BIOINFORMATICS ORIGINAL PAPER
Vol. 28 no. 14 2012, pages 1838–1844
doi:10.1093/bioinformatics/bts280

Sequence analysis
Advance Access publication May 7, 2012

Exploring single-sample SNP and INDEL calling with whole-genome *de novo* assembly

Heng Li
Medical Population Genetics Program, Broad Institute, 7 Cambridge Center, MA 02142, USA

Associate Editor: Michael Brudno

ABSTRACT

Motivation: Eugene Myers in his string graph paper suggested that in a string graph or equivalently a unitig graph, any path spells a valid assembly. As a string/unitig graph also encodes every valid assembly of reads, such a graph, provided that it can be constructed correctly, is in fact a lossless representation of reads. In principle, every analysis based on whole-genome shotgun sequencing (WGS) data, such as SNP and insertion/deletion (INDEL) calling, can also be achieved with unitigs.

Results: To explore the feasibility of using *de novo* assembly in the context of resequencing, we developed a *de novo* assembler, *fermi*, that assembles Illumina short reads into unitigs while preserving most of information of the input reads. SNPs and INDELs can be called by mapping the unitigs against a reference genome. By applying the method on 35-fold human resequencing data, we showed that in comparison to the standard pipeline, our approach yields similar accuracy for SNP calling and better results for INDEL calling. It has higher sensitivity than other *de novo* assembly based methods for variant calling. Our work suggests that variant calling with *de novo* assembly can be a beneficial complement to the standard variant calling pipeline for whole-genome resequencing. In the methodological aspects, we propose FMD-index for forward–backward extension of DNA sequences, a fast algorithm for finding all super-maximal exact matches and one-pass construction of unitigs from an FMD-index.

Availability: http://github.com/lh3/fermi

Contact: hengli@broadinstitute.org

Received on April 3, 2012; revised on May 2, 2012; accepted on May 3, 2012

1 INTRODUCTION

The rapidly decreasing sequencing cost has enabled whole-genome shotgun (WGS) resequencing at an affordable price. Many software packages have been developed to call variants, including SNPs, short insertions and deletions (INDELs) and structural variations (SVs), from WGS data. At present, the standard approach to variant calling is to map raw sequence reads against a reference genome and then to detect differences from the reference. It is well established and has been proved to work from a single sample to thousands of samples (1000 Genomes Project Consortium, 2010). Nonetheless, a fundamental flaw in this mapping-based approach is that mapping algorithms ignore the correlation between sequence reads. They are unable to take full advantage of data and may produce inconsistent outputs which complicate variant calling. This flaw has gradually attracted the attention of various research groups who subsequently proposed several methods to alleviate the effect, including post-alignment filtering (Ossowski *et al.*, 2008), iterative mapping (Manske and Kwiatkowski, 2009), read realignment (Albers *et al.*, 2010; Depristo *et al.*, 2011; Homer and Nelson, 2010; Li, 2011) and local assembly (Carnevali *et al.*, 2011). However, because these methods still rely on the initial mapping, it is difficult for them to identify and recover mismapped or unmapped reads due to high-sequence divergence, long insertions, SVs, copy number changes or misassemblies of the reference genome. They have not solved the problem from the root.

Another distinct approach to variant calling that fundamentally avoids the flaw of the mapping-based approach is to assemble sequence reads into contigs and to discover variants via assembly-to-assemby alignment. It was probably more widely used in the era of capillary sequencing. The assembly based method became less used since 2008 due to the great difficulties in assembling 25 bp reads, but with longer paired-end reads and improved methodology, *de novo* assembly is reborn as the preferred choice for variant discovery between small genomes.

For variant discovery between human genomes, however, the assembly based approach has not attracted much attention. Assembling a human genome is far more challenging than assembling a bacterial genome, firstly due to the sheer size of the genome, secondly to the rich repeats and thirdly due to the diploidy of the human genome. Many heuristics effective for assembling small genomes are not directly applicable to the human genome assembly. As a result, only a few *de novo* assemblers have been applied on human short-read data. Among them, ABySS (Simpson *et al.*, 2009), SOAPdenovo (Li *et al.*, 2010) and SGA (Simpson and Durbin, 2012), as of now, do not explicitly output heterozygotes. Although in theory it is possible to recover heterozygotes from their intermediate output, it may be difficult in practice as the assemblers may not distinguish heterozygotes from sequencing errors. Cortex (Iqbal *et al.*, 2012) is specifically designed for retaining heterozygous variants in an assembly, but it may be missing heterozygotes. ALLPATHS-LG (Gnerre *et al.*, 2011) also paid particular attention to keep heterozygotes, but it still has a relatively low sensitivity. In addition, ALLPATHS-LG only works with reads from libraries with distinct insert size distributions and prefers read pairs with mean insert size below three times of the read length, whereas many resequencing projects do not meet these requirements and thus ALLPATHS-LG may not be applied or work to the best performance. Even if we also include *de novo* assemblers developed for capillary sequence reads, the version of the Celera assembler used for assembling the HuRef genome (Levy *et al.*, 2007) is the only one that retains heterozygotes while capable of assembling a mammalian genome. At last, one may think to map sequence reads back to the assembled contigs to recover heterozygous events, but this procedure will be affected by the same flaw of read mapping. To the best of our knowledge, no existing *de novo* assemblers are able to achieve the sensitivity of the standard mapping-based approach for a diploid mammalian genome.

In this article, we will show that the assembly based variant calling can achieve an SNP accuracy close to the standard mapping approach and have particular strength in INDEL calling, confirming previous studies (Iqbal *et al.*, 2012). In addition, the *de novo* assembly algorithm, *fermi*, developed for this practice is also a capable assembler for human assembly.

