Sequence analysis

PerM: efficient mapping of short sequencing reads with periodic full sensitive spaced seeds

Yangho Chen, Tade Souaiaia and Ting Chen*

Program in Computational Biology and Bioinformatics, University of Southern California, 1050 Childs Way, Los Angeles, CA 90089-2910, USA

Received on May 14, 2009; revised on July 18, 2009; accepted on August 7, 2009

Advance Access publication August 12, 2009
Associate Editor: Joaquin Dopazo

ABSTRACT

Motivation: The explosion of next-generation sequencing data has spawned the design of new algorithms and software tools to provide efficient mapping for different read lengths and sequencing technologies. In particular, ABI’s sequencer (SOLiD system) poses a big computational challenge with its capacity to produce very large amounts of data, and its unique strategy of encoding sequence data into color signals.

Results: We present the mapping software, named PerM (Periodic Seed Mapping) that uses periodic spaced seeds to significantly improve mapping efficiency for large reference genomes when compared with state-of-the-art programs. The data structure in PerM requires only 4.5 bytes per base to index the human genome, allowing entire genomes to be loaded to memory, while multiple processors simultaneously map reads to the reference. Weight maximized periodic seeds offer full sensitivity for up to three mismatches and high sensitivity for four and five mismatches while minimizing the number random hits per query, significantly speeding up the running time. Such sensitivity makes PerM a valuable mapping tool for SOLiD and Solexa reads.

Availability: http://code.google.com/p/perm/

Contact: tingchen@usc.edu

Supplementary information: Supplementary data are available at Bioinformatics online.

1 INTRODUCTION

Next-generation sequencing technology has created the need for highly efficient methods to align short DNA reads (Shendure and Ji, 2008). Current technologies (Ronaghi et al., 1998), e.g. the Illumina and SOLiD platforms, are capable of generating hundreds of millions of short reads in a single run (Applied Biosystems, 2008a; Bennett, 2004). These breakthroughs have led to many important biological applications, including the identification of transcription factor binding sites (Chip-Seq; Mardis, 2007), estimation of RNA expression levels (RNA-Seq; Marioni et al., 2008) and SNP calling (Quinlan et al., 2008).

Applied Biosystems ligation-mediated SOLiD sequencer has the unique property of collecting color signals for the transitions between nucleotides. Each single nucleotide polymorphism changes two transitions (i.e. consecutive colors), providing a large advantage in the detection of SNPs because only a fraction of color changes represent possible SNPs (Applied Biosystems, 2008a) and single color discrepancies can be regarded as sequencing errors. While this unique encoding results in base accuracy as high as 99.94% (Applied Biosystems, 2008a), it also requires algorithms capable of finding alignments with less similarity than those designed for sequencers such as Solexa which output signals directly from nucleotides.

Most read mapping programs are designed to be full sensitive to ‘k’ mismatches, meaning all alignments within k mismatches will be reported. Original alignment tools such as BLAST (Altschul et al., 1990) and BLAT (Kent, 2002), are capable of finding highly sensitive alignments for long reads, but do not provide full sensitivity to specific numbers of mismatches and are incapable of efficiently mapping the amount of reads currently produced by short-read sequencing machines. Many mapping programs have been designed to handle large amounts of short reads, including ELAND (Anthony, J. Cox, 2006, unpublished data), MAQ (Li et al., 2008), ZOOM (Lin et al., 2008), RMAP (Smith et al., 2008) and SeqMap (Jiang and Wong, 2008), which preprocess and index read sets and then scan the reference for potential matches. Programs including SOAP (Li et al., 2008), Pass (Campagna et al., 2009), MOM (Eaves and Gao, 2009), ProbMatch (Kim et al., 2009), SXOligoSearch (Malaysian Genomics Resource Center, 2009), Mosaic (The MarthLab, 2009), BWA (Li and Durbin, 2009) and Bowtie (Langmead et al., 2009) preprocess and index the reference and then search for potential matches among the reads.

