Mapping RNA-seq reads to transcriptomes efficiently based on learning to hash method

Xueting Yu a,b, Xuejun Liu a,b,*

a College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, MIIT Key Laboratory of Pattern Analysis and Machine Intelligence, Nanjing 211106, China
b Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 210023, China

ABSTRACT

Accurate and efficient read-alignment is one of the fundamental challenges in RNA-seq analysis. Due to the increasingly large number of reads generated from the RNA-seq experiments, read-alignment is a time-consuming task. Many mappers adopted various strategies to look for potential alignment locations for reads in a tolerable time, and provide adequate information for downstream analysis. But in some transcript analysis tasks, such as transcriptome quantification, the mapping information about the transcripts and positions for reads is sufficient. Thus the original alignment problem can be simplified to a string searching problem since the reads can be mapped contiguously to the transcriptome. Some models for transcript analysis adopt more efficient strategies to solve this simplified problem, but the efficiency is still restricted by handling RNA-seq data in the original read space. We propose a method, bit-mapping, based on learning to hash algorithm for mapping reads to the transcriptome. It learns hash functions from the transcriptome and generates binary hash codes of the sequences, then maps reads to the transcriptome according to their hash codes. Bit-mapping accelerates mapping problems in RNA-seq analysis by reducing the dimension of the read. We evaluate the performance of bit-mapping based on simulated data and real data, and compare it with other popular and state-of-the-art methods, STAR, RapMap, Bowtie 2 and HISAT2. The comparative results of simulated and real data show that the accuracy of our method is competitive to the existing mappers in terms of mapping efficiency, especially for longer reads (> 100 bp).

1. Introduction

Accurate and efficient read-alignment is one of the fundamental challenges [1] in RNA-seq analysis due to the increasingly large number of reads generated from the experiments, especially single-cell RNA-seq experiments, which involve more and more individual cells and result in extraordinarily numerous reads. Many mappers adopted various strategies to look for the potential alignment locations for a read in a tolerable time, which is the most time-consuming step in the whole procedure, such as bowtie 2 [2], BWA [3], STAR [4], HISAT2 [5], etc. Some methods use variants of the FM-index, while others use kmer-based indices to speed up the process of alignment.

These mappers provide adequate information for the downstream analysis, but in some transcripts analysis tasks, many information is redundant. For example, transcriptome quantification and cluster de novo assembled contigs only need the transcripts, strands and positions from which the read may originate. Since it does not need to consider the splice junction, reads can be mapped contiguously to the transcriptome. Thus, the original alignment problem can be simplified to a string searching problem that finds where the query sequences appear in the reference sequences under the given error tolerance.

Some transcript analysis models adopted more efficient strategies to solve this simplified problem. For example, Kallisto [6], an isoform expression quantification model, introduced pseudoalignment to provide a sufficient statistic for quantification by constructing the De Brujin graph from k-mers presented in the transcriptome and generating an equivalence class for the read. RapMap [7] used in an isoform quantification model, STAR, RapMap, Bowtie 2 and HISAT 2. The comparative results of simulated and real data show that the accuracy of our method is competitive to the existing mappers in terms of mapping efficiency, especially for longer reads (> 100 bp).
functions, and intersects reads with indexed k-mers to estimate transcript abundances. These models are alignment-free methods, which means all these mapping strategies are not base-to-base alignments, but perform considerably well in transcript analysis tasks. However, they map reads to the transcriptome in the use of the raw read data, and the efficiency is still restricted by handling raw RNA-seq data in the original read space.

We find this simplified read mapping problem is similar to the nearest neighbor search (NNS) problem in machine learning area, which is aimed at finding the nearest neighbor to a query item under a certain distance measure. For the read mapping, the NNS problem needs to search the nearest neighbor of a query read from a large number of reference sequences database. A possible idea to accelerate the search is transforming a query to a low-dimensional representation, or equivalently a short hash code.

There are two main categories of hash algorithms: local sensitive hashing [10] and learning to hash [11]. Local sensitive hashing is a data-independent hash approach, and is a prevalent solution of genome assembly. It has been widely used in many assemblers, such as Mhap [12] and Minimap [13]. Learning to hash learns hash functions from a dataset to generate hash codes which preserve the similarity of data topology in the original read space to the hash space approximately. It is therefore possible to utilize learning to hash to transform reads into hash codes in the read mapping problem. There are several categories in learning to hash methods, such as unsupervised hashing [14,15], supervised hashing [16,17], deep hashing [18], online hashing [19] and so on. Since it is a data-dependent approach, learning to hash can preserve more features of data, and has been attracting a large amount of research interest in computer vision and machine learning, such as large scale object retrieval [20], image classification [21] and detection [22]. However, current mappers have not facilitated these methods to accelerate the read mapping process. In the following part, we will first introduce the details of bit-mapping method, and use the reduced forms of reads to detect their mapping positions. The mapping speed is expected to be further improved without the loss of accuracy.