2 METHODS

The methods section is organized as follows. We first review the history of *de novo* assembly in the theoretical aspects, which leads to the rationale behind fermi: to use unitigs as a lossless representation of reads. We then summarize the notations used in the article and introduce bidirectional FM-index for DNA sequences. We will present several algorithms for assembling using the bidirectional FM-index. The key algorithm is based on previous works (Simpson and Durbin, 2010), but we need to adapt it to our new index. We also remove the recursion in the original algorithm. Finally we will discuss practical concerns in implementation.

2.1 Theoretical background

2.1.1 A history of the OLC paradigm Computer assisted sequence assembly can be dated back to the late 1970s (Gingeras *et al.*, 1979; Staden, 1979). In 1984, Peltola *et al.* first formulated the DNA assembling problem as finding the shortest string (the assembly) such that each sequence read can be mapped to the assembly within a required error rate. To solve the problem, they proposed a three-step procedure, which is essentially the overlap-layout-consensus (OLC) approach.

Myers (1995) pointed out that reducing DNA assembly to a shortest string problem is flawed in the presence of repeat. He further proposed the concept of *overlap graph*, where a vertex corresponds to a read and a bidirectional edge to an overlap. Naively, the DNA assembling problem can be cast as finding a path in the overlap graph such that each vertex/read is visited exactly once (though edge/overlap caused by repeats are not required to be traversed), equivalent to a Hamilton path problem which is known to be NP-complete. This has led many to believe that the OLC approach is theoretically crippled.

However, it is worth pointing out that although the assembly problem can be reduced to a Hamilton path problem, it can be reduced to other problems as well and in practice almost no assemblers try to solve a Hamilton path problem. We note that a fundamental difference between a generic graph and an overlap graph is the latter can be transitively reduced while retaining the read relationship. More formally, if $v_1 \to v_2$, $v_2 \to v_3$ and $v_1 \to v_3$ are all present, edge $v_1 \to v_3$ is said to be *reducible*. When we removed all the contained reads and reducible edges, a procedure called *transitive reduction*, the resulting graph is still a loyal representation of the overlap graph (Myers, 1995), but the path corresponding to the assembly is not a Hamilton path any more because reads from repetitive regions need to be traversed multiple times.

In a transitively reduced graph, if there exists $v_1 \to v_2$ with the out-degree of v_1 and in-degree of v_2 both equal to 1, we are able to merge v_1 and v_2 into one vertex without altering the topology of the graph. After we performed all possible merges, we get a *unitig graph* in which each vertex corresponds to a *unitig*, representing a maximal linear sequence that can be resolved by reads. Multiple copies of a repeat may be collapsed to a single unitig. The concept of unitig helps to greatly simplify an assembly graph. It has played a central role in the Celera assembler (Myers *et al.*, 2000).

Finding the optimal tour in a unitig graph is still NP-hard (Medvedev *et al.*, 2007), but such a formulation may not be useful in practice as we can rarely assemble the entire genome into one string. A more practical solution is to compute a traversal count for each edge by solving a minimum cost network flow problem (Myers, 2005) and to drop edges with zero count as false overlaps. In the resulting graph, each unambiguous path can be considered to spell a contig.

Computing traversal counts in a transitively reduced graph can be conducted in small subgraphs separated by some unambiguous edges. The overall time complexity is not much worse than linear—the worst case almost never happens globally. However, deriving an overlap graph takes $O(N^2)$ time, where N is the number of reads, and transitive reduction takes at least $O(E)$ time, where E is the number of edges which is usually much larger than N. This still makes an OLC-based approach less favorable in short-read assembly where N can be of the order of 10^9.

A breakthrough achieved by Simpson and Durbin (2010) finally solved this last remaining problem at least when we only consider exact overlaps. These authors developed an $O(N)$ algorithm to find all the irreducible edges, effectively replacing the overlapping and transitive reduction phases.

In summary, in the OLC paradigm, contig sequences can be constructed in a time roughly linear in the total length of reads, though deriving a single-assembled sequence is NP-hard in theory.

2.1.2 De Bruijn graph and read coherence The de Bruijn graph is an alternative graph representation of sequence reads (Idury and Waterman, 1995). It can be trivially constructed with a simple linear-time algorithm and finding the optimal tour has polynomial-time solutions. These make the de Bruijn graph approach very attractive for assembling many short reads.

However, de Bruijn is ‘lossy’. From a theoretical point view, a de Bruijn graph is equivalent to an overlap graph built by splitting a long read into overlap k-mers and requiring $(k-1)$-mer exact overlaps between non-redundant k-mers. Such a graph does not have transitive edges. Because long reads all effectively work as k-bp reads in a de Bruijn graph, long-range information is lost. As a result, a path in the graph may be invalidated by reads. In contrast, in a unitig graph or equivalently a string graph each path models a valid assembly from input reads. Myers (2005) called this property of path consistency as *read coherence*.

Losing long-range information in reads, a de Bruijn graph by itself has reduced power to resolve short repeats. This flaw is usually amended by solving a *Eulerian superpath problem* (Pevzner *et al.*, 2001) whereby we map reads back to the graph and bisect repeats shorter than the reads, a procedure some also called as *read threading*. Many de Buijn graph-based assemblers essentially take this strategy (Chaisson *et al.*, 2009; Li *et al.*, 2010; Zerbino *et al.*, 2009), though they may use different terminologies. With read threading, it is possible to transform a de Bruijn graph to a coherent graph, but finding the optimal solution is known to be NP-hard (Medvedev *et al.*, 2007) and may be complex to implement given rich repeat structures.

2.1.3 Concluding remark We noted that we only focused on the theoretical aspects of *de novo* assembly. In practice, many assemblers derived the final assembly by applying heuristics on the simplified graph instead of solving a network flow problem or a Eulerian problem. Furthermore, correcting errors, utilizing read pairs and controlling memory usage all pose challenges to large-scale *de novo* assembly. Many practical problems are not solved perfectly. *De novo* assembly is still a field under active development.

2.2 Rationale

Being coherent, a perfectly constructed unitig graph annotated with per-unitig read counts in fact encapsulates all the information of reads and encodes no information invalidated by reads. In this sense, any unitig-based analysis has an equivalent read-based analysis, and vice versa. This article just uses this property to explore the applications for which we usually rely on reads.

