EMMA: An Efficient Massive Mapping Algorithm Using Improved Approximate Mapping Filtering

Xin ZHANG[1], Zhi-Wei CAO[2], Zhi-Xin LIN[3], Qing-Kang WANG[1]*, and Yi-Xue LI[4]*

[1] *Institute of Micro/Nano Science and Technology, Shanghai Jiaotong University, Shanghai 200030, China;*
[2] *Shanghai Center for Bioinformation Technology, Shanghai 200235, China;*
[3] *College of Life Science and Technology, Shanghai Jiaotong University, Shanghai 200030, China;*
[4] *Bioinformation Center of Shanghai Institutes for Biological Sciences, Chinese Academy of Sciences, Shanghai 200031, China*

Abstract Efficient massive mapping algorithm (EMMA), an algorithm on efficiently mapping massive cDNAs onto genomic sequences, has recently been developed. The process of mapping massive cDNAs onto genomic sequences has been improved using more approximate mapping filtering based on an enhanced suffix array coupled with a pruned fast hash table, algorithms of block alignment extensions, and *k*-longest paths. When compared with the classical BLAT software in this field, the computing of EMMA ranges from two to forty-one times faster under similar prediction precisions.

Key words cDNA mapping; maximal exact match; enhanced suffix array; pruned fast hash table; extension algorithm

Mapping cDNAs (e.g., mRNAs and ESTs) onto genomic sequences has become a common and potentially powerful technique in the field of genome research. The resulting alignments are often used in fields of gene finding, alternative splicing prediction and single nucleotide polymorphism studies [1–5]. However with more and more sequences accumulated, the mapping computation has become more and more expensive for most researches. Faster cDNA-genome mapping software is always highly expected, some of which are SIM4 [6], SPIDEY [7], GENESEQER [8,9], BLAT [10], SQUALL [11], GMAP [12] and ESTMAPPER [13]. Most of these algorithms are derived from BLAST [14] and featured as a common four-phase framework: first, finding exact matches longer than given size; second, extending each exact match pair to both directions by an ungapped alignment until the score drops significantly; third, linking the extended matches together to outline the plausible splice patterns; and finally, refining the outlined splicing patterns to produce precise mapping alignments.

Despite their various implementation methodologies, the first step in most existing algorithms is implemented by computing the word pairs (exact matches of fixed length) of the cDNA and the genome. To improve the computation speed, the genome is pre-processed by indexing all of its words in a table. Early algorithms like SIM4 and SPIDEY use simple look-up-table (LUT) functions to store these genome words, which require $O(4^w)$ memory, where w is the word size. For large w values, the memory required for LUT would become impractical for normal computers. To address this problem, modern cDNA mapping algorithms including BLAT and GMAP mostly use hash tables and only consider non-overlapped words, which not only reduces the size of word table but also improves the computation speed by thousands of times without significant loss of precision [10,12].

The idea of the word-based method is simple enough but requires additional treatment to concatenate neighboring word pairs into longer ones. It has been evaluated that such concatenation processes take up to 18% of the entire

Received: August 26, 2006 Accepted: October 10, 2006

This work was supported by a grant from the Major State Basic Research Development Program of China (No. 2004CB720103)

*Corresponding authors:

Qing-Kang WANG: Tel/Fax, 86-21-62933290; E-mail, wangqingkang@sjtu.edu.cn

Yi-Xue LI: Tel, 86-21-64836199; Fax, 86-21-64838882; E-mail, yxli@sibs.ac.cn

DOI: 10.1111/j.1745-7270.2006.00237.x

executing time [15]. Recently, Wu *et al.* proposed a better solution based on maximal exact matches (MEMs), requiring only one sixth of the computation time of BLAT under a similar precision [13]. An MEM is an exact match that can not be extended on either side without creating a mismatch. It is in nature the concatenated neighboring words. For this reason, MEMs are non-overlapping along diagonals of alignment matrices. MEMs of the cDNA and the genome can be efficiently computed by converting a genome into a suffix tree, and comparing it with the cDNA sequence. However, the suffix tree often encounters the problem of intrinsic large memory space [16]. To solve this problem, Abouelhoda *et al.* proposed a space-economical data structure called enhanced suffix arrays (ESAs) which can be used to efficiently solve all of the problems that are usually solved by a bottom-up or a top-down traversal of the suffix tree [17].

