Fast, Parallel INDEL Discovery With Bubble Flank Motifs

Frank Austin Nothaft
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720, USA
fnothaft@berkeley.edu

October 31, 2016

Abstract

One of the primary challenges in variant calling is the discovery of short insertion/deletion variants. These variants are difficult to call because the reads that contain these variants are frequently misaligned around the variant. To improve variant calling accuracy for short insertions and deletions, several tools have switched to use local assembly based approaches. Local assemblers detect areas of the reference genome that appear to be poorly aligned, and call variants by performing de novo assembly on these regions. While local assembly greatly improves variant calling accuracy, it has a substantial runtime cost. In this paper, we introduce an algorithm that uses insight gained from the structure of a colored de Bruijn graph to realign individual reads. We prove that this algorithm has lower runtime complexity than traditional local assembly approaches, while also providing canonical alignments for an individual sequence variant. We demonstrate this algorithm in the open source, parallel variant caller Avocado, which is available from `https://github.com/bigdatagenomics/avocado`.

1 Introduction

While many modern short-read based variant callers can achieve greater than 99% accuracy when calling single nucleotide variants (SNVs), accuracy decreases when calling insertion/deletion (INDEL) variants. For INDELs that are shorter than the read length, some degradation in accuracy is caused by incorrect mapping of reads containing INDELs. However, this is not the primary culprit. As we show in §**??** with a synthetic experiment, the majority of reads that contain short INDELs map correctly. While these reads map correctly, they are frequently locally misaligned. In short, these reads have been placed in the correct area of the reference genome, but their local alignment makes them appear to represent a different sequence variant then they truly contain.

There are two general approaches that can be used to eliminate the effect of incorrect local alignment on variant calling accuracy: we can either use a local sequence alignment algorithm that is less likely to misrepresent INDEL variants when aligning individual reads, or we can try to improve the alignments of all reads that have mapped around a genomic locus by pooling and jointly realigning these reads. Most tools have focused on the second approach, which further bifurcates into realignment-only and reassembly-with-realignment algorithms. Research has focused on realignment-based approaches because realignment-based approaches obliviate the major problem with traditional local sequence alignment algorithms [20, 22, **?**], which by their probabilistic or dynamic nature cannot be guaranteed to emit consistent alignments for all reads that contain

an INDEL variant [**?**]. Realignment algorithms eliminate this problem by identifying all possible INDELs in the reads at the site, scoring these variants at the site, and realigning the reads to contain a consistent representation of the top scoring variant(s).

Although realignment and reassembly algorithms have high accuracy, they also have high computational cost. The traditional formulation of a realignment algorithm has $\mathcal{O}(n^4)$ runtime complexity, while most local sequence alignment algorithms have runtime complexity of $\mathcal{O}(n^2)$ or lower. Additionally, realignment approaches require all of the reads covering a genomic locus to be materialized which necessitates a shuffle of the input dataset and can have significant memory requirements. While the end-to-end runtime can be decreased by only reassembling a portion of the genome, the performance improvements available through these approaches are bound similarly to Amdahl's law: local assembly is sufficiently expensive that an approach that prefilters 90% of the genome only achieves a 3× improvement in end-to-end runtime [4].

As an alternative to traditional realignment and reassembly approaches, we introduce a local sequence alignment algorithm that is inspired by colored de Bruijn graphs [**?**]. This algorithm has several desirable properties: it has linear runtime complexity and it produces provably canonical pairwise alignments. When used for locally realigning previously mapped reads, we can determine if an individual read is already canonically aligned prior to realigning, which allows us to aggressively limit the number of reads realigned. We have implemented this algorithm in Avocado, a parallel variant caller implemented using Apache Spark [24, 25] and the ADAM library for parallel genomic analysis [14, 16]. Our approach achieves a 10× speedup over conventional INDEL realignment tools when run on a single node while maintaining variant calling accuracy, and can be parallelized across a cluster of computers. Our implementation is released as open source code under an Apache 2 license and is available from `https://github.com/bigdatagenomics/avocado`.

2 Background

The accuracy of insertion and deletion (INDEL) variant discovery has been improved by the development of variant callers that couple local reassembly with haplotype-based statistical models to recover INDELs that were locally misaligned [2]. Now, several prominent variant callers such as the Genome Analysis Toolkit's (GATK) HaplotypeCaller [6], Scalpel [15], and Platypus [18]. Although haplotype-based methods have enabled more accurate INDEL and single nucleotide polymorphism (SNP) calls [3], this accuracy comes at the cost of end-to-end runtime [21]. Several recent projects have been focused on improving reassembly cost either by limiting the percentage of the genome that is reassembled [4] or by improving the performance of algorithms of the core algorithms used in local reassembly [18].