2.3 Strings and FM-index

2.3.1 Strings with multiple sentinels Let $\Sigma=\{\texttt{\$},\texttt{A},\texttt{C},\texttt{G},\texttt{T},\texttt{N}\}$ be the *alphabet* of DNA sequences with a predefined lexicographical order $\texttt{\$}<\texttt{A}<\texttt{C}<\texttt{G}<\texttt{T}<\texttt{N}$, where 'N' represents an ambiguous base and '\$' is a sentinel that marks the end of a string. An element in Σ is called a *symbol* and a sequence of symbols is called a *string*. Given a string T, let $|T|$ be the length of the string, $T[i]$, $i=0,\ldots,|T|-1$, be the i-th symbol in the string, $T[i,j]$, $0\le i\le j<|T|$, be a substring and $T_i=T[i,|T|-1]$ be a suffix of T (Table 1). Following the definition by Siren (2009), we define a string terminated with '\$' as a *text*. A text may have multiple sentinels. In a text T, if $T[i]=\$$ and $T[j]=\$$, we mandate $T[i]<T[j]$ if and only if $i<j$. Thus when we compare two suffixes of T, we do not need to compare beyond a sentinel because each sentinel has a different lexicographical rank.

For two strings P and W, let $P\circ W$ be their string concatenation. We may sometimes write $P\circ W$ as PW if it is unambiguous in the context. Given an ordered set of texts, we call their ordered string concatenation as a *collection*, which is also a text. For example, suppose we have two reads. The first is ACG and the second is GTG. The collection of the two reads is $T=\texttt{ACG\$GTG\$}$. Suffix $T_2<T_6$ because the first sentinel is lexicographically smaller than the second.

For convenience, we assign an integer from 0 to 5 to '\$', 'A', 'C', 'G', 'T' and 'N', respectively. We may use both the integer and the letter representations throughout the article. In addition, given a symbol a, we define $\overline{a}$ as the Watson–Crick complement of a. We regard the complement of '\$' and 'N' is identical to itself.

2.3.2 FM-index The *suffix array* S of text T is a permutation of integers between 0 and $|T|-1$, where $S(i)$, $0\le i<|T|$, is the position of the i-th smallest suffix of T. Given a string P, the *suffix array interval* $I^l(P),I^u(P)]$ of P in T is defined as

$$I^l(P)=\min\{k:P\text{ is the prefix of }T_{S(k)}\}$$

$$I^u(P)=\max\{k:P\text{ is the prefix of }T_{S(k)}\}$$

For convenience, we also define $I^s(P)=I^u(P)-I^l(P)+1$ as the size of the interval.

The *Burrows–Wheeler Transform* (Burrows and Wheeler, 1994), or *BWT*, of T is a permutation of symbols in T. The BWT string B is computed as $B[i]=T[S(i)-1]$ for $S(i)>0$ and $B[i]=\$$ otherwise. Given a text T, also define the accumulative count array $C(a)$ as the number of symbols in T that are lexicographically smaller than a, and the occurrence array $O(a,i)$ as the occurrence of symbols a in $B[0,i]$.

FM-index (Ferragina and Manzini, 2000) is a compressed representation of the BWT B, the occurrence array $O(a,i)$ and the suffix array $S(i)$. The key property of FM-index is

$$I^l(aP)\ =C(a)+O(a,I^l(P)-1) \qquad (1)$$

$$I^u(aP)\ =C(a)+O(a,I^u(P))-1 \qquad (2)$$

and $I^l(aP)\le I^u(aP)$ if and only if aP is a substring of T. We note that these two equations are different from the ones in our previous paper (Li and Durbin, 2009) in that $C(a)$ and $O(a,i)$ defined here include the sentinels, but the two arrays in the previous paper exclude them.

Table 1. Notations

Symbol	Description
T	String: $T=a_0a_1\ldots a_{n-1}$ with $a_{n-1}=\$$
$\|T\|$	Length of T including sentinels: $\|T\|=n$
$T[i]$	The i-th symbol in string T: $T[i]=a_i$
$T[i,j]$	Substring: $T[i,j]=a_i\ldots a_j$
T_i	Suffix: $T_i=T[i,n-1]$
S	Suffix array; $S(i)$ is the position of the i-th smallest suffix
B	BWT: $B[i]=T[S(i)-1]$ if $S(i)>0$ or $B[i]=\$$ otherwise
$C(a)$	Accumul. count array: $C(a)=\|\{0\le i\le n-1:T[i]<a\}\|$
$O(a,i)$	Occurrence array: $O(a,i)=\|\{0\le j\le i:B[j]=a\}\|$
$P\circ W$	String concatenation of string P and W
Pa	String concatenation of string P and symbol a: $Pa=P\circ a$
$\overline{P}$	Watson–Crick reverse complement of DNA string P

Given a collection $T=Q_0Q_1\ldots Q_{n-1}$, we can retrieve sequence Q_i in linear time with Algorithm 1 (Mäkinen *et al.*, 2009). The second return value is the rank of Q_i which equals $|\{Q_j:Q_j<Q_i\}|$.

Algorithm 1: Sequence retrieval

```
Input: Sequence index $i\ge 0$; $B$, $O$ and $C$ defined in the text
Output: Sequence $P$ and $k$, the rank of $P$

Function GetSeq($i$) begin
    $k\leftarrow i$;
    $P\leftarrow$ empty string;
    while true do
        $a\leftarrow B[k]$;
        $k\leftarrow C(a)+O(a,k)-1$;
        if $a=0$ then
            return $(P,k)$
        $P\leftarrow aP$
```

2.4 FMD-index

Given DNA texts $R_0,\ldots,R_{n-1}$, define $T=R_0\overline{R}_0R_1\overline{R}_1\ldots R_{n-1}\overline{R}_{n-1}$ as the *bidirectional collection* of the texts. We call the FM-index of T as the *FMD-index* of $R_0,\ldots,R_{n-1}$ and define the *bi-interval* of a string P as $[I^l(P),I^l(\overline{P}),I^s(P)]$. We will show how to compute the bi-interval of aP and Pa when we know the bi-interval of P.