In this paper, we propose a method, bit-mapping, based on learning to hash algorithm to efficiently map reads to the transcriptome in reduced hash code forms. In this method, we choose spherical hashing [23] to solve the hashing process, since it is a lightweight and unsupervised learning met-hod, and performs well in similar image searching problems. Bit-mapping learns the hash functions from sequences in the transcriptome, and map reads to the transcriptome by measuring the distance of their binary hash codes. It computes the hash codes of reads in binary forms rather than integer forms and thus performs efficient mapping due to rapid bit operations. In the following part, we will first introduce the details of bit-mapping algorithm, then demonstrate the usefulness of the method on simulated and real datasets in terms of mapping accuracy and efficiency.

2. Method

In this section, we will first introduce the basic idea of learning to hash, then introduce the definition of spherical hash function and the learning of hash functions, and finally illustrate the procedure of mapping in details.

Learning to hash

Learning to hash aims to learn hash functions from a dataset so that the nearest neighbor search result in the hash coding space preserves the similar topological structure of the original space, and the search cost as well as the space cost keep in a tolerant level.

The hash function is defined as: \( y = h(x) \), where \( h(\cdot) \) is a hash function, \( y \) is the hash value, and can be an integer or a binary value. In the nearest neighbor search, researchers usually use several hash functions together to generate a compound hash code: \( y = h_0(x), h_1(x), \ldots, h_m(x) \). That is, for a query item \( x \), it uses \( m \) hash functions to compute an \( m \)-bit hash code, and each hash function \( h_i(x) \) generates one bit of the hash code. There are many choices for the hash function, such as linear projection, kernels, spherical hash function, neural networks and so on. The form of hash function influences the searching accuracy and the computation speed of hash codes.

The searching accuracy is also affected by the way of learning hash functions. The main methodology of learning is similarity preserving, i.e., minimizing the gap between the similarities computed in the original space and those in the hash coding space by various forms. There are several categories of similarity preserving, such as pairwise similarity preserving, multiwise similarity preserving, implicit similarity preserving and quantization. The readers can refer to [11] for more information.

Spherical hash function

Considering the trade-off between efficiency and accuracy, we adopt the spherical hash function to generate hash codes. Spherical hash function uses a hypersphere to define a hash function, \( h_i(x) \), and maps sequences into binary values. As defined in the following, each hash function is determined by a pivot \( p_i \) and a distance threshold \( t_i \):

\[
h_i(x) = \begin{cases} 
1 & \text{if } d(p_i, x) < t_i \\
0 & \text{if } d(p_i, x) > t_i
\end{cases}
\]

where \( x \) is the sequence with fixed length in the original space, \( h_i(\cdot) \) represents the \( i \)th hash function that generates a binary value as the \( i \)th bit of a hash code, \( t_i \) represents the radius of a hypersphere, \( p_i \) represents the pivot of a hypersphere, and \( d(\cdot, \cdot) \) is the distance measure between two sequences. In the original spherical hashing algorithm, the value of each bit is determined by the distance between the sequence and the pivot of each hash function.

Some learning to hash algorithms usually define the similarity as a function of the distance between any pair of items, such as Gaussian function: \( s_j = \exp(d_j^2/2\sigma^2) \). When handling large amount of data, computing such similarity function frequently is time-consuming comparing to computing a simpler distance measure. Thus, we choose a simple distance measure, Euclidean distance, to measure the distance between the sequence and the pivot of a hash function. As a matter of fact, the ideal distance measure in the original read space is edit distance, since it considers both substitution and indel error. However, the time complexity of edit distance for sequences with length \( n \) is \( O(n^2) \) which is too high to make the mapper efficient. Different from edit distance, Euclidean distance just considers substitution error of sequences, but in linear time complexity. As discussed in [24], there are several sequencing platforms up to now, e.g. Illumina, Ion Torrent and so on. For Illumina platform, the primary error is substitution (\( \geq 0.1\% \) error rates), whereas the primary error of Ion Torrent is indel (\( \sim 1\% \) error rates). Since Illumina has the broadest utility comparing to other platforms, we only consider substitution error in our study to obtain efficient distance measurement.