The performance issues seen in haplotype reassembly approaches derives from the high asymptotic complexity of reassembly algorithms. Although specific implementations may vary slightly, a typical local reassembler performs the following steps:

1. A de Bruijn graph is constructed from the reads aligned to a region of the reference genome,
2. All valid paths (*haplotypes*) between the start and end of the graph are enumerated,
3. Each read is realigned to each haplotype, typically using a pair Hidden Markov Model (HMM, see [7]),
4. A statistical model uses the read↔haplotype alignments to choose the haplotype pair that most likely represents the variants hypothesized to exist in the region,

5. The alignments of the reads to the chosen haplotype pair are used to generate statistics that are then used for genotyping.

In this paper, we focus on steps one through three of the local reassembly problem, as there is wide variation in the algorithms used in stages four and five (see §2). Stage one (graph creation) has approximately $\mathcal{O}(rl_r)$ time complexity, and stage two (graph elaboration) has $\mathcal{O}(h \max(l_h))$ time complexity. The asymptotic time cost bound of local reassembly comes from stage three, where cost is $\mathcal{O}(hrl_r \max(l_h))$, where h is the number of haplotypes tested in this region[1], r is the number of reads aligned to this region, l_r is the read length[2], and $\min(l_h)$ is the length of the shortest haplotype that we are evaluating. This complexity comes from realigning r reads to h haplotypes, where realignment has complexity $\mathcal{O}(l_r l_h)$. We ignore the storage complexity of reassembly here, but provide an extended discussion of de Bruijn graph complexity in §2.

In this paper, we introduce the indexed de Bruijn graph and demonstrate how it can be used to reduce the asymptotic complexity of reassembly. An indexed de Bruijn graph is identical to a traditional de Bruijn graph, with one modification: when we create the graph, we annotate each k-mer with the index position of that k-mer in the sequence it was observed in. This simple addition enables the use of the indexed de Bruijn graph for $\Omega(n)$ local sequence alignment with canonical edit representations for most edits. This structure can be used for both sequence alignment and assembly, and achieves a more efficient approach for variant discovery via local reassembly.

Current variant calling pipelines depend heavily on realignment based approaches for accurate genotyping [12]. Although there are several approaches that do not make explicit use of reassembly, all realignment based variant callers use an algorithmic structure similar to the one described in §1. In non-assembly approaches like FreeBayes [9], stages one and two are replaced with a single step where the variants observed in the reads aligned to a given haplotyping region are filtered for quality and integrated directly into the reference haplotype in that region. In both approaches, local alignment errors (errors in alignment *within* this region) are corrected by using a statistical model to identify the most likely location that the read could have come from, given the other reads seen in this area.

Although the model used for choosing the best haplotype pair to finalize realignments to varies between methods (e.g., the GATK's IndelRealigner uses a simple log-odds model [6], while methods like FreeBayes [9] and Platypus [18] make use of richer Bayesian models), these methods require an all-pairs alignment of reads to candidate haplotypes. This leads to the runtime complexity bound of $\mathcal{O}(hrl_r \min(l_h))$ given in §1, as we must realign r reads to h haplotypes, where the cost of realigning one read to one haplotype is $\mathcal{O}(l_r \max(l_h))$, where l_r is the read length (assumed to be constant for Illumina sequencing data) and $\max(l_h)$ is the length of the longest haplotype. Typically, the data structures used for realignment ($\mathcal{O}(l_r \max(l_h))$ storage cost) can be reused. These methods typically retain *only* the best local realignment per read per haplotype, thus bounding storage cost at $\mathcal{O}(hr)$.

For non-reassembly based approaches, the cost of generating candidate haplotypes is $\mathcal{O}(r)$, as each read must be scanned for variants, using the pre-existing alignment. These variants are typically extracted from the CIGAR string, but may need to be normalized [12]. de Bruijn graph based reassembly methods have similar $\mathcal{O}(r)$ time complexity for building the de Bruijn graph as each read must be sequentially broken into k-mers, but these methods have a different storage cost. Specifically, storage cost for a de Bruijn graph is similar to $\mathcal{O}(k(l_{\text{ref}} + l_{\text{variants}} + l_{\text{errors}}))$, where l_{ref}

[1]The number of haplotypes tested may be lower than the number of haplotypes reassembled. Several tools (see [6, 9]) allow users to limit the number of haplotypes evaluated to improve performance.