In many SOLiD datasets (Ondov et al., 2008, Section 3.3), more than half of the sequenced reads will not align to the reference with fewer than three mismatches. For this reason, programs designed specifically for SOLiD data, including the Corona Lite pipeline (Applied Biosystems, 2008c) and SOCS (Ondov et al., 2008), provide full sensitivity to alignments including three or more mismatches. Such sensitivity usually comes at a cost; the shorter subsequences used to provide full sensitivity to high numbers of mismatches also slow mapping such that genome-scale mapping may not be possible.

PerM (Periodic Seed Mapping) ameliorates these difficulties through the design of periodic spaced seeds to maximize efficiency for many distance measures, including those specific to the location of polymorphism in color space. For example, rather than accepting the slower performance associated with sensitivity to multiple mismatches, we introduce a faster method which provides full sensitivity to one potential SNP locus (consecutive mismatches) and

*To whom correspondence should be addressed.
This method is displayed in Figure 1. The read is divided into four equal size fragments, resulting in the hashing of six substring pairs. These six pairs of substrings are encompassed into index tables required to be built during preprocessing. PerM’s single spaced seed with weight (number of ‘care’ positions) 11. Mathematical results (Buhler et al., 2003; Xu et al., 2006) have shown that spaced seeds are more sensitive than consecutive seeds (those without ‘don’t care’ positions) in finding local similarities between two strings. Kucherov et al. (2005) attempted to minimize the number of multiple spaced seeds necessary to achieve different levels of sensitivity. Lin et al. (2008) used multiple spaced seeds for short-read mapping and provided bounds for the number of lookups necessary to achieve full sensitivity for varying seed lengths and weights. They showed that for a 32 bp read, seeds with the weight of 16 require at least six lookups to obtain full sensitivity for two mismatches. They implemented this idea into a program, called ZOOM (Lin et al., 2008).

SHRIMP (Rumble et al., 2009) also uses spaced seeds to find hits, however, they find alignments with InDels as well as mismatches which requires significantly longer running time. PerM uses single periodic weight-maximized spaced seeds. An example, shown in Figure 2, is composed of four repeating patterns of (111*1**) whose length is seven and is full sensitive to two mismatches. For any alignment within two mismatches, at least one of the seven subsequences will match exactly. This seed is composed of repeating the pattern (111*1**).

In Section 2, we describe how PerM’s periodic seeds allow to increase mapping efficiency and sensitivity when compared with the conventional consecutive seeds used by many other programs.

2 METHODS

2.1 Seed notation

\[ C_k \]: the conventional seed family which divides reads into \( k+2 \) fragments (used in ELAND, MAQ and SOAP) to provide full sensitivity to \( k \) mismatches.

\[ F_k \]: the maximum-weight periodic spaced seed family which is full sensitive to \( k \) mismatches.

\[ S_{x,k} \]: the special weight maximized periodic spaced seed family for mapping SOLiD reads, full sensitive to \( x \) SNP candidates (consecutive mismatches) and \( k \) free mismatches.

2.2 Motivation for periodic design

For any full sensitive seed family, there exists two performance criteria, memory load and running time. Memory load is directly related to the number of index tables required to be built during preprocessing. PerM’s single
The periodic spaced seed, applied to a read and slid through positions 8–14 six times, covers all the 21 pair of positions exactly once:

\[
\begin{align*}
\text{Slide 0:} & \quad 1 \quad 1 \quad 1 \quad * \quad 1 \quad * \quad (11,13) \quad (11,14) \quad (13,14) \\
\text{Slide 1:} & \quad * \quad 1 \quad 1 \quad 1 \quad * \quad 1 \quad * \quad (8,12) \quad (8,14) \quad (12,14) \\
\text{Slide 2:} & \quad * \quad 1 \quad 1 \quad 1 \quad * \quad 1 \quad (8,9) \quad (8,13) \quad (9,13) \\
\text{Slide 3:} & \quad 1 \quad * \quad 1 \quad 1 \quad * \quad 1 \quad * \quad (9,10) \quad (9,14) \quad (10,14) \\
\text{Slide 4:} & \quad * \quad 1 \quad * \quad 1 \quad 1 \quad (8,10) \quad (8,11) \quad (10,11) \\
\text{Slide 5:} & \quad 1 \quad * \quad 1 \quad * \quad 1 \quad (9,11) \quad (9,12) \quad (11,12) \\
\text{Slide 6:} & \quad 1 \quad 1 \quad 1 \quad * \quad * \quad 1 \quad (10,12) \quad (10,13) \quad (12,13)
\end{align*}
\]