We note that when we know the bi-interval of P, $I^l(aP)$ and $I^s(aP)$ can be readily computed with Equation (1). $[I^l(\overline{aP}),I^u(\overline{aP})]$ is a sub-interval of $[I^l(\overline{P}),I^u(\overline{P})]$ because $\overline{P}$ is a prefix of $\overline{aP}=\overline{P}\circ\overline{a}$. Due to the innate symmetry of T, $I^s(\overline{cP})=I^s(cP)$ for all $c\in\Sigma$ with $\sum_c I^s(cP)=I^s(P)=I^s(\overline{P})$. We can compute $I^s(cP)$ for all $c\in\Sigma$ with Equation (1), use these interval sizes to divide $[I^l(\overline{P}),I^u(\overline{P})]$ and finally derive $[I^l(\overline{aP}),I^u(\overline{aP})]$. This completes the computation of the bi-interval of aP (Algorithm 2). Furthermore, when we backward extend P, we actually forward extend $\overline{P}$. Conversely, backward extension of $\overline{P}$ yields forward extension of P (Algorithm 3). An FMD-index is bidirectional.

In comparison to the bidirectional BWT (Lam *et al.*, 2009) which uses two FM-indices, the FMD-index builds both forward and reverse strand DNA sequences in one index. Although the FMD-index is not applicable to generic texts, it is conceptually more consistent with double-strand DNA and improves the speed of exact matching as we only need to search against one index. For example, BWA-SW (Li and Durbin, 2010) gets a 80% speedup when we adopt the FMD-index as the data structure.

2.5 Unitig construction

2.5.1 Labeling reads and overlaps Given a bidirectional collection $T=R_0\overline{R}_0\ldots R_{n-1}\overline{R}_{n-1}$, fermi labels the i-th input read R_i with an *ordered* integer pair $[k,l]$, where k is the rank of R_i and l the rank of $\overline{R}_i$. The pair $[k,l]$ can be computed by GetSeq($2i$) and GetSeq($2i+1$), respectively. Obviously, if read R_i is labeled by $[k,l]$, $\overline{R}_i$ should be labeled by $[l,k]$, with the two integer swapped.

For two reads labeled by $[k,l]$ and $[k',l']$, if the tail ($3'$ end) of read $[k,l]$ overlaps the head ($5'$ end) of $[k',l']$, we use an *unordered* integer pair $\langle l,k'\rangle$ to label the overlap. Such is a tail-to-head overlap. Similarly, we use

Algorithm 2: Backward extension

```
Input: Bi-interval $[k,l,s]$ of string $W$ and a symbol $a$
Output: Bi-interval of string $aW$
Function BackwardExt($[k,l,s],a$) begin
    for $b \leftarrow 0$ to 5 do
        $k_b \leftarrow C(b)+O(b,k-1)$
        $s_b \leftarrow O(b,k+s-1)-O(b,k-1)$
    $l_0 \leftarrow l$;
    $l_4 \leftarrow l_0+s_0$;
    for $b \leftarrow 3$ to 1 do
        $l_b \leftarrow l_{b+1}+s_{b+1}$
    $l_5 \leftarrow l_1+s_1$;
    return $[k_a,l_a,s_a]$
```

Algorithm 3: Forward extension

```
Input: Bi-interval $[k,l,s]$ of string $W$ and a symbol $a$
Output: Bi-interval of string $Wa$
Function ForwardExt($[k,l,s],a$) begin
    $[l',k',s'] \leftarrow$ BackwardExt($[l,k,s],\overline{a}$);
    return $[k',l',s']$
```

Algorithm 4: Finding irreducible overlaps

```
  Input: Read $P$ and the minimum overlap length $x$
  Output: Set of bi-intervals of reads having irreducible
          overlaps with the $3'$ end of $P$
  Function IrrOverlap($P,x$) begin
      Initialize Curr and Prev as empty arrays;
      $a \leftarrow P[|P|-1]$;
1     $[k,l,s] \leftarrow [C(a),C(\overline{a}),C(a+1)-C(a)]$;
2     for $i \leftarrow |P|-2$ to 0 do
          if $|P|-i-1 \geq x$ then
              $[k',l',s'] \leftarrow$ BackwardExt($[k,l,s],0$);
              if $s' \neq 0$ then
                  Append ($[k',l',s'],\epsilon$) to Curr;
          $[k,l,s] \leftarrow$ BackwardExt($[k,l,s],P[i]$);
      Reverse array Curr, and swap Curr and Prev;
      Finished $= \emptyset$;
      $\mathcal{I} = \emptyset$;
3     while Prev is not empty do
          Reset Curr to empty;
          for ($[k,l,s],W$) in Prev do
              if $W \in$ Finished then
4                 continue;
              $[k',l',s'] \leftarrow$ ForwardExt($[k,l,s],0$);
5             if $s' \neq 0$ then
                  Finished $\leftarrow$ Finished $\cup \{W\}$;
                  $\mathcal{I} \leftarrow \mathcal{I} \cup \{[k',l',s']\}$;
6                 continue;
              for $a \leftarrow 1$ to 5 do
                  $[k',l',s'] \leftarrow$ ForwardExt($[k,l,s],a$);
                  if $s' \neq 0$ and $[k',l',s']$ is not in Curr then
                      Append ($[k',l',s'],Wa$) to Curr;
          Swap Curr and Prev
      return IrrOvlp
```

$\langle l',k \rangle$ for a head-to-tail overlap, $\langle l,l' \rangle$ for tail-to-tail and $\langle k,k' \rangle$ for a head-to-head overlap. The four types of overlaps correspond to the four types of bidirectional edges in the bidirectional overlap graph (Myers, 1995).