[2]For simplicity, we assume constant read length. This is a reasonable assumption as many of the variant callers discussed target Illumina reads that have constant length.

is the length of the reference haplotype in this region, l_{variants} is the length of true variant sequence in this region, l_{errors} is the length of erroneous sequence in this region, and k is the k-mer size. In practice, we can approximate both errors and variants as being random, which gives $\mathcal{O}(kl_{\text{ref}})$ storage complexity. From this graph, we must enumerate the haplotypes present in the graph. Starting from the first k-mer in the reference sequence for this region, we perform a depth-first search to identify all paths to the last k-mer in the reference sequence. Assuming that the graph is acyclic (a common restriction for local assembly, see §**??**), we can bound the best case cost of this search at $\Omega(h \min l_h)$.

The number of haplotypes evaluated, h, is an important contributor to the algorithmic complexity of reassembly pipelines, as it sets the storage and time complexity of the realignment scoring phase, the time complexity of the haplotype enumeration phase, and is related to the storage complexity of the de Bruijn graph. The best study of the complexity of assembly techniques was done by Kingsford et al. [10], but is focused on *de novo* assembly and pays special attention to resolving repeat structure. In the local realignment case, the number of haplotypes identified is determined by the number of putative variants seen. We can naïvely model this cost with (1), where f_v is the frequency with which variants occur, ϵ is the rate at which bases are sequenced erroneously, and c is the coverage (read depth) of the region.

$$h \sim f_v l_{\text{ref}} + \epsilon l_{\text{ref}} c \tag{1}$$

This model is naïve, as the coverage depth and rate of variation varies across sequenced datasets, especially for targeted sequencing runs [8]. Additionally, while the ϵ term models the total number of sequence errors, this is not completely correlated with the number of *unique* sequencing errors, as sequencing errors are correlated with sequence context [6]. Many current tools allow users to limit the total number of evaluated haplotypes, or apply strategies to minimize the number of haplotypes considered, such as filtering observed variants that are likely to be sequencing errors [9], restricting realignment to INDELs (IndelRealigner, [6]), or by trimming paths from the assembly graph. Additionally, in an de Bruijn graph, errors in the first k or last k bases of a read will manifest as spurs (see §**??**) and will not contribute paths through the graph. We provide (1) solely as a motivating approximation, and hope to study these characteristics in more detail in future work.

3 Method

As opposed to traditional realignment based approaches, we canonicalize INDELs in the reads by looking for "bubble flank motifs." In a colored de Bruijn graph, a bubble refers to a location where the graph diverges between two samples. In §3.1, we demonstrate how we can use the reconvergence of the de Bruijn graph in the flanking sequence around a bubble to define provably canonical alignments of the bubble between two sequences. For a colored de Bruijn graph containing reads and the reference genome, this allows us to canonically express INDEL variants in the reads against the reference. In §3.2, we then show how this approach can be implemented efficiently without building a de Bruijn graph per read, or even adding each read to a de Bruijn graph.

3.1 Preliminaries

Our method relies on an *indexed de Bruijn* graph, which is a slight extension of the colored de Bruijn graph [**?**]. Specifically, each k-mer in an indexed de Bruijn graph knows which sequence position (index) it came from in it's underlying read/sequence. To construct an indexed de Bruijn graph, we start with the traditional formulation of a *de Brujin* graph for sequence assembly:

Definition 1 (de Bruijn Graph)**.** *A de Bruijn graph describes the observed transitions between adjacent k-mers in a sequence. Each k-mer s represents a k-length string, with a $k-1$ length prefix given by prefix(s) and a length 1 suffix given by suffix(s). We place a directed edge ($\rightarrow$) from k-mer s_1 to k-mer s_2 if prefix$(s_1)^{\{1,k-2\}}$ + suffix(s_1) = prefix(s_2).*

Now, suppose we have n sequences $\mathcal{S}_1, \ldots, \mathcal{S}_n$. Let us assert that for each k-mer $s \in \mathcal{S}_i$, then the output of function $\text{index}_i(s)$ is defined. This function provides us with the integer position of s in sequence $\mathcal{S}_i$. Further, given two k-mers $s_1, s_2 \in \mathcal{S}_i$, we can define a distance function $\text{distance}_i(s_1, s_2) = |\text{index}_i(s_1) - \text{index}_i(s_2)|$. To create an indexed de Bruijn graph, we simply annotate each k-mer s with the $\text{index}_i(s)$ value for all $\mathcal{S}_i, i \in \{1, \ldots, n\}$ where $s \in \mathcal{S}_i$. This index value is trivial to log when creating the original de Bruijn graph from the provided sequences.