Positions 8 9 10 11 12 13 14 Covering 21 pairs of positions

periodic seeds store a single index for each locus, requiring only 4.5 bytes per base to index the entire genome. Running time is dependent linearly on the number of queries or slides required of a seed but can be influenced exponentially by the seed weight (number of ‘care’ positions). Specifically, periodic seeds that are composed of repeating patterns of length \( |P| \) will generate, for each read, \( |P| \) subsequences of weight \( w \) for query, where \( w \) is the seed weight. An approximation for the expected number of random hits per read is \( 2|G||P|/4^{|G|} \), where \( |G| \) is the length of the genome and the number 2 is necessary for the forward and reverse complement. For large-scale genome projects, if the seed weight is not sufficiently large, the number of random hits will grow by a factor of four each time the number 2 is necessary for the forward and reverse complement.

To achieve full sensitivity a periodic seed will slide \( |P| \) times and generate \( |P| \) subsequences on a read. As shown in Figure 2, this is accomplished by keeping open the last \( |P| \) positions at the end of the read. However, as periodic seeds maintain full sensitivity when generalized, these empty positions are not required. This is shown with a different example in Figure 4, each slide seeds can be extended to the end of the read so that all but the last are likely to have increased seed weight. In the example shown in Figure 4, seed extension results in weight increasing from the minimum of 14 to between 14 and 19 which correspond to an approximately five times faster mapping performance for the whole human genome.

To take the advantage of the extended seed weight without additional requirements to memory, PerM combines hashing with the binary search. Our special data structure consists of a hash table which stores pointers to the starting position of each bucket in an index table which stores the addresses of reads or the genome locations. In the example shown in Figure 4, we preprocess a genome using the maximum weight seed with \( W = 19 \) to generate one subsequence for every position in the genome. The prefixes of each subsequence (e.g. the first 14 bases) is used to hash the location into a bucket in the index table. Inside each bucket, multiple subsequences with the same prefix are stored lexicographically and stored in the index table. Note that within each bucket, we can search for exact matches of variable-length strings using the binary search. Thus, we applied the extended variable-weight seeds to each read to generate variable-length subsequences, and use the extended bases after 14 to find the exact matches using the binary search in the index table. This method results in seed weight which varies from 14 to 19 in the example shown in Figure 4.

Table 1. The periodic spaced seed, applied to a read and slid through positions 8–14 six times, covers all the 21 pair of positions exactly once

<table>
<thead>
<tr>
<th>Positions</th>
<th>8 9 10 11 12 13 14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slide 0</td>
<td>1 1 1 * 1 * (11,13) (11,14) (13,14)</td>
</tr>
<tr>
<td>Slide 1</td>
<td>* 1 1 1 * 1 * (8,12) (8,14) (12,14)</td>
</tr>
<tr>
<td>Slide 2</td>
<td>* 1 1 1 * 1 (8,9) (8,13) (9,13)</td>
</tr>
<tr>
<td>Slide 3</td>
<td>1 * 1 1 * 1 * (9,10) (9,14) (10,14)</td>
</tr>
<tr>
<td>Slide 4</td>
<td>* 1 * 1 * 1 (8,10) (8,11) (10,11)</td>
</tr>
<tr>
<td>Slide 5</td>
<td>1 * 1 * 1 1 (9,11) (9,12) (11,12)</td>
</tr>
<tr>
<td>Slide 6</td>
<td>1 1 1 * * 1 (10,12) (10,13) (12,13)</td>
</tr>
</tbody>
</table>

Table 2. The maximum weights of patterns that are full sensitivity to \( x \) SNPs and \( k \) free mismatches

| Sensitivity threshold | Periodic pattern length \( |P| \) |
|-----------------------|-----------------|
| k=2                   | 2 3 4 5 6 7 8 9 10 |
| k=3                   | 2 2 3 4 5 6 6 7 7 |
| k=4                   | 1 1 1 2 3 3 3 4 4 5 |

