information in the corresponding sequence gets updated. The second problem is that of memory constraints. Random access algorithms (Kurtz *et al.*, 2004; Lecroq, 2007; Ning *et al.*, 2001) come with an undesirable space complexity of $O(L_T)$, where L_T is the length of the text T is of the order of several billion bases.

In our work, we propose an efficient methodology that employs down-sampled version of T and polyphase decomposition of P to determine potential areas of exact match. These, in conjunction with hash tables and the use of Q grams to successively refine potential-search regions results in superior space and time complexity. Note that the present method differs from the existing methods (Lecroq, 2007; Ning *et al.*, 2001) in that, firstly, we consider down-sampled substrings of both text and the pattern that result in large memory savings. Secondly, the Q-grams have traditionally been used only for the purposes of string matching, but in the proposed method, their locations have been used to progressively localize the potential locations of exact match. We now describe the proposed algorithm.

3 PROPOSED METHOD

We first explain various terminologies we use in the exposition that follows. For this purpose, we use an example sequence, given by:

$$S = \text{'actgcttctact'}. \tag{1}$$

Let the length of the sequence S be denoted by L_S. Also, $S[n]$ is used to represent the n-th base of the sequence S. For example, $S[0]$ = 'a' and $S[1]$ = 'c'.

Downsampling (Vaidyanathan, 1993) a sequence by a factor of M means that we pick every M-th base from sequence S to form a new sequence SM given by:

$$\text{SM}[n] = S[\text{Mn}], 0 \le n \le \left\lfloor \frac{L_S}{M} \right\rfloor \tag{2}$$

where, $\lfloor\ \rfloor$ indicates the largest integer less than the argument. For example, for the sequence S and $M = 3$, S3 = 'agta'

M-channel polyphase decomposition (Vaidyanathan, 1993) gives M possible down-sampled sequences for different integer-phase shifts. The generalized form of polyphase decomposition is given by:

$$\text{SM}i[n] = S[Mn+i], 0 \le n \le \left\lfloor \frac{L_S}{M} \right\rfloor, 0 \le i \le M-1. \tag{3}$$

Note that SM = SM0. The polyphase decomposition of the sequence S in Equation (1) for $M = 3$ yields:

$$\text{S30} = \text{'agta'},\ \text{S31} = \text{'cccc' and S32} = \text{'tttt'}$$

A Q-gram of a sequence S is denoted by QS(n) and is made up of Q consecutive bases starting from position n. Thus for sequence S as in Equation (1) example Q-grams are:

$$\text{QS}(0) = \text{'actg' QS}(1) = \text{'ctgc'}$$
$$\text{QS}(2) = \text{'tgct' QS}(8) = \text{'tact'}$$

Contiguous Q-grams of a sequence S is the set:

$$\text{CQS} = \{\text{QS}(0)\text{QS}(1)\cdots\text{QS}(L_S - Q)\} \tag{4}$$

where the cardinality of the set CQS is $|\text{CQS}| = L_S - Q + 1$. Note that there are Q – 1 common bases between any two consecutive Q-grams in CQS. For example, given S as in Equation (1) and $Q = 4$,

CQS = {'actg', 'ctgc', 'tgct', 'gctt', 'cttc', 'ttct', 'tcta', 'ctac', 'tact'}.

NQS is the set of non-overlapping Q-grams of sequence S given by:

$$\text{NQS} = \left\{\text{QS}(0)\text{QS}(Q)\ldots\text{QS}\left(\left\lfloor \frac{L_S}{Q} \right\rfloor\right)\right\}. \tag{5}$$

Note that:

$$\forall\ \text{QS}(i) \text{ and } \text{QS}(j) \in \text{NQS},$$
$$\text{QS}(i) \cap \text{QS}(j) = \phi \text{ if } i \neq j$$

where ϕ denotes the null set. That is the Q-grams in NQS are pair-wise and non-overlapping.

For S as in Equation (1) and $Q = 4$, NQS = {'actg', 'cttc', 'tact'}.

We now describe the proposed method which is composed of two stages, namely:

(1) Preprocessing stage.

(2) Pattern-search stage.

The detailed description of each stage is presented in the following subsections.

3.1 Preprocessing stage

A block diagram for the preprocessing stage is shown in Figure 1. In this step, text T is downsampled and processed into a hash table to support random access into the text. A hash table is a data structure that efficiently links keys to corresponding values called buckets (Cormen *et al.*, 1990). In our case, the key refers to each distinct Q-gram while bucket refers to the list of locations of that Q-gram in the text. Here, T is first downsampled by a factor of M to give TM and a hash table H_{TMQ} is then constructed using contiguous Q-grams of TM. The bucket of a given Q-gram q is denoted by $H_{\text{TMQ}}[q]$.