After the exact matches are computed, existing alignment algorithms generally make the traditional BLAST-like ungapped extension (also called gap-free extension) to determine if an exact match is likely to form part of a high-score alignment segment [14]. Since no gap is considered in computation, the resulting extended matches may not approximate the local alignments well. Recently, Cameron *et al.* proposed a semi-gapped extension algorithm, which makes better extended matches with gap insertions allowed at limited locations [15].

The majority of cDNA mapping algorithms are optimized for the genome-scale alignment. Some studies, however, require efficiently mapping massive cDNAs onto a relative short genomic sequence. For example, in order to predict alternative splicing isoforms of a gene, millions of cDNAs have to be tried to map onto the corresponding genomic sequence. In our previous work, a pipeline named ASA (alternative splicing assembler) was developed to resolve this problem based on BLAST [18]. Nevertheless, this method still spends more time searching redundant databases like dbEST due to the unnecessary computation of many short local alignments.

As the efforts to develop faster mapping algorithms never cease, efficient massive mapping algorithm (EMMA) has been set up. EMMA is a more efficient algorithm using an approximate mapping filtering method based on many new techniques including ESAs, pruned fast hash tables, block extensions and the *k*-longest-path algorithm. The performance of EMMA is evaluated by comparing it with BLAT using the same dataset. Comparison results show that it is about 2 to 41 times as fast as BLAT when reaching similar prediction precision. Further performance comparison using a different dataset shows that EMMA reaches precisions comparable with other programs, and is at least four times as fast as them.

Materials and Methods

Approximate mapping filtering

Approximate mapping filtering is a highly efficient method of searching cDNAs that could be mapped onto the genomic sequence. False local alignments impossible to be part of any plausible alignment are filtered before precise and expensive dynamic programming computation. The approximate mapping filtering consists of three steps: compute MEM based on ESA and pruned fast hash table (PFHT), extend MEMs using block extension, and link extended MEM.

Compute MEM based on ESA and PFHT

EMMA uses space-economic ESA structure to compute MEM matches. All cDNAs are first concatenated and converted into ESA. Then each suffix of the genomic sequence is compared to the preprocessed cDNA database. The constructed ESA is divided into buckets where each consists of suffixes with distinct common prefixes. These buckets are then indexed by their respective distinct common prefix into a PFHT [19], which uses the extended Bloom filter [20] and universal hash function [21] to improve the lookup performance. The preprocess only needs to be carried out once per database, since the ESA is a flat array and can be easily stored and reloaded for future computation.

Let $G(i)$ be the ith suffix of a genomic sequence G. The right maximal exact matches (only have mismatches at the right end) are then computed by first comparing $G(i)$ to the PFHT to get the corresponding bucket, and then traversing along $G(i)$ down the part of the ESA in the bucket. The resulting right maximal exact matches are removed if they have matches at the $(i-1)$th base. Therefore each remaining maximal exact matche is bounded with mismatches at both ends. By carrying out this computation for each possible $G(i)$, all MEMs of the cDNAs and the genomic sequence will be revealed.

Like word-based algorithms, using shorter MEMs will produce more precise results, and using longer ones will greatly reduce the computation time. To balance the precision and efficiency of the alignment computation, the minimal length of initial MEMs (T_w) must be carefully determined. The T_w value is equivalent to the word size (W) that can be determined in the BLAST algorithm, thus they can be determined in the same way. If no MEMs of a cDNA and a genome sequence are longer than the preset threshold T_w, the cDNA can not be mapped onto the genome sequence under that criterion.