Let us require that all sequences $\mathcal{S}_1, \ldots, \mathcal{S}_n$ are not repetitive, which implies that the resulting de Bruijn graph is acyclic. If we select any two sequences $\mathcal{S}_i$ and $\mathcal{S}_j$ from $\mathcal{S}_1, \ldots, \mathcal{S}_n$ that share at least two k-mers s_1 and s_2 with common ordering ($s_1 \rightarrow \cdots \rightarrow s_2$ in both $\mathcal{S}_i$ and $\mathcal{S}_j$), the indexed de Bruijn graph G provides several guarantees:

1. If two sequences $\mathcal{S}_i$ and $\mathcal{S}_j$ share at least two k-mers s_1 and s_2, we can provably find the maximum edit distance d of the subsequences in $\mathcal{S}_i$ and $\mathcal{S}_j$, and bound the cost of finding this edit distance at $\mathcal{O}(nd)$,[3]
2. For many of the above subsequence pairs, we can bound the cost at $\mathcal{O}(n)$, *and* provide canonical representations for the necessary edits,
3. $\mathcal{O}(n^2)$ complexity is restricted to aligning the subsequences of $\mathcal{S}_i$ and $\mathcal{S}_j$ that exist *before* s_1 or *after* s_2.

Let us focus on cases 1 and 2, where we are looking at the subsequences of $\mathcal{S}_i$ and $\mathcal{S}_j$ that are between s_1 and s_2. A trivial case arises when both $\mathcal{S}_i$ and $\mathcal{S}_j$ contain an identical path between s_1 and s_2 (i.e., $s_1 \rightarrow s_n \rightarrow \cdots \rightarrow s_{n+m} \rightarrow s_2$ and $s_{n+k} \in \mathcal{S}_i \wedge s_{n+k} \in \mathcal{S}_j \forall k \in \{0, \ldots, m\}$). Here, the subsequences are clearly identical. This determination can be made trivially by walking from vertex s_1 to vertex s_2 with $\mathcal{O}(m)$ cost.

However, three distinct cases can arise whenever $\mathcal{S}_i$ and $\mathcal{S}_j$ diverge between s_1 and s_2. For simplicity, let us assume that both paths are independent (see Definition 2). These three cases correspond to there being either a canonical substitution edit, a canonical INDEL edit, or a non-canonical (but known distance) edit between $\mathcal{S}_i$ and $\mathcal{S}_j$.

Definition 2 (Path Independence)**.** *Given a non-repetitive de Bruijn graph G constructed from $\mathcal{S}_i$ and $\mathcal{S}_j$, we say that G contains independent paths between s_1 and s_2 if we can construct two subsets $\mathcal{S}'_i \subset \mathcal{S}_i, \mathcal{S}'_j \subset \mathcal{S}_j$ of k-mers where $s_{i+n} \in \mathcal{S}'_i \forall n \in \{0, \ldots, m_i\}, s_{i+n-1} \rightarrow s_{i+n} \forall n \in \{1, \ldots, m_i\}$, $s_{j+n} \in \mathcal{S}'_j \forall n \in \{0, \ldots, m_j\}, s_{j+n-1} \rightarrow s_{j+n} \forall n \in \{1, \ldots, m_j\}$, and $s_1 \rightarrow s_i, s_j; s_{i+m_i}, s_{j+m_j} \rightarrow s_2$ and $\mathcal{S}'_i \bigcap \mathcal{S}'_j = \emptyset$, where $m_i = \text{distance}_{\mathcal{S}_i}(s_1, s_2)$, and $m_j = \text{distance}_{\mathcal{S}_j}(s_1, s_2)$. This implies that the sequences $\mathcal{S}_i$ and $\mathcal{S}_j$ are different between s_1, s_2,*

We have a canonical substitution edit if $m_i = m_j = k$, where k is the k-mer size. Here, we can prove that the edit between $\mathcal{S}_i$ and $\mathcal{S}_j$ between s_1, s_2 is a single base substitution k letters after $\text{index}(s_1)$:

Proof regarding Canonical Substitution. Suppose we have two non-repetitive sequences, $\mathcal{S}'_i$ and $\mathcal{S}'_j$, each of length $2k+1$. Let us construct a de Bruijn graph G, with k-mer length k. If each sequence

[3]Here, $n = \max(\text{distance}_{\mathcal{S}_i}(s_1, s_2), \text{distance}_{\mathcal{S}_j}(s_1, s_2))$.

begins with k-mer s_1 and ends with k-mer s_2, then that implies that the first and last k letters of $\mathcal{S}'_i$ and $\mathcal{S}'_j$ are identical. If both subsequences had the same character at position k, this would imply that both sequences were identical and therefore the two paths between s_1, s_2 would not be independent (Definition 2). If the two letters are different *and* the subsequences are non-repetitive, each character is responsible for k previously unseen k-mers. This is the only possible explanation for the two independent k length paths between s_1 and s_2. □

To visualize the graph corresponding to a substitution, take the two example sequences `CCACTGT` and `CCAATGT`. These two sequences differ by a `C` ↔ `A` edit at position three. With k-mer length $k = 3$, this corresponds to the graph in Figure 1.