Essentially, hyperspheres divide the original space into several subspaces. Each subspace contains some similar sequences, and owns a unique hash code that represents certain features of the sequences in this subspace. For example, three hash functions can partition the original space into eight subspaces (as shown in Fig. 1). Each subspace has a unique hash code, and the sequences mapped into the same subspace are most similar in some extent. As described above, the dimension of the sequence \( x \) is the same as the dimension of the pivot \( p_i \) in a spherical hash function \( h_i(x) \). When computing \( m \)-bit hash code for a read with \( n \) bases length, the Euclidean distance between the sequence and the pivot of each hash function has to be measured \( m \) times, and the time complexity of each measurement is \( O(n) \), so the whole time complexity of computing a hash code for a read is \( O(mn) \).
This will slow down the computation speed of hash code when the length of the read $n$ is long. The existing learning to hash methods hardly consider computation speed of the hash code, since their main purpose is to generate more accurate hash codes. While for the read mapping problem, the computation speed is an important issue.

For the spherical hash function, we consider to speed up the process by reducing the dimension of the sequence $x$. So we apply spherical hash functions on the $k$-mers of the read. In this case, $x$ in hash functions is substituted k-mers for the whole sequence, which means that the $i$th hash function generates the $i$th bit by measuring the Euclidean distance between its pivots and the $i$th $k$-mer of the read (see Fig. 2). The time complexity of computing a hash code is about $O(km)$, where $k$ is the length of the $k$-mer that is much shorter than the length of the whole sequence, and $m$ is the length of the hash code. Therefore, the time complexity of the computation of hash codes is decreased in this way.

**Hash function learning**

Spherical hashing learns hash functions by implicit similarity preserving, which partitions the space without explicitly evaluating the relation between the distances in the original space and the coding space.

It learns the parameters, $p_i$ and $i_i$, of the $i$th hash function from the $i$th $k$-mer of the sequences in the transcriptome (see Fig. 3). The learning process achieves two properties: the balanced partition of data for each hash function and the independence between hash functions. Thus, suppose there is a dataset containing $N$ sequences, $S = \{ s_1, s_2, \ldots, s_N \}$, spherical hashing computes two variables, $o_{i,j}$ and $i_{i,j}$ given $1 \leq i,j \leq N$:

$$o_{i,j} = | \{ s_k | h_i(s_k) = +1, 1 \leq k \leq N \} |,$$

$$i_{i,j} = | \{ s_k | h_j(s_k) = -1, 1 \leq k \leq N \} |,$$

where $| \cdot |$ is the cardinality of the given set. $o_{i,j}$ records the number of sequences that have 1 bit for the $i$th hash function. $o_{i,j} \approx \frac{N}{2}$ means that the probability that the $i$th function generates 1 bit or 0 bit is equal, so the $i$th hash function satisfies the balanced partition of data. Threshold $i_j$ is adjusted to satisfy $o_{i,j} \approx \frac{N}{2}$. While $o_{i,j}$ records the number of sequences that have 1 bit for both the $i$th and the $j$th hash functions. $o_{i,j} \approx \frac{N}{4}$ means that the $i$th hash function and the $j$th hash function are independent to each other, since they satisfy the following formula,

$$P[h_i(s_k) = 1, h_j(s_k) = 1] = \frac{1}{4}.$$

For an iteration, $p_i$ is adjusted by the following formula:

$$f_{i,j} = \frac{1}{m} \sum_{j=1}^{m} f_{i,j},$$

$$f_{i,j} = \frac{1}{2} \left( \frac{1}{m} \sum_{j=1}^{m} f_{i,j} \right) - \frac{1}{2} \left( \frac{1}{m} \sum_{j=1}^{m} f_{i,j} \right),$$

$$p_i \leftarrow p_i + f_i.$$

where $f_{i,j}$ is the repulsive or attractive force from $p_i$ to $p_j$, $f_i$ is the average of forces from all the pivots, and $m$ is the number of hash functions.

The procedure of learning hash functions is illustrated in Algorithm 1. For detailed explanation, please refer to [23]. In practice, we randomly sampled 10,000 sequences with given length from the transcripts in the transcriptome to learn hash functions, and the maximum learning iteration is 50.

When mapping reads to the transcriptome, we compute the hash codes of the reads and the sequences in the transcriptome, and measure their similarity by calculating the distance between their hash codes. Here, we use Hamming distance, namely counting the number of different bits between binary codes.