2.5.2 Finding irreducible overlaps Finding irreducible overlaps plays a central role in fermi as well as in SGA. Given its importance, we present a restructured version of this algorithm (SD10; Simpson and Durbin 2010) using our notations (Algorithm 4).

In Algorithm 4, Line 1 computes the bi-interval of a single symbol. The loop at Line 2 uses backward extensions to find all the reads overlapping with the input string P. The loop at Line 3 uses forward extensions base by base to exclude reducible overlaps found at the previous step. W is this loop keeps the common substring of reads overlapping P extended from the $3'$ end of P. If in an iteration we find the sentinel of a read R (Line 5), then all the reads sharing the same W with R must overlap with both R and P and therefore their overlaps with P are reducible. In this case, no further forward extensions are necessary (Lines 4 and 6).

Similar to the original algorithm, Algorithm 4 requires that there are no contained reads. Fermi actually implements a modified version that detects reads containment on the fly, but we think the algorithm is a little overcomplicated. It is probably easier to filter contained reads first and then run Algorithm 4, as SGA does.

2.5.3 Unitig construction Unitig construction is a process of unambiguous merge of overlapped reads. If $[k,l]$ and $[k',l']$ have an irreducible overlap $\langle l,k' \rangle$ and can be unambiguously merged, we label the merged sequence with $[k,l']$; the similar can be applied to other three types of overlaps. With this simple labeling procedure, we are able to fully keep track of the graph topology during the unitig construction and without staging the graph in RAM. This procedure can also be easily multi-threaded.

2.6 Finding the SMEMs

An FMD-index can be used to find *supermaximal exact matches* (SMEMs) between a reference and a query sequence. Formally, a *maximal exact match* (MEM) is a an exact match that cannot be extended in either direction of the match. An SMEM is a MEM that is not contained in other MEMs on the query sequence. Fermi uses SMEMs to map reads back to the unitigs.

Algorithm 5 describes the details. Basically, we use forward–backward extension to extend an exact match and detect the boundary of a maximal match by tracking the change of interval sizes. Fermi implements a variant of Algorithm 5. It finds full-length read matches and can optionally exclude matches identical to the query sequence.

2.7 Other implementation details

2.7.1 Constructing FM-index To compute suffix arrays for strings with multiple sentinels, we modified an optimized implementation of the SA-IS algorithm (Nong *et al.*, 2011) by Yuta Mori. We used the established algorithm to merge BWTs of subsets of reads (Ferragina *et al.*, 2010; Hon *et al.*, 2007; Siren, 2009). The BWT string is run-length encoded with the length in the delta encoding (Elias, 1975).

2.7.2 Error correction Fermi corrects potential sequencing errors using an algorithm similar to solving the spectrum alignment problem (Pevzner *et al.*, 2001), correcting bases in underrepresented k-mers. It also shares similarity to HiTEC (Ilie *et al.*, 2011). Nonetheless, the fermi's algorithm differs in that it is quality aware and does not rely on a user defined threshold on the k-mer occurrences.

Algorithm 5: Finding SMEMs

```
Input: String $P$ and start position $i_0$; $P[-1]=0$
Output: Set of bi-intervals of SMEMs overlapping $i_0$

Function SuperMEM1($P, i_0$) begin
    Initialize Curr, Prev and Match as empty arrays;
    $[k,l,s] \leftarrow [C(P[i_0]), C(\overline{P[i_0]}), C(P[i_0]+1)-C(P[i_0])]$;
    for $i \leftarrow i_0+1$ to $|P|$ do
        if $i=|P|$ then
            Append $[k,l,s]$ to Curr
        else
            $[k',l',s'] \leftarrow$ ForwardExt($[k,l,s], P[i]$);
            if $s' \neq s$ then
                Append $[k,l,s]$ to Curr
            if $s'=0$ then
                break;
            $[k,l,s] \leftarrow [k',l',s']$
    Swap array Curr and Prev;
    $i' \leftarrow |P|$;
    for $i \leftarrow i_0-1$ to $-1$ do
        Reset Curr to empty;
        $s'' \leftarrow -1$;
        for $[k,l,s]$ in Prev do
            $[k',l',s'] \leftarrow$ BackwardExt($[k,l,s], P[i]$);
            if $s'=0$ or $i=-1$ then
                if Curr is empty and $i+1 < i'+1$ then
                    $i' \leftarrow i$;
                    Append $[k,l,s]$ to Match
            if $s' \neq 0$ and $s' \neq s''$ then
                $s'' \leftarrow s'$;
                Append $[k,l,s]$ to Curr
        if Curr is empty then
            break
        Swap Curr and Prev;
    return Match
```

Fermi corrects errors in two phases. In the first phase, it collects all 23 mer occurring 3 or more times using a top-down traversal over the trie represented by the FMD-index. For each such 23 mer, fermi counts the occurrences of the next (i.e. the 24-th) base and stores the information in a hash table with the 23 mer being the key. In the second phase, fermi processes each read by using the 23 mer hash table to correct errors by minimizing a heuristic cost function of base quality and the occurrences of the 24-th base. Roughly speaking, fermi tries to correct a low-quality base if by looking up its 23 mer prefix we know the base is different from an overwhelmingly frequent 24-th base. This algorithm can be adapted to correct INDEL sequencing errors in principle, but this has not been done. More works are needed to perform minimization efficiently.

2.7.3 Simplifying complex bubbles A *bubble* is a directed acyclic subgraph with a single source and a single sink having at least two paths between the source and the sink. A *closed bubble* is a bubble with no incomming edges from or outgoing edges to other parts of the entire graph, except at the source and the sink vertices. A closed bubble is *simple* if there are exactly two paths between the source and the sink; otherwise it is *complex*. In *de novo* assembly, a bubble is frequently caused by sequencing errors or heterozygotes. Most short-read assemblers uses a modified Dijkstra's algorithm to pop bubbles progressively. Such an algorithm works fine for haploid genomes, but it is not straightforward to distinguish heterozygotes from errors when the bubble is complex.