2.3 Periodic seeds: generalization, indexing and extendability

2.3.1 Generalization for different read lengths

That the full sensitive periodic seeds generalize to all lengths is a function of their repeating pattern. For example, sliding the following seed (length 28) six times generates seven subsequences that provide full sensitivity to two mismatches for a 34 bp read:

\[
\text{Positions 8–14 six times, covers all the 21 pair of positions exactly once. Therefore, this pattern is locally optimal, providing local full sensitivity to two mismatches.}
\]

In fact, local full sensitivity of this pattern implies global full sensitivity of the periodic spaced seed because every position in the read shares a similar relationship with some position within 8–14. Formally, for any read position \( i \) (excluding the boundaries of the reads), there exists a position \( j, 8 \leq j \leq 14 \), where \( i \mod 7 = j \mod 7 \) and such that when \( i \) is allowed to ** exactly once. Therefore, this pattern is locally optimal, providing local full sensitivity to two mismatches.

To achieve full sensitivity a periodic seed will slide \( |P| \) times and generate \( |P| \) subsequences on a read. As shown in Figure 2, this is accomplished by keeping open the last \( |P| \) positions at the end of the read. However, as periodic seeds maintain full sensitivity when generalized, these empty positions are not required. This is shown with a different example in Figure 4, each slide seeds can be extended to the end of the read so that all but the last are likely to have increased seed weight. In the example shown in Figure 4, seed extension results in weight increasing from the minimum of 14 to between 14 and 19 which correspond to an approximately five times faster mapping performance for the whole human genome.

To take the advantage of the extended seed weight without additional requirements to memory, PerM combines hashing with the binary search. Our special data structure consists of a hash table which stores pointers to the starting position of each bucket in an index table which stores the addresses of reads or the genome locations. In the example shown in Figure 4, we preprocess a genome using the maximum weight seed with \( W = 19 \) to generate one subsequence for every position in the genome. The prefixes of each subsequence (e.g. the first 14 bases) is used to hash the location into a bucket in the index table. Inside each bucket, multiple subsequences with the same prefix are stored lexicographically and stored in the index table. Note that within each bucket, we can search for exact matches of variable-length strings using the binary search. Thus, we applied the extended variable-weight seeds to each read to generate variable-length subsequences, and use the extended bases after 14 to find the exact matches using the binary search in the index table. This method results in seed weight which varies from 14 to 19 in the example shown in Figure 4.
2.5 Implementation and bitwise encoding

2.5.1 Quality scores and paired end reads If reads include a corresponding quality score, file PerM will evaluate mappings by their alignment scores according to the quality score of each base. PerM can also filter out poor quality reads before mapping. If paired end reads and bounds for their separation distance are given as input, PerM will output both the best alignments within and beyond the separation bounds. Each alignment will be stored in a separate file.

2.5.2 Base to color encoding PerM encodes each read into two 64 bit words with two bits for each base, that is, A= (0,0), C = (0,1), G = (1,0) and T= (1,1). We adopt RMAP’s method (Smith et al., 2008) to encode these two bits separately in two words. Similarly, each base on the reference genome is encoded into two 64 bit words with two bits for each base, that is, A= (0,0), C = (0,1), G = (1,0) and T= (1,1). Each encoding enables a quick translation of encoded base to encoded color signals by using bitwise operations of ‘SHIFT’ and ‘XOR’. In the example shown in Figure 5, a five-base read ‘ATGGA’ is encoded as two binary strings, U = (01110) and V = (110)0. The color signals of this read, ‘Red Green Blue Yellow’ encoded as U’= (100)1 and V’= (1100), can be obtained by U’= U XOR (U SHIFT 1) and V’= V XOR (V SHIFT 1). This allows the reference genome to be saved as an encoded string of bases, while the corresponding color spaced encoding can be read out in few bitwise operations.