Block extension algorithm

We propose a greedy algorithm, so-called block extension algorithm, to extend MEMs or maximal exact matches into approximate local alignments (**Fig. 1**). The block extension algorithm is performed by iteratively extending a MEM by the optimal global alignment (*BestGlobalAlign*) or the best extension alignment (*BestPrefixAlign*) of the following *k*-base pair (called a block) until the score drops significantly. Since gaps at different blocks are treated independently, only a non-affine gapping scheme shall be used in block extension.

```
procedure $(d_{best}, i_b, j_b) \leftarrow BlockAlign(S_1, S_2)$
    $BestScore \leftarrow 0, score \leftarrow 0$
    $d_{best} \leftarrow 0, i_b \leftarrow 0, j_b \leftarrow 0$
    $d_{global} \leftarrow 0, i \leftarrow 0, j \leftarrow 0$
    while $i \le |S_1| - k$ and $j \le |S_2| - k$ do
        $d_G \leftarrow BestGlobalAlign(S_1(i : i+k-1), S_2(j : j+k-1))$
        $(d_L, m, n) \leftarrow BestPrefixAlign(S_1(i : i+k-1), S_2(j : j+k-1))$
        $d_{best} \leftarrow d_{best} + d_L$
        $i_b \leftarrow i_b + m, j_b \leftarrow j_b + n$
        $d_{global} \leftarrow d_{global} + d_G$
        $i \leftarrow i + k, j \leftarrow j + k$
        if $Score(d_{best}, i_b, j_b) > BestScore$ then
            $BestScore \leftarrow Score(d_{beset}, i_b, j_b)$
        end if
        $score \leftarrow Score(d_{global}, i, j)$
        if $BestScore - score > X$ then
            $break$
        end if
    end while
end procedure
```

Fig. 1 **Block extension algorithm**

For small *k* values, the optimal alignment scores of all possible pairs of length *k* can be pre-computed and indexed by their corresponding sequence pairs into a table (called a block table). With Zhang's greedy algorithm where the award for a match (*mat*) and penalties for a mismatch (*mis*) and gap (*ind*) shown in **Equation 1**, the score of an alignment can be computed from the edit distance and alignment lengths by **Equation 2** [22].

$$ind = mis - mat/2 \qquad 1$$

$$Score(d,i,j) = (i+j) \times mat/2 - d \times (mat - mis) \qquad 2$$

This implies that each pre-computed alignment could be represented as a triplet *(d,i,j)*, where *d*, *i* and *j* are the edit distance and the alignment lengths, respectively. Therefore, the block table needs 4^{2k+1} additional memory to store triplets of *BestGlobalAlign* and *BestPrefixAlign* of all possible *k*-length sequence pairs. Each triplet is computed independently with at most *O(k)* time and *O(k)* space. Therefore, the time and space complexities of precomputing the block table are $O(k4^{2k+1})$ and *O(k)*, respectively. According to our experiments, a *k* value of 5 is sufficient for most cases. In this case, the block table needs about 4M memory. Let *n* be the average alignment size. With the pre-computed block table, the block extension can be performed in *O(n/k)* time, which is comparable with the ungapped extension.

One major advantage of block extension over the ungapped extension and semi-gapped extension is that the block extension allows gap insertions in the sequence at any location. Thus the extension can stride over small insertions and deletions. This feature enables extended MEMs computed by block extension to approximate the alignment segments much better.

Link extended MEMs

To link extended MEMs into approximate mapping, a graph is constructed, where each vertex except one source vertex represents an extended MEM. Two vertices are connected by a directed edge if they are possible to be parts of the same mapping alignment. The direction of each edge indicates the order of its end vertices in a mapping alignment, and its weight is set to the score of the vertex it points to. It is obvious that an extended MEM graph is actually a weighted directed acyclic graph, where each path starting from the source vertex represents a possible approximate mapping whose score equals to the length of the weighted path.

Instead of the longest-path algorithm used by most cDNA mapping algorithms, we choose the *k*-longest-path algorithm [23] to compute best *k* extended MEM chains. The path computation can be performed in *O(m+kn)* time, where *m* and *n* are the number of edges and vertices in the graph, respectively. Among the *k* best chains, only those that are not sub-paths of any other ones and have lengths larger than a preset threshold T_c are chosen as approximate mappings of the cDNA on the genomic sequence.