Fermi uses a different algorithm. It effectively performs topological sorting from the end of a vertex while keeping track of the top two paths containing most reads. A bubble is detected when every path ends at a single vertex. It then drops vertices not on the top two paths and thus turns a complex bubble to a simple one.

2.7.4 Using the paired-end information Given paired-end reads with short-insert sizes, fermi maps reads back to the unitigs with Algorithm 5. If two unitigs are linked by at least five read pairs, fermi will locally assemble the ends of unitigs together with unpaired reads pointing to the gap under a relax setting. Fermi tries to align the ends of unitigs using the Smith–Waterman algorithm, which may reveal imperfect overlaps caused by sequencing errors or heterozygotes. Fermi also uses paired-end reads to break contigs at regions without bridging read pairs. This helps to reduce misassemblies during the unitig construction.

3 RESULTS

We evaluated fermi on 101 bp paired-end reads from NA12878 (Depristo *et al.*, 2011). The total coverage of the original data is ~70-fold, but we only used half of them. We assembled the 35-fold reads with fermi on a machine with 12 CPUs and 96 GB memory in ~5 days. The peak memory usage is 92 GB.

We obtained unitigs of N50 1022 bp, totaling 3.83 Gb. After collapsing most heterozygotes and closing gaps with paired-end reads, we got longer contigs (Table 4). Unitigs are short and redundant mainly because they break at heterozygotes.

For SNP and INDEL calling, we aligned unitigs to the reference genome using BWA-SW (Li and Durbin, 2010) with command line options '-b9 -q16 -r1 -w500'. We called SNPs with the SAMtools caller and called INDELs by directly counting INDELs from the pileup output. We did not run a standard INDEL caller as short-read INDEL callers do not work well with long contig sequences.

3.1 Performance on *de novo* assembly

We obtained the HuRef capillary read assembly (Levy *et al.*, 2007) and the ALLPATHS-LG NA12878 contigs (AC:AEKP01000000) from NCBI, the SGA scaffolds from http://bit.ly/jts12878 (Simpson and Durbin, 2012) and the ABySS assembly provided by Shaun Jackman (personal communication). For both SGA and ABySS scaffolds, we split at any ambiguous bases to get contigs; for the HuRef assembly, we split at contiguous 'N' longer than 20 bp. The ABySS, fermi and SGA assemblies are derived from essentially the same input reads. ALLPATH-LG uses a superset of reads at 100-fold coverage, including reads from multiple long-insert libraries.

From Table 2, we can see that the HuRef assembly has much better contiguity than short-read assemblies. It appears to yield more alignment break points, some of which may be caused by true SVs not easily detectable with short reads. The quality of short-read assemblies varies in terms of contiguity, misassembly rate and redundancy between contigs, but overall, they are largely comparable to each other.

3.2 Performance on SNP and INDEL calling

One of the key motivations of fermi is to explore the power of *de novo* assembly in calling short variants. We collected several SNP and INDEL call sets (Table 3) and compared the performance of fermi (Tables 4 and 5).

Table 2. Statistics on human whole-genome assemblies

	ABySS	AllPaths-LG	Fermi	SGA	HuRef
Aligned contig bp	2.73 G	2.62 G	2.82 G	2.74 G	2.88 G
Aligned N50	9.0 k	22.6 k	15.6 k	9.8 k	81.4 k
Covered ref. bp	2.69 G	2.59 G	2.74 G	2.70 G	2.78 G
No. of type-1 breaks	5856	13 738	5704	6049	16 318
No. of type-2 breaks	1617	3823	1120	1735	6626

Contigs over 150 bp in length are aligned to the human reference genome GRCh37 with BWA-SW using option '-b33 -q50 -r17'. A type-1 break point is detected if a contig is split during alignment and mapped to two distict locations, and at each location the alignment is longer than 500 bp and the mapping quality is no less than 10. Type-2 break points exclude type-1 break points which can be patched with gaps no longer than 500 bp.

Table 3. Evaluated SNP and INDEL call sets

Label	Data	Assembler	Mapper	Caller
AC	96X Illumina PE[a]	AllPaths-LG	BWA-SW[b]	SAMtools[c]
BS	70X Illumina PE		BWA[d]	SAMtools
CG	Complete Genom.		cgatools2[e]	cgatools2
CV	26X Illumina SE[f]	Cortex		Cortex-var
FC	35X Illumina SE[f]	Fermi	BWA-SW[b]	SAMtools[c]
MD	60X multiple		MAQ	1000 g pilot[g]
MI	Capillary reads[h]			
SS	35X Illumina SE[f]		BWA-SW	SAMtools

[a] AS uses reads from Illumina jumping and fosmid libraries.

[b] BWA-SW is invoked with 'bwa bwasw -b9 -q16 -r1 -w500'.

[c] INDELs are called from pileup without using the SAMtools caller.

[d] Realigned by GATK (Depristo *et al.*, 2011) also around known INDELs.

[e] By Complete Genomics (Drmanac *et al.*, 2010); only 'VQHIGH' calls retained.

[f] CV, FC and SS do not use the pairing information in calling.

[g] 1000 Genomes Project pilot calls; generated from Dindel and multiple SNP callers.

[h] INDEL calls by Mills *et al.* (2011).

Table 4. Statistics of SNP call sets

	FC	CV	SS	BS	CG	MD
No. of SNPs (M)	3.37	2.20	3.24	3.50	3.34	2.69
No. of hets (M)	1.97	1.04	1.94	2.11	2.04	1.65
Ts/tv	2.04	2.03	2.08	2.11	2.12	2.06
DN50 (bp)	3593	6662	3523	3392	3447	3992
DN2/DN50	22.3	20.8	23.4	22.7	22.3	22.9

Ts/tv is the transition-to-transversion ratio of SNPs. DN50 is calculated as follows. The reference genome is masked according to the align-ability mask (http://bit.ly/snpable) and segmented into intervals at heterozygous SNPs. DN50 is computed such as 50% of unique positions in the genome are in intervals longer than DN50. DN2 is calculated similarly and D2/DN50 is the ratio of DN2 and DN50. DN50 measures the sensitivity; the smaller the better. DN2/DN50 measures the precision of heterozygous SNPs; the higher the better.