2.5.3 Color mutations for SNPs Applied Biosystems lists rules (Applied Biosystems, 2008b) to check whether two consecutive mismatches are valid in their indication a base substitution (SNP). Only one-third of possible combinations of consecutive color mismatches are valid for a base substitution. For example, if ‘AATAA’ is encoded into the colors ‘BRRB’ and mapped to ‘BBBB’ the reference may have resulted from ‘AAAAA’ meaning a SNP from ‘A’ to ‘T’ causing two identical color mismatches, Blue ⇔ Red. However, if ‘BRRB’ is mapped to ‘BBGB’, a SNP cannot be present because the nucleotide sequence which corresponds to ‘BBGB’ is ‘AAACC’. Instead, this is likely the result of two color sequencing errors. For ease of explanation, the combinations of consecutive mismatches are classified into three types:

- Type I: Blue ⇔ Red or Green ⇔ Yellow
- Type II: Blue ⇔ Green or Red ⇔ Yellow
- Type III: Blue ⇔ Yellow or Green ⇔ Red

Only when both color mismatches are of the same type, does it indicate a valid SNP. These three types of consecutive mismatches correspond to the three classes of base substitutions, (i) Transversion I [A ⇔ T or G ⇔ C]; (ii) Transversion II [A ⇔ G or C ⇔ T]; (iii) Transition, respectively. The validation of SNP candidates is simple given our encoding method. Given two colors, one encoded into two bits b1 and b2, and the other into two bits b1’ and b2’, the three types of SNPs can be determined by the following bitwise operations:

- Type I: iff (B1 XOR b1) AND (NOT (B2 XOR b2)) = 1
- Type II: iff (NOT (B1 XOR b1)) AND (B2 XOR b2) = 1
- Type III: iff (B1 XOR b1) AND (NOT (B2 XOR b2)) = 1

3 RESULTS

3.1 Results of seed design

As shown in the experimental results, the periodic spaced seeds used in PerM outperform the seeds used in MAQ in terms of mapping speed and sensitivity for both Illumina and SOLiD data. Table 3
Table 3. PerM’s single periodic spaced seeds for SOLiD 34-color reads

<table>
<thead>
<tr>
<th>Seed name</th>
<th>Seed patterns parenthesized according to their repeats</th>
<th>Seed weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_{1,1}</td>
<td>(I</td>
<td>l</td>
</tr>
<tr>
<td>S_{1,2}</td>
<td>(I</td>
<td>l</td>
</tr>
<tr>
<td>S_{1,3}</td>
<td>(I</td>
<td>l</td>
</tr>
<tr>
<td>S_{2,0}</td>
<td>(I</td>
<td>l</td>
</tr>
<tr>
<td>S_{4}</td>
<td>(I</td>
<td>l</td>
</tr>
</tbody>
</table>

Table 4. Three seed families are compared in their ability to map 34-color SOLiD reads to a preprocessed human genome

<table>
<thead>
<tr>
<th>Seed name</th>
<th>No. of index tables</th>
<th>No. of queries per read</th>
<th>Seed weight</th>
<th>Extended weights</th>
<th>E(Rand Hits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_{2}</td>
<td>1</td>
<td>7</td>
<td>16</td>
<td>16–20</td>
<td>1.89</td>
</tr>
<tr>
<td>C_{2}</td>
<td>3</td>
<td>6</td>
<td>16</td>
<td>8.38</td>
<td>3.61</td>
</tr>
<tr>
<td>S_{1,1}</td>
<td>1</td>
<td>10</td>
<td>14</td>
<td>14–19</td>
<td>68.91</td>
</tr>
<tr>
<td>F_{3}</td>
<td>1</td>
<td>11</td>
<td>12</td>
<td>12–16</td>
<td>627.25</td>
</tr>
<tr>
<td>C_{4}</td>
<td>4</td>
<td>10</td>
<td>12</td>
<td>3576.28</td>
<td>1215.67</td>
</tr>
<tr>
<td>S_{2,0}</td>
<td>1</td>
<td>11</td>
<td>12</td>
<td>12–16</td>
<td>534.42</td>
</tr>
<tr>
<td>C_{4}</td>
<td>4</td>
<td>15</td>
<td>10</td>
<td>8.8</td>
<td>858.30</td>
</tr>
<tr>
<td>F_{4}</td>
<td>1</td>
<td>10</td>
<td>8</td>
<td>8–11</td>
<td>216.007</td>
</tr>
</tbody>
</table>

An index table is, by definition, an array of N index for a genome of N base pairs. Thus, C_{2} requires three index tables as shown in Figure 1.