The details of the preprocessing algorithm are given below. This is followed by an example on sample text.

Algorithm–A

(1) Downsampled the text T by a factor of M to yield TM.

(2) Generate the set of contiguous Q-grams for TM, namely, CQTM.

(3) For each Q-Gram in CQTM, consult the key field of the hash table H_{TMQ}.

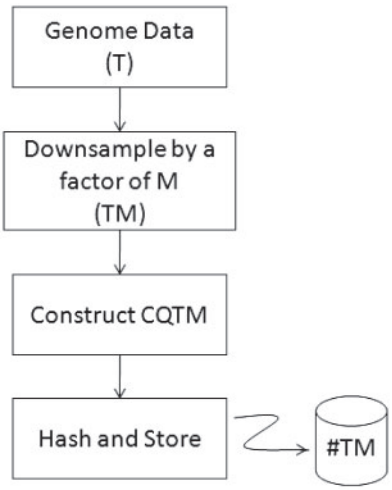

Fig. 1. Block diagram of the preprocessing stage.

Table 1. A sample Q-gram hash table

Key: Q(=3) gram	Bucket: Locations' list
agt	0, 3, 6
gta	1, 4, 7
tag	2, 5
taa	8
aac	9
aca	10

(4) If the Q-Gram exists in the key field, append the new position in the bucket $H_{\text{TMQ}}[Q\text{-gram}]$.

(5) Else, add a new Q-gram key and its position in the corresponding bucket in H_{TMQ}.

Example:
Consider the sequence T = 'acc gat tag aag ggt tta aga gtc tca acc aga cta agc'. For $M = 3$ and $Q = 3$, the result of the algorithm is given below.

(1) T3 = agtagtagtaaca.

(2) CQT3:{agt(0), gta(1), tag(2), agt(3), gta(4), tag(5), agt(6), gta(7), taa(8), aac(9), aca(10)}. Note that we have also mentioned the indices for each Q-gram in CQT3 in parentheses.

(3) Finally the H_{TMQ} is as given in Table 1.

3.2 Pattern-search stage

The idea behind the pattern-search algorithm is that given T and P, if P occurs in T at unknown locations, it is necessary that at least one downsampled polyphase PMi of P occurs in TM. Note that the reverse need not be true, that is, a certain PMi occurring in TM does not guarantee that P occurs in T. Therefore, we first mine for all occurrences of PMi in TM and search around the resulting indices for an exact match in T. Figure 2 presents the block diagram of the procedure.

The definitions of crucial variables are given below:

(1) PPAll: this is the array that holds the locations of exact matches for all polyphases of P. The elements of PPAll denoted by PPAll(n).

(2) PP: this is the array that holds the locations of exact matches of a particular polyphase of P. The elements of PP are denoted by PP(n).

The algorithm breaks each polyphase into non-overlapping Q-grams and bases its search on a successive refinement principle by employing beam pruning technique. That is, matches to the first non-overlapping Q-gram are first found. If these matches extend to the next Q-gram, then these locations are retained. This process is carried out for all the Q-grams in the given polyphase. At the end of this process, those locations where all the Q-grams match represent the locations where the polyphase PMi matches TM. These locations are then mapped to the original text T where the final search takes place. In this manner, the algorithm successively refines the search regions and thus speeds the search process. The algorithm is presented below followed by an example on a sample pattern.

Algorithm—B

(1) Generate PMi $0 \le i \le M-1$ from P using Polyphase Decomposition.

(2) Initialize PPAll = $\{\phi\}$ (an empty array).

(3) For each polyphase PMi, do (4) and (8).

(4) Generate NQPMi (set of non-overlapping Q-grams of polyphase PMi).

(5) PP = H_{TMQ}[NQPMi(0)] (this represents all positions in TM where an exact match is found for the first non overlapping Q-gram of PMi).

(6) For all NQPMi(j) ∈ NQPMi, $j > 0$ (the rest of the entries in NQPMi) do (7).

(7) Delete from PP, PP(k) such that
PP(k) + jQ ∉ H_{TMQ}[NQPMi(j)] NQPMi(j) should have occurred at location PP(k) + jQ if there was an exact match. But, if that is not the case, there is no chance of an exact match of polyphase PMi in TM at location PP(k). Hence, it must be pruned from the array PP.

(8) Append all PP (k) ∈ PP into PPAll. That is, make note of all locations of exact match of PMi in TM in the array PPAll that holds the locations of exact match for all PMis in TM.