For SNP calling (Table 4), fermi misses 3% of SNPs called in SS, but finds more additional ones. Manual examination reveals that the additional calls are mainly caused by two factors. Firstly, in the single-end mode, BWA-SW is very conservative. It may consistently give a correct alignment a low-mapping quality which are all downweighted by samtools. Fermi is able to assemble such reads into longer sequences which increase the power of BWA-SW. Secondly, in the fermi alignment, some regions may be mapped with a high-mismatching rate. These may be due to small-scale misassemblies in fermi unitigs or in the reference assembly, or copy-number variations. It is possible that these clustered SNPs contain more errors. Such errors may lead to reduced ts/tv, but tend not to break long homozygous blocks due to very recent coalescences. That is why FC has a good DN2/DN50 ratio, which measures how often false heterozygotes arise from a long homozygous block.

Table 5. Fraction of INDELs found in other call sets

	MD	CG	BS	CV	FC	MI	ALL
MD	240 424	0.819	0.937	0.678	0.947	0.054	0.977
CG	0.752	264 696	0.915	0.629	0.924	0.052	0.965
BS	0.564	0.597	404 646	0.498	0.844	0.044	0.906
CV	0.708	0.726	0.882	251 769	0.902	0.052	0.923
FC	0.588	0.624	0.873	0.522	393 841	0.045	0.952
MI	0.593	0.618	0.790	0.527	0.804	23 216	0.864

INDELs that start within a homopolymer run longer than 6 bp are excluded in all call sets. An INDEL in call set R (indexed by row) is said to be *found* in call set C (indexed by column) if there exists an INDEL in C such that the left-aligned starting positions of the two INDELs are within 20 bp from each other. An INDEL in R is considered to be found in 'ALL' if it is found in one of the other INDEL sets in the table, plus the AC call set. In the table, a number on the diagonal equals $|R|$, the number of INDEL calls in the call set. The fraction equals $|\{g \in R : g \text{ is found in } C\}|/|R|$.

Table 5 shows the comparison between different INDEL call sets. We excluded INDELs around long homopolymer runs in all call sets because INDEL sequencing errors tend to occur around long homopolymer runs and their error profile is still unclear (the 1000 Genomes Project Analysis group, personal communication). In addition, we have excluded the SS INDEL call set which is nearly contained in BS due to the use of the same INDEL caller.

For the call sets in Table 5, MD and CG are relatively small due to the use of very short reads. CV uses 26X 100 bp reads. It is a small call set due to the high-false negative rate of the calling method (Iqbal *et al.*, 2012). The fermi call set FC is slightly smaller than BS, but it has larger overlap with other call sets than BS, and more FC calls are confirmed by others. One explanation to the lower overlapping ratio between BS and ALL is that BS is the only call set that uses 101 bp paired-end information, which gives it higher power for INDELs not detectable with single-end or very short reads. Nonetheless, purely based on Table 5, fermi appears to have higher overall accuracy.

Even with all short-read call sets combined, as many as 14% of double-hit INDELs called by Mills *et al.* (2011) are missed. We manually checked 30 missing INDELs in an alignment viewer. For half of the cases, the short-read alignment and fermi alignment strongly suggest no variations, and for all these cases, the HuRef sequences are identical to GRCh37. In addition, there are a few cases called from regions under clear copy-number changes. In all, we believe INDELs called by Mills *et al.* (2011) only may have high-error rate. With short reads, we can recover most of short INDELs found by capillary sequencing.

4 DISCUSSIONS

In this article, we derived FMD-index by storing both forward and reverse complement DNA sequences in FM-index. This simple modification enables faster forward–backward search than bi-directional BWT (Lam *et al.*, 2009) and makes FMD-index a more natural representation of DNA sequences. Based on FMD-index, we developed a new *de novo* assembler, fermi, which achieves similar quality to other mainstream assemblers.

We demonstrated that it is possible to call SNPs and short INDELs by aligning assembled unitigs to the reference genome. This approach has similar SNP accuracy to the standard mapping-based SNP calling and arguably outperforms the existing methods on INDEL calling in terms of both sensitivity and precision. Assembly based variant calling is a practical and beneficial complement to mapping-based calling.

In the course of evaluating INDEL accuracy, we found that outside long homopolymer regions, INDEL call sets do not often contain false positives, but they may have high-false negative rate, which leads to the apparent small overlap between call sets (Lam *et al.*, 2012).

As a theoretical remark, we note that with read counts kept, unitigs are a *lossless* but *reduced* representation of sequence reads. They are 'reduced' in that individual reads are lost; they are 'lossless' in that all the information in reads, such as small variants, copy numbers and structural changes are fully preserved in unitigs, as long as they are constructed correctly. For single-end reads, it is theoretically possible to 'compress' reads to unitigs, which are largely non-redundant and much smaller in size. Accurately and efficiently constructing unitigs might provide an interesting alternative to data storage and downstream analyses in future, though practical challenges, such as the high-computational cost and the lack of accuracy of unitigs, remain at present.

ACKNOWLEDGEMENTS

We are grateful to David Reich, Evan Eichler and Peter Sudmant for providing additional data and computing resources for evaluating fermi, to David Altshuler and the GSA group at Broad for the helpful discussions and to Richard Durbin and Jared Simpson for their comments on the initial draft of the manuscript. We also thank Shaun Jackman for providing the ABySS assembly and Michael Brudno for his insight into the algorithm time complexity.

Funding: NIH Joint SNP and CNV calling in 1000 Genomes sequence data grant (1U01HG005208-01).