Table 4 displays our fixed weight periodic spaced seeds generalized to 34-color SOLiD reads. F_{2} denotes a seed full sensitive to k mismatches, while S_{1,1} denotes a SOLID-specific seed full sensitive to k consecutive color mismatches (SNPs) and k free color mismatches. Table 3 groups F\_2 and S_{1,1} into one category because both are full sensitive to reads with one SNP and one color error. However, S_{1,1} achieves higher weight than F_{2} weight by taking advantage of SNP’s signature in color space. The introduction of positional restriction at one mismatch at the SNP locus significantly reduces the number of combinations of three mismatches, leading to the higher seed weight. Similarly, S_{2,0} and F_{4} are both full sensitive to two SNPs, but S_{2,0} provides an increase of four (12 to 8) in seed weight. Thus, the design of seeds specifically for the color space will provide a significant advantage in mapping speed.

3.2 Theoretical performance of periodic spaced seeds

Table 4 further compares the F-seed family, the S-seed family and the conventional C_{k} seed method described in Section 1. Our comparison is based on the most common implementation where the human genome is preprocessed and 34-color SOLID reads are divided into k+2 fragments for mapping.

3.2.1 Memory requirements

As shown in Table 4, PerM’s use of a single seed results in the requirement of a single index table to preprocess the human genome no matter the sensitivity requirement, compared with three to five index tables for the conventional method C_{k}. The use of single periodic spaced seeds allows us to preprocess the human genome efficiently into 4.5 bytes per base, and load it to 14 GB of memory, without the swapping of index tables between disk and memory.

3.2.2 Running time

The total running time of a mapping project can be divided into two major components:

1. Preprocessing: the time to preprocess the reference genome (or the reads set) into one or more index tables.
2. Mapping: the total time to find matches in the index tables for all queried subsequences, and the time to examine all matches using the full read-genome substring alignments.

PerM’s requirement of a single index table results in faster preprocessing time than methods which use the conventional multi-seed, multi-table approach. Mapping time consists of two parts: the time to query each seed-induced subsequence and to validate matches which result in true alignments, and the time to examine all matches and ignore matches that result from random hits. The former is fixed as the number of true alignments is constant given a particular sensitivity level, while the latter is related directly to the seed weight. Ideally, mapping time is largely spent on the matching and validation of true alignments, but if the seed weight is insufficient, the examination of random hits will dominate the running time. As listed in last column of Table 4, the expected number of random hits per read can grow so large that most of the running time is wasted on filtering out the random hits. For example, using F_{2} on the human genome will require the examination of approximately 627 random hits per read which will result in drastically slower performance than the F_{2} which is expected to examine fewer than two random hits per read. Thus, the number of random hits is the most important indicator of the actual running time. Table 4 shows that the weight increase associated with extended variable-weight periodic spaced seeds will result in a large reduction in random hits and significantly faster running times. As expected, this increased efficiency is greatest for the S_{1,1} family, when we compare S_{2,0} with C_{k}, we expect 161 times fewer random hits for the periodic seed.

3.3 Experimental results

We performed genome-scale comparison with two popular mapping programs, MAQ (version 0.6.6; Li et al., 2008) and Bowtie (version 0.10.0; Langmead et al., 2009). Both Illumina and SOLID reads from The 1000 Genomes Project were used for mapping the human genome. We also compared PerM with SOCS, a mapping program designed specifically for ABI SOLID reads.

3.3.1 Genome-scale mapping with SOLID reads

We mapped 5 million 35-color SOLID reads (the first 5 million reads in the NCBI dataset ERR000455 are available on our web site) to the whole human genome. Overall, we were able to map 58% of the reads (2.94 M) with five or fewer mismatches. Over 78% of the mapped reads included at least one mismatch and 22% of the mapped reads have four or five mismatches in their best alignment to the reference, indicative of a high machine error rate. Considering that each SNP causes consecutive mismatches, it is likely that the majority of reads which cover SNP loci will contain at least three color mismatches. Thus, the detection of genomic variation with SOLID reads requires far greater sensitivity than necessary for Illumina data. For this reason, PerM offers the seed S_{1,1} to maximize seed weight while...
All PerM seeds provide a minimum of full sensitivity to two mismatches and report