(9) Translate PPAll(n) to corresponding location in original text T and verify exact match of P in T in that location.

Note that Steps (4) and (8) deal with extracting exact locations of PMi in TM. Step (7) is a beam-pruning method to prune out non-exact-match locations of PMi from the PP.

Example:
Consider P = 'aag ggt tta aga gtc tca'. Also M, Q and T as are the same as those used in the previous illustration. We show the steps of the algorithm for one of the polyphases: PM0. The results of the other two polyphases are presented in Step (8).

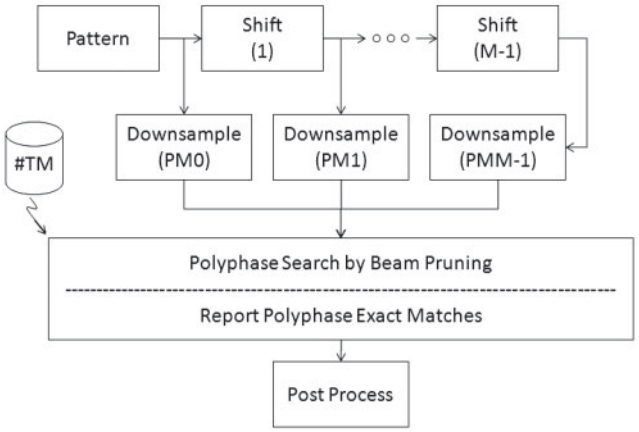

Fig. 2. Block diagram of the pattern-search stage.

(1) PM0 = 'agtagt' PM1 = 'agtgtc' PM2 = 'gtaaca'.

(2) PPAll = $\{\phi\}$ (an empty array)

(3) Iterating for PM0 for Steps (4) and (8).

(4) NQPM0 = {agt(0), agt(1)} (note that we have mentioned the indices of the Q-gram in NQPM0 in parenthesis).

(5) PP = H_{TNQ}[NQPM0(0)] = [0,3,6].

(6) Iterating over {agt(1)} for Step (7).

(7) Prune only PP(2).
Because PP(2)+1 × Q = 6+3 = 9 ∉ H_{TNQ}[NQPM0(1)].

(8) PPAll = {0,3},

(a) result of PM1: PP = $\{\phi\}$;

(b) result of PM2: PP = {7}.
Therefore, PPAll = {0,3,7}.

(9) Translating the locations of PPAll to those in T gives {0,9,19}. Note that we have considered the actual polyphase the exact match has come from. For example, as PPAll[2] = 7 come from the second polyphase, the mapping of this location to a location in T would be PPAll[2] × M − 2 = 19.
Exact matches are now verified starting from these locations in T. In this example, the exact match occurs at index 9 in T.

Note that if the length of PMi is not an integral multiple of Q, then certain trailing nucleotides will be missed by the NQPMi. One of the solutions in order to avoid this loss is to verify exact match of PMi in TM for all PP(n) just before Step (8).

4 ALGORITHM ANALYSIS

We now present the space and time complexity analysis of the proposed method.

4.1 Space complexity of hash table

From Algorithm A, because every overlapping Q-gram in TM contributes to an entry in the hash table, the algorithm's space complexity is $O(L_{TM})$ where the length of TM, $L_{TM} = L_T/M$ due to downsampling. Also, because constructing the hash table is a sequential process, its time complexity is $O(L_{TM})$.

Note that there are two types of memory in a computer—main memory, which is fast, costly and scarce and the secondary memory, which is slow, cheap and abundant. An efficient design of an algorithm is an optimal balance between its speed and its main memory requirements. In our design, we have retained a smaller, down-sampled version of the text T in the main memory, which we consider for space complexity calculations. For the purposes of exact match verification in Step (9) of Algorithm B, the original text T can reside in the cheaper secondary memory, and relevant sections (with length $<< L_T$) can be paged into the main memory when required. Also note that if we set $M = 1$, the H_{TMQ} reduces to its naïve version and its main memory requirements are then $O(L_T)$.

4.2 Time complexity of pattern search

We will now discuss the time complexity for pattern search (Algorithm B). In our analysis, we model the base distribution as an iid process and following uniform distribution. This is reasonable over large databases because it has been shown that DNA sequences at best have weak long-range correlations (Bernaola *et al.*, 2002). The complexity of processing a particular polyphase as described in Steps (4) and (8) is as given below.

Step 4 is the generation of the NQPMi list. This step which generates non overlapping Q-grams is computationally simple and its effect on the search complexity can be neglected.