Since the spherical hash function just maps a sequence to a binary value, it is not a bijection and may map multiple sequences to a single hash code. Thus, this algorithm may find the totally wrong alignments occasionally. Although we cannot ensure original reads and sequences in the transcriptome with similar hash codes are also exactly similar to each other when applying spherical hashing to read mapping, it can be promised that the most similar sequences of an original read are contained in its hash code subspace. Hence, for paired-end reads we can find the transcripts from which these similar sequences of two mates are generated respectively, then get the common transcripts of them. These common transcripts are the potential transcripts where the paired-end reads come from. Thus, our method is most suitable for paired-end reads.
Algorithm 1 Learning process of hash functions

Require: sample sequences $S = \{s_1, s_2, \ldots, s_N\}$, error tolerances $\epsilon_1, \epsilon_2$, and the number of hash functions $m$

Ensure: pivots $p_1, p_2, \ldots, p_m$ and distance thresholds $t_1, \ldots, t_m$ for $m$ hash functions

Initialize $p_1, p_2, \ldots, p_m$ randomly
Determine $t_1, t_2, \ldots, t_m$ to satisfy $o = \frac{N}{4}$
Compute $o_j$ for each pair of hash functions

while $\text{avg}(o_j - \frac{N}{4}) > N$ or $\text{std} - \text{dev}(o_j) > N$ do
    for $i = 1$ to $m - 1$ do
        for $j = i + 1$ to $m$ do
            $f_{i,j} = \frac{1}{2} \left( o_j - \frac{N}{4} \right) (p_i - p_j)$
            $f_{i,j} = -f_{i,j}$
        end for
    end for
    for $i = 1$ to $m$ do
        $f_i = \frac{1}{m} \sum_{j=1}^{m} f_{i,j}$
        $p_i = p_i + f_i$
    end for
    Determine $t_1, t_2, \ldots, t_N$ to satisfy $o = \frac{N}{4}$
    Compute $o_j$ for each pair of hash functions
end while

i. Merging all the transcripts in the reference transcriptome to a long sequence with a special separator character, then constructing the suffix array of the long sequence.

ii. Using hash part of the sequences in the suffix array to compute their hash codes.

iii. Determining the measuring region of the suffix array for reads according to the region-searching part of the reads.

iv. Detecting the most similar sequences for each read in its measuring region.

The first two steps compose the indexing phase, and the last two steps compose the mapping phase.

When mapping reads to the transcriptome, since it is time-consuming to measure Hamming distances between a read and all the sequences in the suffix array, we have to reduce the region of measurement to accelerate the searching speed. There are various strategies to deal with this problem. For instance, Bowtie 2 constructs the FM index of suffix array to accelerate the searching speed, and RapMap computes the MMP and NIP of reads iteratively to determine the searching region. However, these strategies are not suitable for the hash code operation in bit-mapping. In the first step, bit-mapping directly divides suffix array into several separate measuring regions according to the first $k$ bases of sequences in the suffix array. Each measuring region contains the sequences that have the same first $k$ bases (see Fig. 5).

In the third step, we determine the measuring region for reads according to their region-searching part. The measuring region in the suffix array contains the sequences whose first $k$ bases are the same as the region-searching part of the read. By identifying the measuring region, the calculation of Hamming distance is performed between the read and the sequences in the measuring region rather than in the whole suffix array. The calculation is therefore sped up substantially.

In the final step, we first compute the hash code of the read using spherical hashing, then measure the Hamming distances between the hash code of the read and the hash codes of the sequences in its measuring region. The sequences with the minimum Hamming distance are most similar to the read, and the positions of these sequences in transcripts are the most possible positions where the read generates. At this step, the efficient binary operations of Hamming distance calculation also accelerate the algorithm.

Unlike traditional alignment methods, such as Bowtie 2 and HISAT 2, the mapping procedure is much easier since bit-mapping needs not consider the splice junctions of reads when mapping reads to the transcriptome. When comparing to alignment-free methods, such as RapMap, our method avoids computing the MMP, NIP and LCP of reads. This also leads to efficient calculation. It should be noticed that bit-mapping is designed for Illumina paired-end reads in consequence of primary substitution error. For paired-end reads, bit-mapping finds the candidate transcripts sets of two mates, $T_i = \{t_1, t_2, \ldots, t_m\}$ and $T_j = \{t_1, t_2, \ldots, t_n\}$. The final potential transcripts set of the paired-end reads is $T_{i,j} = \{t_k \mid t_k \in T_i \cap T_j\}$, which is the common candidate transcripts of two mates.