Conflict of Interest: none declared.

REFERENCES

Albers,C.A. *et al.* (2010) Dindel: accurate indel calls from short-read data. *Genome Res.*, **21**, 961–973.

Burrows,M. and Wheeler,D.J. (1994) A block-sorting lossless data compression algorithm. *Technical Report* 124, Digital Equipment Corporation, Palo Alto, CA.

Carnevali,P. *et al.* (2011) Computational techniques for human genome resequencing using mated gapped reads. *J. Comput. Biol.*, **19**, 279–292.

Chaisson,M.J. *et al.* (2009) De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Res.*, **19**, 336–346.

Depristo,M.A. *et al.* (2011) A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat. Genet.*, **43**, 491–498.

Drmanac,R. *et al.* (2010) Human genome sequencing using unchained base reads on self-assembling DNA nanoarrays. *Science*, **327**, 78–81.

Elias,P. (1975) Universal codeword sets and representations of the integers. *IEEE Trans. Inf. Theory*, **21**, 194–203.

Ferragina,P. and Manzini,G. (2000) Opportunistic data structures with applications. In *FOCS*, Redondo Beach, California, USA. IEEE Computer Society, pp. 390–398.

Ferragina,P. *et al.* (2010) Lightweight data indexing and compression in external memory. In López-Ortiz, A. (ed.), *LATIN*, Oaxaca, Mexico; volume 6034 of *Lecture Notes in Computer Science*, Springer, pp. 697–710.

Gingeras, T. R. *et al.* (1979) Computer programs for the assembly of DNA sequences. *Nucleic Acids Res.*, **7**, 529–545.

Gnerre,S. *et al.* (2011) High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc. Natl. Acad. Sci. USA*, **108**, 1513–1518.

Homer,N. and Nelson,S.F. (2010) Improved variant discovery through local re-alignment of short-read next-generation sequencing data using SRMA. *Genome Biol.*, **11**, R99.

Hon,W.-K. *et al.* (2007) A space and time efficient algorithm for constructing compressed suffix arrays. *Algorithmica*, **48**, 23–36.

Idury,R.M. and Waterman,M.S. (1995) A new algorithm for DNA sequence assembly. *J. Comput. Biol.*, **2**, 291–306.

Ilie,L. *et al.* (2011) HiTEC: accurate error correction in high-throughput sequencing data. *Bioinformatics*, **27**, 295–302.

Iqbal,Z. *et al.* (2012) *De novo* assembly and genotyping of variants using colored de bruijn graphs. *Nat. Genet.*, **44**, 226–232.

Lam,T.W. *et al.* (2009) High throughput short read alignment via bi-directional BWT. In *BIBM*, Washington, DC, USA. pp. 31–36.

Lam,H.Y.K. *et al.* (2012) Performance comparison of whole-genome sequencing platforms. *Nat. Biotechnol.*, **30**, 78–82.

Levy,S. *et al.* (2007) The diploid genome sequence of an individual human. *PLoS Biol.*, **5**, e254.

Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

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

Li,H. (2011) Improving SNP discovery by base alignment quality. *Bioinformatics*, **27**, 1157–1158.

Li,R. *et al.* (2010) *De novo* assembly of human genomes with massively parallel short read sequencing. *Genome Res*, **20**, 265–272.

Mäkinen,V. *et al.* (2009) Storage and retrieval of individual genomes. In Batzoglou,S. (ed.), *RECOMB*. Tucson, AZ, USA; volume 5541 of *Lecture Notes in Computer Science*, Springer, pp. 121–137.

Manske,H.M. and Kwiatkowski,D.P. (2009) SNP-o-matic. *Bioinformatics*, **25**, 2434–2435.

Medvedev,P. *et al.* (2007) Computability of models for sequence assembly. In Giancarlo,R. and Hannenhalli,S. (eds.), *WABI*, Philadelphia, PA, USA; volume 4654 of *Lecture Notes in Computer Science*, Springer, pp. 289–301.

Mills,R.E. *et al.* (2011) Natural genetic variation caused by small insertions and deletions in the human genome. *Genome Res.*, **21**, 830–839.

Myers,E.W. (1995) Toward simplifying and accurately formulating fragment assembly. *J. Comput. Biol.*, **2**, 275–290.

Myers,E.W. *et al.* (2000) A whole-genome assembly of drosophila. *Science*, **287**, 2196–2204.

Myers,E.W. (2005) The fragment assembly string graph. *Bioinformatics*, **21** (Suppl. 2), ii79–ii85.

Nong,G. *et al.* (2011) Two efficient algorithms for linear time suffix array construction. *IEEE Trans. Comput.*, **60**, 1471–1484.

Ossowski,S. *et al.* (2008) Sequencing of natural strains of arabidopsis thaliana with short reads. *Genome Res.*, **18**, 2024–2033.

Peltola,H. *et al.* (1984) SEQAID: a DNA sequence assembling program based on a mathematical model. *Nucleic Acids Res.*, **12**, 307–321.

Pevzner,P.A. *et al.* (2001) An eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci. USA*, **98**, 9748–9753.

Simpson,J.T. and Durbin,R. (2010) Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, **26**, i367–i373.

Simpson,J.T. and Durbin,R. (2012) Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.*, **22**, 549–556.

Simpson,J.T. *et al.* (2009) ABySS: a parallel assembler for short read sequence data. *Genome Res.*, **19**, 1117–1123.

Siren,J. (2009) Compressed suffix arrays for massive data. In *String Processing and Information Retrieval*, Saariselkä, Finland, pp. 63–74.

Staden,R. (1979) A strategy of DNA sequencing employing computer programs. *Nucleic Acids Res.*, **6**, 2601–2610.

Zerbino,D.R. *et al.* (2009) Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. *PLoS ONE*, **4**, e8407.

1000 Genomes Project Consortium. (2010) A map of human genome variation from population-scale sequencing. *Nature*, **467**, 1061–1073.