Step 5 is a look up from the hash table. Because this step is only a main memory lookup (Cormen *et al.*, 1990), it contributes $O(1)$ to the search complexity. Also, It follows from our assumption that the distribution of bases is uniform and iid, the probability of the existence of any given Q-gram = $1/4^Q$. Thus, the expected number of any Q-grams in TM = $L_{TM} \times (1/4^Q)$. That is, the expected number of matches for the first Q-gram in any given polyphase is $L_{TM}/4^Q$.

Step 7 is the beam pruning procedure, where entries from the array PP are removed based on the entries in the bucket H_{TMQ}[NQPMi(j)]. This translates to traversal of both arrays, namely, PP and H_{TMQ}[NQPMi(j)]. The complexity of this step then depends on the lengths of each array.

- The bucket: following the above explanation, the bucket has an average of $L_{TM}/4^Q$ entries.
- The PP Array: note that for any j, the PP Array holds the locations of a string of length $j \times Q$. Therefore, the expected length of the PP Array for any j is $L_{TM}/4^{jQ}$.

As a result, employing binary search, the complexity of the beam pruning procedure for a single polyphase is $O((L_{TM}/4^Q)\log(L_{TM}/4^Q))$.

Also, note that Steps (1), (2) and (8) are computationally simple and their contribution to the search complexity can be neglected.

Now, the complexity of the algorithm until Step (9) is the complexity of beam pruning procedure for all M polyphases = $O(M(L_{TM}/4^Q)\log(L_{TM}/4^Q))) = O((L_T/4^Q)\log(L_{TM}/4^Q))$. The complexity of Step (9) is negligible as its of order $O(L_P)$, $L_P << L_T$. Thus the overall time complexity for the search procedure:

$$O((L_T/4^Q)\log(L_T/M4^Q)).$$

A comparison of complexities of various search algorithms are given in Table 2.

As can be seen from the table, the proposed method has superior complexity as compared to most existing methods. Also, while

Table 2. Comparison of theoretical complexities of various pattern/homology search algorithms

Search algorithm	Space complexity	Time complexity
BLAST[a]	$O(L_P L_{T'})$[b]	$O(L_P L_{T'})$
FASTA[a]	$O(L_P L_{T'})$	$O(L_P L_{T'})$
Finite automaton	$O(L_P)$	$O(L_T)$
Knuth Morris Pratt	$O(L_P)$	$O(L_T)$
Suffix tree based	$O(L_T)$	$O(L_P)$
BWA-SW	$O(L_P)$	$O(L_P^{0.628} L_T)$
SSAHA	$O(L_T)$	$O((L_T/4^Q)\log(L_T/4^Q))$
Proposed	$\boldsymbol{O(L_T/M)}$	$\boldsymbol{O((L_T/4^Q)\log(L_T/M4^Q))}$

[a]Methods also yield approximate matches.
[b]$L_P << L_{T'} << L_T$.
The bold line corresponds to the complexity of the proposed algorithm.

Table 3. Hash table size for $M-Q$ combinations (MB)

	$Q=8$	$Q=9$	$Q=10$	$Q=11$
$M=7$	117.0	118.3	120.3	129.1
$M=15$	55.0	55.3	57.6	65.0
$M=23$	35.9	36.4	38.4	45.3
$M=31$	26.7	27.2	29.2	35.2
$M=39$	21.3	21.7	23.7	28.9

the suffix-tree approach would possibly have the advantage of a slightly better time complexity when compared to the proposed method, this advantage could be offset by its huge memory requirements. The space savings would be particularly useful in processing huge sequences such as whole genomes. Also, note that because the algorithm works on polyphase decomposition with little inter-dependency between each polyphase, it lends itself to easy parallelization thereby speeding up its operation.

4.3 Experimental analysis

The algorithm was implemented using Python. The text T considered is the Human Chromosome 1 ($L_T = 250$ M bases) (Genome Reference Consortium, UCSC) the pattern P is randomly chosen segment of 300 bases ($L_P = 300$) from T. All experiments were conducted on a PC with a 2 GB RAM and Intel 2.4 GHz Quad core processor.

The variation of the size of the hash table with varying M and Q is presented in Table 3. It can be seen that the size of the hash table decreases with increasing M as expected. Also, it can be seen that the size of the hash table increases slightly with increasing Q.

The variations in time taken to process the pattern and generate a list of locations for post processing are presented in Table 4. Note that the combination of parameters M and Q must satisfy the condition $MQ < L_P$. Otherwise, at least one of the polyphases will contain $< Q$ bases, and thus cannot be looked up through the hash table. The blocks whose $M-Q$ combination is infeasible for $L_P = 300$ are marked with '–'. It can be seen that as M and Q increases, the search time decreases as expected.