Code

We have implemented bit-mapping in a free C++ tool, which is publicly available from https://github.com/PUGEA/Bit-mapping. For details on implementation, in indexing phase, we first merge all the transcripts in the reference transcriptome to a string with the specified separator terminator and make use of a C++ library, libdivsufsort, to construct suffix array. Then bit-mapping learns parameters of hash functions and compute the hash code of sequences appearing in the transcriptome. For efficiently determining the measuring region of the reads, we use a minimal perfect hashing algorithm, BBHash [26], to hash the region-searching part of the read and the first $k$ bases of sequences in suffix array.

![Fig. 4. The segmentation of the read. In this example, the first 5 bases are skipped, and the next 7 bases form the region-searching part. The hash part containing 15 bases is used to compute hash code, and the rest are the ignored bases.](image-url)
Fig. 5. Mapping procedure. The mapping contains four steps, the first two steps form the indexing phase, and the last two form the mapping phase. In the suffix array, the sequences in the red box form the measuring region of the read, and the subsequences in the top blue box are the same as the region-searching part of the read. In the mapping process, bit-mapping just compares the Hamming distance of the hash codes of the read and the sequence, and looks for the nearest hash code in the measuring region. The sequences with nearest hash code are most similar to the read.

Table 1
Information of simulated and real data. All datasets are the paired-end reads, since bit-mapping is more suitable for paired-end reads.

<table>
<thead>
<tr>
<th>GEO accession</th>
<th>Run</th>
<th>Type</th>
<th>Size</th>
<th>Length of reads</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulated data</td>
<td>None</td>
<td>None</td>
<td>Paired-end</td>
<td>2.08 million</td>
<td>Flux simulator</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>None</td>
<td>Paired-end</td>
<td>2.5 million</td>
<td>Flux simulator</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>None</td>
<td>Paired-end</td>
<td>2.8 million</td>
<td>ART simulator</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>None</td>
<td>Paired-end</td>
<td>4 million</td>
<td>ART simulator</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>None</td>
<td>Paired-end</td>
<td>2 million</td>
<td>ART simulator</td>
</tr>
<tr>
<td>Real data</td>
<td>GSM 2533673</td>
<td>SRR5337025</td>
<td>Paired-end</td>
<td>4.46 million</td>
<td>Illumina</td>
</tr>
<tr>
<td></td>
<td>GSM 981249</td>
<td>SRR534302</td>
<td>Paired-end</td>
<td>73.68 million</td>
<td>Illumina</td>
</tr>
<tr>
<td></td>
<td>GSM 1395234</td>
<td>SRR1293902</td>
<td>Paired-end</td>
<td>25.93 million</td>
<td>Illumina</td>
</tr>
<tr>
<td></td>
<td>GSM 1395233</td>
<td>SRR1293901</td>
<td>Paired-end</td>
<td>9.52 million</td>
<td>Illumina</td>
</tr>
</tbody>
</table>

3. Results

Datasets

We evaluate the performance of bit-mapping on both simulated and real RNA-seq datasets. Three simulated datasets (50 bp, 76 bp, 100 bp) are generated for the human transcriptome from the latest annotation of ENSEMBL using Flux Simulator [27]. Two simulated datasets with longer reads (150 bp and 250 bp) are generated by ART simulator [28]. We use four real data with various lengths of reads collected from four GEO accessions. The details on the datasets in use are shown in Table 1.

We filtered adaptors and low-quality reads using Trimmomatic [29], which is a flexible preprocessing for Illumina sequencing data. The size of SRR5337025 is reduced to ~ 4.32 million, and the size of SRR1293901 is reduced to ~ 2.9 million after trimming. For the last two datasets, we randomly take 4 million reads from each of them for efficient verification of our method since the mapping time linearly increases with the size of the dataset. The reference used in experiment is the GRCh38 assembly of the human transcriptome.

The performance comparison between various methods is conducted in terms of mapping speed and accuracy. All processes are performed on a machine with Intel Core i5-7500 and 64GB main memory.

Parameter settings of bit-mapping

There are several important parameters in bit-mapping, which are divided into two categories in light of their functions. The first category includes the parameters for training spherical hash functions, bit-mapping has set fixed values for them to ensure a good performance, so users do not need to set these parameters manually. Another category includes the parameters for mapping, e.g., hash code length, skipping size, mapping length (to ignore some bases in the end of reads), the size of region-searching bases and tolerance (to determine the threshold of similarity in hash coding space).