Table 6. Running time comparison of mapping the 35 bp SOLiD reads to the whole human genome

<table>
<thead>
<tr>
<th>Program</th>
<th>Seed/mode</th>
<th>weight</th>
<th>(Full) Sensitivity</th>
<th>Speed (M/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PerM</td>
<td>F2</td>
<td>16–20</td>
<td>2 colors</td>
<td>3.53</td>
</tr>
<tr>
<td>PerM</td>
<td>S1,1</td>
<td>14–19</td>
<td>1 base + 1 color</td>
<td>1.17</td>
</tr>
<tr>
<td>PerM</td>
<td>F3</td>
<td>12–16</td>
<td>3 colors</td>
<td>0.75</td>
</tr>
<tr>
<td>MAQ</td>
<td>-</td>
<td>14</td>
<td>2 colors</td>
<td>0.56</td>
</tr>
</tbody>
</table>

still maintaining full sensitivity to reads which contain one SNP and a color error. For each seed, Table 5 lists the discovery rate for alignments containing five or fewer mismatches. The small difference in the discovery rates of S1,1 and F3 provides further reason to use S1,1 for SOLiD data.

MAQ does not provide an alignment option which adheres to the definition of full sensitivity to greater than two mismatches. Instead, MAQ aligns SOLiD reads by finding hits using a seed similar to C2 (shown in Fig. 1), and checks each for the possibility of a feasible base-color alignment. It should be noted that each SOLiD read includes thirty-five colors preceded by the base of the primer used to synthesize the read. Thus, the first color represents the transition from the primer (which is not part of the reference) to the first base on the reference, leaving only thirty four colors which represent transitions on the reference. MAQ’s current implementation uses only these thirty four colors for alignment. Unfortunately, each transition on the reference. MAQ’s current implementation uses only these thirty four colors for alignment. Unfortunately, each transition on the reference.

Table 7. Running time comparison of mapping the Illumina reads with different read lengths and seeds to the whole human genome

<table>
<thead>
<tr>
<th>Length</th>
<th>Weight</th>
<th>Reads/h</th>
<th>Weight</th>
<th>Reads/h</th>
<th>Weight</th>
<th>Reads/h</th>
</tr>
</thead>
<tbody>
<tr>
<td>36 bp</td>
<td>2</td>
<td>18–21</td>
<td>14</td>
<td>14</td>
<td>0.49</td>
<td>4.43</td>
</tr>
<tr>
<td>40 bp</td>
<td>2</td>
<td>27–28</td>
<td>14</td>
<td>14</td>
<td>0.55</td>
<td>3.87</td>
</tr>
<tr>
<td>47 bp</td>
<td>2</td>
<td>38–39</td>
<td>14</td>
<td>14</td>
<td>0.67</td>
<td>2.64</td>
</tr>
</tbody>
</table>

The default mode of Bowtie is equivalent to -k 1. The -v k mode is set with -a –best –

<table>
<thead>
<tr>
<th>Seed/mode</th>
<th>weight</th>
<th>(Full) Sensitivity</th>
<th>Speed (M/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2</td>
<td>18–21</td>
<td>5.92 M</td>
<td>20.1 M</td>
</tr>
<tr>
<td>MAQ</td>
<td>14</td>
<td>0.49 M</td>
<td>4.43</td>
</tr>
<tr>
<td>Bowtie -v3</td>
<td>4.28 M</td>
<td>3.38 M</td>
<td>1.63 M</td>
</tr>
<tr>
<td>Bowtie -v2</td>
<td>9.27 M</td>
<td>7.95 M</td>
<td>7.20 M</td>
</tr>
</tbody>
</table>

3.3.2 Comparison: PerM and MAQ

As shown in Tables 6 and 7, PerM is able to map both SOLiD and Solexa reads significantly faster than MAQ while offering greater levels of full sensitivity. The differences in performance are a testament to the benefit of extendable periodic spaced seeds which provide greater seed weights than the fixed length consecutive seeds implemented in MAQ. This increase in seed weight allows PerM to avoid the bottleneck which results from the many random hits present on a large genome. It should also be noted that MAQ preprocesses reads, requiring it to build an index table for each mapping project while PerM can reuse the same index because it preprocesses the genome.