Choosing multiple chains enables production of more robust results but requires more computation time at the same time. The robustness and the speed of the linking process can be balanced by adjusting the value of *k*. In our experience, a *k* value of 3 is proper for most cases.

Precise alignment

After the approximate mapping filtering process, we have the approximate mappings of each cDNA on the genomic sequence. To compute the precise alignments, all of these approximate mappings are re-computed by a dynamic programming method.

EMMA adopts GMAP's sandwich dynamic programming algorithm to compute the precise EST-genome alignments. Exon-intron boundaries are adjusted to meet canonical splice sites, i.e., AG/GT, AC/GT and AT/AC at donor/acceptor sites [24,25].

Efficient massive mapping algorithm

By finding and extending maximal exact matches, our mapping algorithm filters MEMs by the block extension algorithm. Then the extended MEMs are linked into approximate mappings. In the end, dynamic programming is performed to compute the precise mapping alignments. The overview of our algorithm is as follows:

I Preprocess

1.1 Concatenate cDNAs from a database into a long sequence, and convert the sequence into ESA indexed in a PFHT.

1.2 Given a block size k_b, compute the optimal global alignments of all possible k_b-base pairs and store their scores in the block table.

II Approximate mapping filtering

2.1 Compute MEMs longer than T_w based on the pre-constructed ESA and PFHT.

2.2 Extend each MEM by the block extension algorithm. The extended MEMs with scores larger than T_b are selected for the next process.

2.3 Link the extended process by the k-longest-path algorithm. Approximate mappings of each cDNA are selected from the linked MEM chains.

III Re-compute the approximate mappings by the sandwich dynamic programming method. Exon/intron boundaries are adjusted to meet the canonical splicing sites.

Dataset

Three hundred and ninety-seven genes (14 million bases) with annotated RefSeq [26,27] mRNAs in the well-annotated human chromosome 22 are selected from GenBank [28] and serve for genes to be investigated. A total of 34,908 ESTs (20 million bases) are selected by searching the 397 genes against dbEST using NCBI BLAST software (word size=11, expect=1e-10, no low complexity filter).

Considering the sequencing errors, 30 bases of unmapped regions are allowed at both ends of each EST. The mapping alignments of each EST on the chromosome 22 are recorded as references for the performance comparison.

Results

To evaluate the efficiency of EMMA, a tool has been developed by C language on a LENOVO®R410 SMP server with 4 Intel® Itanium® 2 processors and 32GB RAM, and compared to BLAT using the same set of dataset prepared in the "Methods and Materials".

Evaluate block extension

With a 260-base sequence selected from human exons, two simulation experiments are carried out to evaluate the precision of block extension. In each experiment, 10 datasets are prepared of which each contains 2000 samples generated by randomly and independently introducing errors including mismatches, insertions and deletions into the 260-base sequence using the same error rate. For each pair of sample/original sequence, the approximate mapping filtering is performed using ungapped extension and block extension, respectively, and their precise alignment is also computed by dynamic programming. In experiment (a), the error rates for the 10 datasets range from 0.01 to 0.1 stepping 0.01, and T_w for both methods are set to 20. In experiment (b), the error rate is set to 0.05 for all datasets, and T_w ranges from 15 to 24 for the 10 datasets. The same scoring scheme is used in each alignment computation.

The accuracy of an extension method for a dataset is defined as the ratio of the average approximate alignment score to the average precise alignment score over all samples in the dataset. According to the definition, an extension algorithm with a higher accuracy value is expected to produce better approximate alignments. **Fig. 2** shows the error rate-accuracy curves of block extension and ungapped extension in the two experiments.

It can be seen from **Fig. 2(A)** that even with an error rate up to 0.1, the accuracy of block extension still reaches a high value over 0.98, while the accuracy of block extension significantly drops to 0.75. It seems that the approximate alignments computed by the block extension algorithm are always very close to the precise ones.