Among the parameters for mapping mentioned above, the most important parameter is the length of hash code. Setting longer hash code will generate more reliable results, but slow down the mapping speed. Thus, we test the effect of the length of the hash code on the 250 bp simulated dataset with one million reads. The results are shown in...
Parameter settings for simulated data and real data.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Hash code length</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRR5337025</td>
<td>27 bits</td>
</tr>
<tr>
<td>SRR534302</td>
<td>57 bits</td>
</tr>
<tr>
<td>SRR1293901</td>
<td>97 bits</td>
</tr>
<tr>
<td>SRR1293902</td>
<td>57 bits</td>
</tr>
<tr>
<td>50 bp</td>
<td>27 bits</td>
</tr>
<tr>
<td>76 bp</td>
<td>57 bits</td>
</tr>
<tr>
<td>100 bp</td>
<td>57 bits</td>
</tr>
<tr>
<td>150 bp</td>
<td>97 bits</td>
</tr>
<tr>
<td>250 bp</td>
<td>97 bits</td>
</tr>
</tbody>
</table>

Performance on the simulated RNA-seq data

We map the reads to the transcriptome directly and compare bit-mapping with Bowtie 2, HISAT 2, STAR and Rap-Map. Specially, we set Bowtie 2 to report at most 200 multi-mapping locations rather than the optimal locations as producing the optimal location leads to too low accuracy. We therefore refer the settings of Bowtie 2 and STAR in [7] to obtain a higher level of accuracy.

For benchmarking the accuracy of each method, we define that the read is correctly mapped if the mapping result of the read recorded by the method contains the true transcript reported by Flux Simulator and ART simulator, otherwise the read is incorrectly mapped. And we define the read is unmapped if the read cannot map to any transcripts. We calculate the correctly mapped rate, the incorrectly mapped rate, and the unmapped rate of all the methods to compare their accuracy.

Fig. 7 shows the average accuracy of each method on the five simulated data. Bowtie 2 outperforms the other four methods on the three short read datasets (50 bp, 76 bp and 100 bp), followed by bit-mapping and RapMap. It achieves the highest correctly mapped rate and the lowest incorrectly mapped rate. Bit-mapping performs similarly to RapMap and outperforms HISAT 2 and STAR. While for longer read datasets (150 bp and 250 bp), bit-mapping surpasses the others, and HISAT 2 achieves the worst performance. Since we set Bowtie 2 to report at most 200 multi-mapping locations, the results of Bowtie 2 contains more suboptimal multi-mapping locations for each read, while other methods only report the optimal multi-mapping locations. It is thus not surprising that Bowtie 2 obtains the highest correctly mapped ratio. The unmapped ratio of STAR is large on both the short and the longer reads probably due to the inapplicability of transcriptome mapping. While the incorrectly mapped ratio of STAR is the lowest among the five methods on both the short and the longer reads. HISAT 2 performs much poorly on the longer reads comparing to the accuracy on the short reads, and the incorrectly mapped ratio is largest. It must be noticed that since bit-mapping maps reads to the transcriptome by measuring Hamming distance of their hash codes and the calculation of Hamming distance depends on the learned hash functions, the mapping results of reads vary slightly when different training datasets sampled from the transcriptome are used. However, the mapping accuracy does not vary significantly.

To test the speed of each method, we ran all the methods in 4 threads. Since the time of loading the suffix array and other useful data is determined by the computer environment and the manner of implementation, we mainly focus on the searching time of mapping process. We test the speed on five simulated data. When we make Bowtie 2 report at most 200 multi-mapping locations of reads, the runtime is larger than 4000 s, which is much slower than the others. Also, the speed of Bowtie 2 is still slower than the others even if reporting the optimal mapping location of reads. STAR is a little faster than Bowtie 2, but is still much slower than the other three methods. Fig. 8 shows the average mapping speed of each method on short reads (50 bp, 76 bp, 100 bp) and longer reads (150 bp, 250 bp), and bit-mapping takes the lowest time comparing to others. The gap between bit-mapping and RapMap on the longer reads is greater than that on short reads, which means bit-mapping is more efficient on longer reads comparing to the others.
Fig. 8. Mapping speed on the simulated data. The short reads include 50 bp, 76 bp and 100 bp reads, and the longer reads include 150 bp and 250 bp reads. The speeds of bit-mapping, RapMap and HISAT 2 are much faster than STAR and Bowtie 2. And bit-mapping achieves the highest speed among the five methods.