3.3.3 Comparison: PerM and Bowtie

Although PerM and Bowtie both index the genome, PerM finds full sensitive alignments through seed subsequence matching while Bowtie uses a modified exact matching algorithm and quality-aware backtracking to report alignments. As shown in Table 7, Bowtie’s performance slows down when aligning long reads because of the increase in backtracking steps required to find inexact alignments. PerM’s performance is primarily a result of it’s seed weight, which is maximized for any
Table 8. Comparing of PerM and SOCS in chromosome-level reference

<table>
<thead>
<tr>
<th>Full sensitivity</th>
<th>PerM</th>
<th>SOCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 color mis</td>
<td>11 min 46 s</td>
<td>16–20</td>
</tr>
<tr>
<td>1 base + 1 color mis</td>
<td>23 min 0 s</td>
<td>14–19</td>
</tr>
<tr>
<td>3 color mis</td>
<td>32 min 41 s</td>
<td>12–16</td>
</tr>
</tbody>
</table>

The running time includes pre-processing and I/O. The memory usage of both the programs is <2 GB. The tests are performed on Sun X4600, Opteron, 2.6 GHz, using single node and thread.

3.3.4 Comparison: PerM with SOCS PerM was compared with a program dedicated to SOLiD reads, SOCS (version 1.2.1). Both PerM and SOCS provide full sensitivity to three mismatches, but SOCS does not provide sufficient seed weight to map reads to the entire genome. For this reason, we mapped the 5 million 35 bp SOLiD reads used in 3.3.1 to chromosome X. Eight percent of the reads included a mapping to chromosome with three of fewer substitutions. Table 8 lists the mapping times for different sensitivity levels. PerM’s faster running time in comparison to SOCS is primarily the result of much higher seed weight.

3.3.5 Genome preprocessing Genome preprocessing time is linear with respect to the size of the reference regardless of the seed used. PerM requires 3h 30 min to index the whole human genome to 14 GB of memory. In comparison, Bowtie requires 4h 47 min to build a compressed 2 GB genome index. However, for large-genome resequencing projects, preprocessing time is negligible in comparison with mapping time. Once the genome has been preprocessed, its index can be shared and reused by multiple processors to map different read sets. Both PerM and Bowtie use multiple cores in one computer to map read sets in parallel by querying the same genome index table in the shared memory. Thus, on a server with shared memory architecture, PerM is more memory efficient in terms of ‘memory per CPU’ compared with MAQ, despite a 14 GB index table.

4 DISCUSSION

PerM provides highly efficient mapping solutions for genome-scale mapping projects involving Illumina or SOLiD data. PerM owes its performance primarily to the use of single periodic spaced seeds which are capable of providing sufficient weight and sensitivity to significantly increase genome-scale mapping performance in comparison with other mapping programs.

However, for applications that require full sensitivity to many mismatches (k ≥ 4) on a short read, single periodic seeds may prove incapable of providing efficient mapping performance. In this situation, the costly step of hashing to multiple index tables may be necessary to increase seed weight and eliminate a bottleneck in the checking step. Already a topic of much interest (Li et al., 2004; Nicolas and Rivals, 2008; Noé and Kucherov, 2004; Sun and Buhler, 2005; Yang et al., 2004), Ma and Yao (2008) showed that the optimization of multiple seeds cannot be easier than the Golomb Ruler Design problem, considered likely to be NP-hard. Thus, although we cannot guarantee optimality over the entire search space, we propose three methods to design high weight multiple seeds: a constrained exhaustive search, a reduction to the integer programming problem and a ‘tuples-grouping’ algorithm. These methods and additional performance analysis and experiments are discussed in detail in the Supplementary Material.

ACKNOWLEDGEMENTS

The authors would like to thank Dr Zhengyu Xuan and Dr Richard McCombie from the Genome Center of Cold Spring Harbor Laboratory and Dr Andrew Smith at USC for providing us with data and helpful suggestions. We would also like to thank Zheng Zhang from ABI as well as graduate students Dazhe Meng and Zach Frazier from USC for their helpful discussion.

Funding: National Institutes of Health Center of Excellence in Genomic Sciences.

Conflict of Interest: none declared.

REFERENCES