Fig. 2(B) shows the influence of minimal MEM size (T_w) on the accuracy of both extension algorithms. As T_w increases, the accuracy of block extension remains at a high level close to 1, while the accuracy of the ungapped extension drops significantly. It is not surprising that the accuracy of the ungapped extension decreases with T_w,

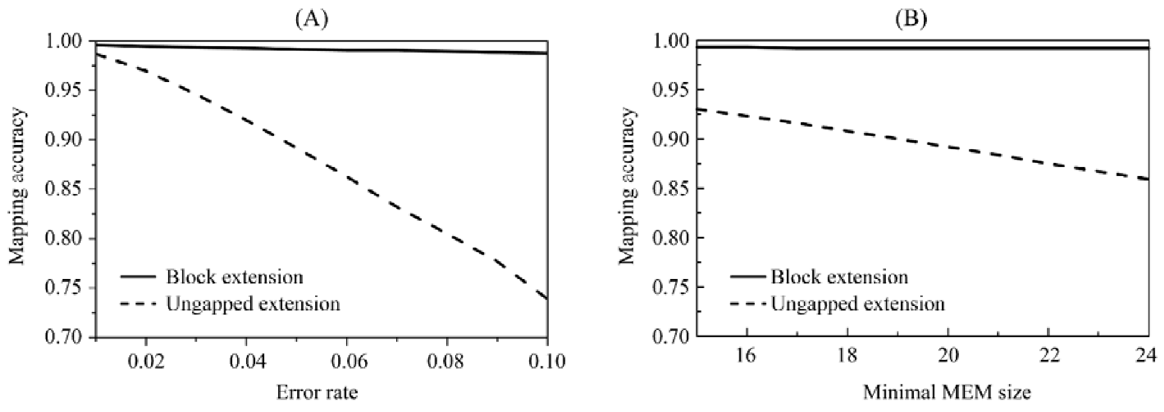

Fig. 2 **The accuracy curves of block extension algorithm and ungapped extension algorithm**

because the requirement of large exact matches makes some parts of the alignment unreachable. However, the block extension seems to be able to deal with these parts.

The advantage of the block extension may come from the fact that it considers gaps in computation and does not limit their locations. The feature enables the block extension process to stride over small gaps, which is impossible for the ungapped extension. Block extension is expected to produce approximate alignments very close to the precise ones, which means more false alignments would be filtered, and the computation in the linking process can be greatly reduced.

Parameterize PFHT

Let k_u be the number of universal hash functions. To parameterize PFHT, we search the 397 genes against randomly selected 34,908 ESTs with different k_u values. Executing time and corresponding k_u values are recorded and shown in **Fig. 3**.

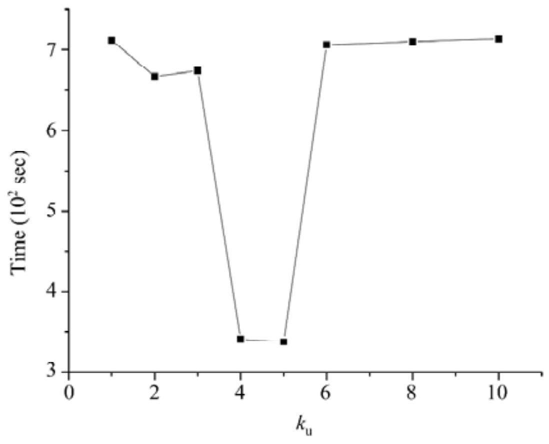

Fig. 3 **Executing time of the efficient massive mapping algorithm (EMMA) when using different k_u values for pruned fast hash table (PFHT)**

It is seen that EMMA reaches its best performance when *k* is set to 4 or 5. The optimal size for the PFHT is *nk*/*ln2*, where *n* is the number of items to be indexed. With smaller *k* values, the size of PFHT decreases, which increases the collision chance and decreases the indexing speed. With larger *k* values, PFHT may suffer from the poor memory locality due to its large size, and the increased computation for hash functions.