Fig. 9. Average mapped ratio on the real data. The mapped ratio of bit-mapping surpasses others, while the mapped ratio of STAR is less than half.

Performance on real RNA-seq data

For real data, since the true transcripts from which the reads are generated are unknown, we first evaluate the mapping ratio of each method on the five real datasets. The mapped ratio means the percentage of reads that can be mapped to the transcriptome. The average results are reported in Fig. 9. Bit-mapping yields the highest mapped ratio on average at 74.45%, followed by RapMap, and the mapped ratios of the others are a little lower than bit-mapping. Certainly, achieving higher mapped ratio does not mean the better performance, since the incorrectly mapped reads may be contained. We next investigate the agreement among the mapping results of the five methods.

We say two methods agree on the mapping locations of a read if they map this read to the same set of transcripts. For example, the transcripts of read \( r_i \) mapped by method A is \( \{t_1, t_2\} \), while method B maps \( r_i \) to transcripts \( \{t_2, t_3\} \), then A and B do not agree on \( r_i \). The average agreement of the mapping results reported by the four methods on the four datasets is shown in Fig. 10. Bar plot in Fig. 10 shows the size of the intersections among the results of the methods, and black dots below the bar plot represent the considered methods. The black dots linked to each other mean that the corresponding bar is the agreement of methods represented by these dots. These results show that most mapping results of bit-mapping are consistent with other methods, which indirectly indicates the mapping accuracy of bit-mapping.

From the comparison of mapping speed (see Fig. 11), all the methods still map reads in 4 threads. We make Bowtie 2 report the optimal mapping location rather than at most 200 multi-mapping locations for efficiency reason. STAR, Bowtie 2 and HISAT 2 run slowly as predicted. Bit-mapping runs slower than RapMap on SRR5337025 dataset, and performs similarly to RapMap on SRR1293902 and SRR534302. Bit-mapping is especially efficient for longer reads (SRR129-3901), which is also consistent with the findings on the simulated data.

Performance on isoform quantification

To further estimate the mapping accuracy of bit-mapping on real data, we apply the mapping methods to the benchmark SEQC dataset [30] for the downstream isoform expression quantification. SEQC (GSE47792) has independently quantified 20801 transcripts on RT-PCR assay and provides RT-PCR results. The RT-PCR measurements in this dataset are regarded as the ground truth. The Pearson correlation between RT-PCR values and TPM values reflects the accuracy of isoform expression quantification. Since the mapping results of all methods are processed by the same expression quantification method, the read mapping accuracy determines the quantification accuracy. Thus, we evaluate the mapping accuracy of each method on SEQC data by evaluating the Pearson correlation between RT-PCR values and TPM values.

The length of RNA-seq reads in SEQC is 100 bp. We test the five methods on the 8 lanes of sample A (UHRR) and sample B (HBRR). The reference transcriptome used in expression quantification is Ensembl GRCh37, release 75. TPM (transcripts per million reads) values of those transcripts validated by RT-PCR assay are calculated by Salmon [31], and we compute the mean TPM values of the 8 lanes for further analysis. We filter the common expressed transcripts between the transcripts detected by RT-PCR assay and the transcripts reported by Salmon for both samples. The number of filtered transcripts is 4,253 for UHRR, and 4,074 for HBRR. We use these transcripts to evaluate the agreement of the expression measurements between RNA-seq profiles and the RT-PCR assay.

Fig. 12 shows the scatter plots of the RNA-seq measurements versus the RT-PCR results. The Pearson correlation between RT-PCR and RNA-seq data is computed to indicate the agreement between them. The higher agreement between RT-PCR and TPM calculated by Salmon indicates that the mapping result is more accurate. Pearson correlation coefficients of sample UHRR and HBRR are shown on the top left corner in Fig. 12. According to these Figures, RapMap \((R^2 = 0.8035 \text{ for UHRR, and } R^2 = 0.7976 \text{ for HBRR})\) outperforms other methods, followed by Bit-mapping \((R^2 = 0.7991 \text{ for UHRR, and } R^2 = 0.7915 \text{ for HBRR})\), STAR \((R^2 = 0.7959 \text{ for UHRR, and } R^2 = 0.7851 \text{ for HBRR})\) and HISAT 2 \((R^2 = 0.7898 \text{ for UHRR, and } R^2 = 0.7797 \text{ for HBRR})\). The lowest is Bowtie 2 \((R^2 = 0.7867 \text{ for UHRR, and } R^2 = 0.7701 \text{ for HBRR})\). The low correlation of Bowtie 2 is possibly caused by the influence of reporting the unique optimal mapping location, and the low correlation of HISAT 2 may result from the high incorrectly mapped ratio according to the simulated data analysis. From the comparison on the downstream analysis, the accuracy of bit-mapping on real data is proved to be considerably competitive to others.