Also, as M increases, the number of potential exact matches that must be post-processed also increases. Table 5 provides this data.

Table 4. Search times for various $M-Q$ combinations (in micro seconds)

	$Q=8$	$Q=9$	$Q=10$	$Q=11$
$M=7$	1830	277	27	30
$M=15$	288	33	7	3
$M=23$	105	9	1.4	0.5
$M=31$	108	8	–	–
$M=39$	–	–	–	–

Table 5. Number of matches to be post processed versus M

	$M=7$	$M=15$	$M=23$	$M=31$
Number of matches	1	1	2	330

Thus, it can be seen from the data presented that a large M results in a smaller hash table, but also generates larger number of potential matches that must be post processed. Also, a larger Q speeds up the polyphase search, but demands larger hash table size. Therefore, values M and Q must be carefully chosen.

For example, for setting parameters M and Q for a pattern of length ~300, Table 3 inspires us to use a highest value of $M = 39$ (because this gives the smallest hash table). However, the numbers in table Table 4 suggest that it would be reasonable to select the higher values of Q and set $M \sim 23$ (because higher $M-Q$ combinations are either slow or infeasible) and further consultation with Table 5 indicates that a combination of $M = 23$ and $Q = 11$ is practical (because the number of matches that must be post processed are near minimal). Thus an optimal choice of parameters would be $M = 23$ and $Q = 11$, which requires 45.3 MB for the hash table (Text size = 250 M bases). Also, with respect to the pattern (Pattern size = 300 bases) related searching time, 0.5 ms are required to generate the list of exact matches of polyphases in the downsampled text. Another 1 ms is required to verify if the exact polyphase match translates to exact match in text. Thus a total of 1.5 ms were required for mining exact matches.

5 CONCLUSION

In this article, we presented a method for fast exact sequence search that relies on downsampling and polyphase representations to expedite the search process. We also computed the complexity of the algorithm and showed it to be better than existing methods. Because the proposed method uses polyphase representations, and because searching for exact matches in multiple polyphases does not have any data or functional inter-dependency, the algorithm can be parallelized. This would further reduce time complexity. Implementation of a parallel version of the algorithm will be a topic of further work.

The proposed algorithm addresses the problem of finding exact matches to a substring. Our future research will extend the utility of this algorithm to finding approximate matches.

Conflict of Interest: none declared.

REFERENCES

Altschul,S.F. *et al*. (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Bernaola-Galvan,P. *et al*. (2002) Study of statistical correlations in DNA Sequences. *Gene*, **300**, 105–115.

Charras,C. and Lecroq,T. (2004) *Handbook of Exact String Matching Algorithms*. Kings College, London Publications, London.

Cormen,T. *et al*. (1990) Hash tables. In *Introduction to Algorithms*, 2nd edn. McGraw Hill, MIT Press, Cambridge, MA, pp. 221–245.

Eric,V.C. *et al*. (2009) Cetuximab and chemotherapy as initial treatment for metastatic colorectal cancer. *N. Engl. J. Med.*, **360**, 567–578.

Genome Reference Consortium, UCSC. (2009) http://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/.

Gusfield,D. (1997) *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, MA.

Knuth,D. *et al*. (1977) Fast pattern matching in strings. *SIAM J. Comput.*, **6**, 323–350.

Kurtz,S. *et al*. (2004) Versatile and open software for comparing large genomes. *Genome Biol.*, **5**, R12.

Lecroq,T. (2007) Fast exact string matching algorithms. *Inf. Process. Lett.*, **102**, 229–235.

Li,H. and Durbin,R. (2010) Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, **26**, 589–595.

Lipman,D.J. and Pearson,W.R. (1985) Rapid and sensitive protein similarity searches. *Science*, **227**, 1435–1441.

Ma,B. *et al*. (2002) PatternHunter: faster and more sensitive homology search. *Bioinformatics*, **18**, 440–445.

Melichar,B. (1995) Approximate string matching by finite automata. Computer analysis of images and patterns. *LNCS*, **970**, 342–349.

Ning,Z. *et al*. (2001) SSAHA: a fast search method for large DNA databases. *Genome Res.*, **11**, 1725–1729.

Pearson,W.R. and Lipman,D.J. (1988) Improved tools for biological sequence comparison. *Proc. Natl Acad. Sci.*, **85**, 2444–2448.

Vaidyanathan,P.P. (1993) *Multirate Systems and Filter Banks*. Prentice-Hall, Englewood Cliffs, NJ.