Evaluate EMMA

To evaluate the performance of EMMA, the 397 genes are searched against the 34,908 ESTs by EMMA with T_w values ranging from 10 to 20. As a comparison, BLAT (version 3.2) is downloaded and applied to the same dataset using different word sizes ranging from 8 to 12. The parameter *N* of BLAT is set to 2 for using a two-hit scheme. The resulting mappings computed by either method are compared to the corresponding reference ones. The mapping precision is then defined to be the percentage of the correctly mapped ESTs. **Tables 1** and **2** list the execution time and precision of results using EMMA and BLAT, respectively. **Table 3** compares the space used by EMMA and BLAT. **Fig. 4** compares the precision-time curves between EMMA and BLAT.

One may notice that the running time does not decrease monotonically as T_w increases. According to our analysis, the fluctuation may come from the influence of the locality of PFHT memory reference. The space requirement of PFHT increases as T_w increases. The poor locality of such a huge memory space can significantly lower the indexing

Table 1 Search time and precision of efficient massive mapping algorithm (EMMA) using various minimal maximal exact match (MEM) size configurations

T_w	Time (s)	Precision
10	3907.0	93.2%
11	1212.9	93.0%
12	714.5	92.9%
13	287.5	92.7%
14	332.1	92.5%
15	177.4	92.2%
16	202.1	91.8%
17	170.7	91.5%
18	120.7	91.0%
19	110.5	90.6%
20	117.4	90.2%

Table 2 Search time and precision of BLAT using various minimal word size *W* configurations

W	Time (s)	Precision
8	24729	93.6%
9	15509	93.2%
10	11375	92.3%
11	2683	91.4%
12	1917	90.3%

Table 3 Physical memory used by BLAT and the efficient massive mapping algorithm (EMMA)

Group	Memory (MB)
BLAT	24–212
EMMA	338–604

efficiency of PFHT. With certain parameter settings, the penalty of this poor memory locality may exceed the efficiency award brought by the increased T_w. In such cases, the running time will increase although T_w increases.

It can be seen from **Fig. 4** that the mapping precision of EMMA is between 0.9 and 0.932, which is comparable with BLAT. However, the executing time for BLAT rises rapidly with the mapping precision increases, while the time for EMMA remains within a very small value and almost stable until the mapping precision goes beyond 0.93. At any mapping precision, BLAT needs much more executing time than that of EMMA, especially for the mapping precisions ranging from 0.915 to 0.93.

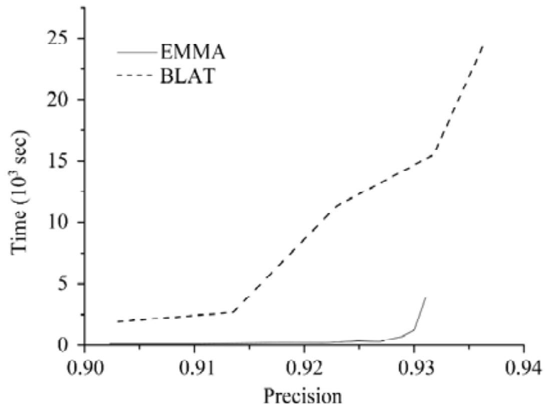

Fig. 4 Precision-time curves of efficient massive mapping algorithm (EMMA) and BLAT

Fig. 5 shows the ratio of the executing speed of EMMA to that of BLAT. The curve shows that EMMA is about 25 times faster than BLAT on average, 41 times at most, and two times at least.