4. Discussion

In summary, we have estimated the performance of our new method on both simulated reads and real reads. The experiment results have shown that bit-mapping is competitive in terms of mapping accuracy and efficiency.

While evaluating the mapping speed on the real reads, we find the mapping speed of bit-mapping is a little slower than the speed on the simulated reads. This is likely due to the different distributions of reads between the simulated and real reads. That is, the measuring regions of real reads may contain more sequences, which slows down the searching. For example, we calculate the percentage of the reads in different measuring regions on SRR5337025 (50 bp) dataset and the simulated dataset with 50 bp reads (Fig. 13). The majority of reads are distributed in the range from 100 to 1000 on both real reads and simulated reads, while the percentage of real reads in other
region is slightly different from that of simulated data. This may lead
to different mapping speed of real data. Especially, in the measuring
region with the largest size of 75,753 and 82,652, the percentage
of reads rises dramatically. Since the prefix of these two measuring
regions are TTTTTTTTTT and AAAAAAAATA, respectively, the higher
proportion of low complexity reads [32,33] in this region may cause the
slowdown of mapping for the real data. Users can filter low complexity
reads before mapping to improve the mapping efficiency by FASTQ
preprocessors, such as fastp [34].

Another factor that affects the mapping speed is the twice searching
in the mapping process. As summarized in [24], each mate of paired-
end reads points towards each other on opposite strands. Since the
orientation of the paired-end reads is unknown in advance, bit-mapping
presumes that the orientation of the first read is the forward strand
when mapping. If the minimum Hamming distance of the read is larger
than the given tolerance, bit-mapping will reverse the orientation of
the read and search again. Thus most of reads will search twice if
the true orientation is opposite to the presumption. For SRR1293902
and SRR534302, the mapping speed is raised by around 15% when
presuming that the orientation of the first read is reverse strand. But
for SRR5337025, the mapping speed is invariant under the different
presumption since the number of the first read with forward strand is
the same as that with reverse strand.

Finding the measuring region of reads correctly is an important step
for bit-mapping. We used a perfect hash algorithm library [26] to index
the region-searching part of reads and the prefix of each measuring
region. It is an injective hash algorithm which means each measuring
region owns the unique code without conflict. However, this feature
will cause the problem that once there are some mismatches in the
region-searching part of a read, the read will inevitably match to a
wrong measuring region. On the contrary, if using a non-injective hash
algorithm, a read will match more than one measuring region. This
easily slows down the searching speed. It is thus a trade-off between
efficiency and accuracy, so the best solution at present is to choose the
perfect hash algorithm with a suitable length of the region-searching
part to ensure the minimum mismatches.

Another thing which should be pointed out is that when mapping
longer reads to the transcriptome, such as 150 bp and 250 bp simulated
reads, the average accuracy of bit-mapping surpasses others apparently.
This is because for the longer reads which may contain more errors, the
methods based on the original reads easily limit fault tolerance of bases,
while bit-mapping increases the fault tolerance of bases by mapping reads in a lower dimension space. This helps to obtain robustness of the method to the noise.

5. Conclusion

In this contribution, we present bit-mapping, a fast mapping method for RNA-seq read alignment. It is a novel method to map reads to the transcriptome by measuring the binary hash codes generated by spherical hashing. Bit-mapping reduces the dimension of reads to speed up the mapping time, and performs well on both simulated data and real data, especially on longer RNA-seq reads. It shows competitive performance compared with other alternatives, such as STAR, Bowtie 2, RapMap and HISAT 2, in terms of the mapping accuracy and efficiency. Our work provides an experimental tool for RNA-seq read mapping, especially for transcript expression analysis area where less alignment information is needed. Besides, we are next working to improve bit-mapping to be a more general method for genome alignments in the future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Key R&D Program of China (Grant Nos. 2018YFC2001600, 2018YFC2001602).

References


Heng Li, Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences, Bioinformatics 32 (14) (2016) 2103–2110.


Cong Leng, Jiaxiang Wu, Jian Cheng, Xia Bai, Hanqing Lu, Online sketching hashing, 2015, pp. 2503–2511.