To compare the efficiency of EMMA with other cDNA mapping algorithms or alignment algorithms, the results of the performance comparison evaluated using ESTMAPPER, SQUALL, BLAT, SPIDEY, SIM4 and BLAT [13] and GMAP and BLAT [12] are used to compare with our performance evaluation. The comparison is based on a hypothesis that relative performances of these algorithms do not vary greatly when using different datasets. The comparison results are shown in **Table 4**. The preliminary comparison results show that EMMA is expected to be more efficient than existing cDNA mapping algorithms

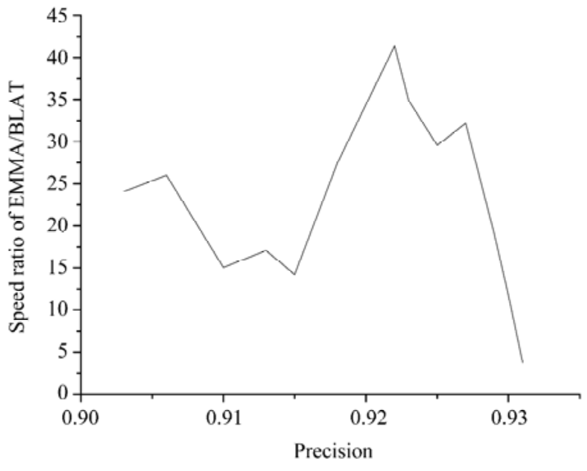

Fig. 5 Speed ratio of efficient massive mapping algorithm (EMMA) and BLAT under different prediction precision

Table 4 Performance comparison of some cDNA mapping algorithms

Algorithm	Relative speed	Relative space	Precision
Sim4	1.5e-3	4.2	99.6%
Spidey	6e-3	2.1	90.8%
BLAT	1.0	1.0	89.7%
Squall	1.4	1.0	90.1%
ESTmapper	6.0	3.4	99.9%
GMAP	6.0	1.2	99.3%
EMMA	25.0	4.7	93.6%

EMMA, efficient massive mapping algorithm; GMAP, genomic mapping and alignment program.

when reaching similar precisions.

Discussion

Benefiting from the advanced technologies including ESA and pruned fast hash table, the computation speed of EMMA has been greatly improved. The approximate mapping filtering method avoids the expensive computation on the abundant false local alignments. All of these factors have made EMMA a competitive tool for mapping massive cDNAs onto a genomic sequence.

The advantage of EMMA over existing cDNA mapping algorithms is shown in two ways. On one hand, for massive mapping performed with similar parameter setting, EMMA generally needs much less time than other cDNA mapping algorithms. This surely benefits most time-critical applications that require the mapping process to be performed in a very short time. On the other hand, for people who expect accurate mapping results and can afford enough computation time, EMMA can provide much better mapping results than other cDNA mapping algorithms within the similar running time.

EMMA improves the computational speed by loading a preprocessed cDNA database and the block table into the memory. This requires more space than algorithms using a simple hash table, (e.g., BLAT). In this experiment, EMMA takes 604MB memory at most, and BLAT takes only 212MB. The large space requirement of EMMA limits its usage on some commodity hardware with less power, where GMAP could be a better choice in this case.

In this paper, we presented EMMA, an efficient algorithm for massive mapping of cDNAs onto gene sequences based on approximate mapping filtering. Comparison results using a dataset of 397 genes and 34,908 ESTs show that it is about 2 to 41 times as fast as BLAT when reaching similar prediction precision. Further performance comparison using different dataset shows that EMMA reaches precisions comparable with other programs, and is at least four times as fast as them.

References

1 Jiang J, Jacob HJ. EbEST: An automated tool using expressed sequence tags to delineate gene structure. Genome Res 1998, 8: 268–275

2 Irizarry K, Kustanovich V, Li C, Brown N, Nelson S, Wong W, Lee CJ. Genome-wide analysis of single-nucleotide polymorphisms in human expressed sequences. Nat Genet 2000, 26: 233–236

3 Zavolan M, van Nimwegen E, Gaasterland T. Splice variation in mouse full-length cDNAs identified by mapping to the mouse genome. Genome Res 2002, 12: 1377–1385

4 Modrek B, Lee C. A genomic view of alternative splicing. Nat Genet 2002, 30: 13–19

5 Bonizzoni P, Rizzi R, Pesole G. Computational methods for alternative splicing prediction. Brief Funct Genomic Proteomic 2006, 5: 46–51

6 Florea L, Hartzell G, Zhang Z, Rubin GM, Miller W. A computer program for aligning a cDNA sequence with a genomic DNA sequence. Genome Res 1998, 8: 967–974

7 Wheelan SJ, Church DM, Ostell JM. Spidey: A tool for mRNA-to-genomic alignments. Genome Res 2001, 11: 1952–1957

8 Schlueter SD, Dong Q, Brendel V. GeneSeqer@PlantGDB: Gene structure prediction in plant genomes. Nucleic Acids Res 2003, 31: 3597–3600

9 Usuka J, Zhu W, Brendel V. Optimal spliced alignment of homologous cDNA to a genomic DNA template. Bioinformatics 2000, 16: 203–211

10 Kent WJ. BLAT—the BLAST-like alignment tool. Genome Res, 2002, 12: 656–664

11 Ogasawara J, Morishita S. Fast and sensitive algorithm for aligning ESTs to human genome. Proc IEEE Comput Soc Bioinform Conf 2002, 1: 43–53

12 Wu TD, Watanabe CK. GMAP: A genomic mapping and alignment program for mRNA and EST sequences. Bioinformatics 2005, 21: 1859–1875

13 Wu X, Lee W, Tseng C. ESTmapper: Efficiently aligning DNA sequences to genomes, parallel and distributed processing symposium. Proc 19th IEEE International 2005

14 Altschul SF, Gish W, Miller W, Myers EW , Lipman DJ. Basic local alignment search tool. J Mol Biol 1990, 215: 403–410

15 Cameron M, Williams HE, Cannane A. Improved gapped alignment in BLAST. IEEE/ACM Transactions on Computational Biology and Bioinformatics 2004, 1: 116–129

16 Ma B, Tromp J, Li M. PatternHunter: Faster and more sensitive homology search. Bioinformatics 2002, 18: 440–445

17 Abouelhoda M I, Kurtz S, Ohlebusch E. Replacing suffix trees with enhanced suffix arrays. J Discrete Algorithms 2004, 2: 53–86

18 Hui L, Zhang X, Wu X, Lin Z, Wang Q, Li Y, Hu Gl. Identification of alternatively spliced mRNA variants related to cancers by genome-wide ESTs alignment. Oncogene 2004, 23: 3013–3023

19 Song H, Dharmapurikar S, Turner J. Fast hash table lookup using extended Bloom filter: An aid to network processing. Proceedings of ACM SIGCOMM 2005

20 Bloom B. Space/time trade-offs in hash coding with allowable errors. Communications of ACM 1970, 13: 422–426

21 Carter L, Wegman M. Universal class of hash functions. J Comput System Sci 1979, 18: 143–154

22 Zhang Z, Schwartz S, Wagner L, Miller W. A greedy algorithm for aligning DNA sequences. J Comput Biol 2000, 7: 203–214

23 Eppstein D. Finding the *k* shortest paths. SICOMP 1998, 28: 652–673

24 Burset M, Seledtsov IA, Solovyev VV. Splice DB: Database of canonical and non-canonical mammalian splice sites. Nucleic Acids Res 2001, 29: 255–259

25 Wu Q, Krainer AR. AT-AC pre-mRNA splicing mechanisms and conservation of minor introns in voltage-gated ion channel genes. Mol Cell Biol 1999, 19: 3225–3236

26 Pruitt KD, Katz KS, Sicotte H, Maglott DR. Introducing RefSeq and LocusLink: Curated human genome resources at the NCBI. Trends Genet 2000, 16: 44–47

27 Pruitt KD, Tatusova T, Maglott DR. NCBI Reference Sequence (RefSeq): A curated non-redundant sequence database of genomes, transcripts and proteins. Nucleic Acids Res 2005, 33: D501–D504

28 Wheeler DL, Chappey C, Lash AE, Leipe DD, Madden TL, Schuler GD, Tatusova TA *et al*. Database resources of the National Center for Biotechnology Information. Nucleic Acids Res 2006, 34: D173–D180

Edited by
Jeong-Hyeon CHOI