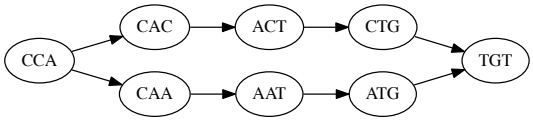

Figure 1: Subgraph Corresponding To a Single Nucleotide Edit

If $m_i = k - 1, m_j \geq k$ or vice versa, we have a canonical INDEL edit (for convenience, we assume that $\mathcal{S}'_i$ contains the $k-1$ length path). Here, we can prove that there is a $m_j - m_i$ length insertion[4] in $\mathcal{S}'_j$ relative to $\mathcal{S}'_i$, $k-1$ letters *after* index(s_1):

Lemma 1 (Distance between k length subsequences)**.** Indexed de Bruijn *graphs naturally provide a distance metric for k length substrings. Let us construct an indexed de Bruijn graph G with k-mers of length k from a non-repetitive sequence* $\mathcal{S}$*. For any two k-mers* $s_a, s_b \in \mathcal{S}, s_a \neq s_b$*, the* $\text{distance}_{\mathcal{S}}(s_a, s_b)$ *metric is equal to* $l_p + 1$*, where* l_p *is the length of the path (in k-mers) between* s_a *and* s_b*. Thus, k-mers with overlap of* $k-1$ *have an edge directly between each other (*$l_p = 0$*) and a distance metric of 1. Conversely, two k-mers that are adjacent but not overlapping in* $\mathcal{S}$ *have a distance metric of k, which implies* $l_p = k - 1$*.*

Proof regarding Canonical INDELs. We are given a graph G which is constructed from two non-repetitive sequences $\mathcal{S}'_i$ and $\mathcal{S}'_j$, where the only two k-mers in both $\mathcal{S}'_i$ and $\mathcal{S}'_j$ are s_1 and s_2 and both sequences provide independent paths between s_1 and s_2. By Lemma 1, if the path from $s_1 \to \cdots \to s_2 \in \mathcal{S}'_i$ has length $k-1$, then $\mathcal{S}'_i$ is a string of length $2k$ that is formed by concatenating s_1, s_2. Now, let us suppose that the path from $s_1 \to \cdots \to s_2 \in \mathcal{S}'_j$ has length $k + l - 1$. The first l k-mers after s_1 will introduce a l length subsequence $\mathcal{L} \subset \mathcal{S}'_j, \mathcal{L} \not\subset \mathcal{S}'_i$, and then the remaining $k-1$ k-mers in the path provide a transition from $\mathcal{L}$ to s_2. Therefore, $\mathcal{S}'_j$ has length of $2k + l$, and is constructed by concatenating $s_1, \mathcal{L}, s_2$. This provides a canonical placement for the inserted sequence $\mathcal{L}$ in $\mathcal{S}'_j$ between s_1 and s_2. □

To visualize the graph corresponding to a canonical INDEL, take the two example sequences `CACTGT` and `CACCATGT`. Here, we have a `CA` insertion after position two. With k-mer length $k = 3$, this corresponds to the graph in Figure 2.

Where we have a canonical allele, the cost of computing the edit is set by the need to walk the graph linearly from s_1 to s_2, and is therefore $\mathcal{O}(n)$. However, in practice, we will see differences that cannot be described as one of the earlier two canonical approaches. First, let us generalize from the two above proofs: if we have two independent paths between s_1, s_2 in the de Bruijn graph

[4]This is equivalently a $m_j - m_i$ length deletion in $\mathcal{S}'_i$ relative to $\mathcal{S}'_j$.

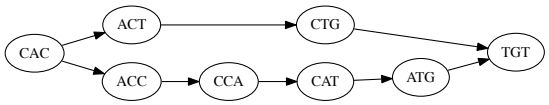

Figure 2: Subgraph Corresponding To a Canonical INDEL Edit

G that was constructed from $\mathcal{S}_i, \mathcal{S}_j$, we can describe $\mathcal{S}_i$ as a sequence created by concatenating $s_1, \mathcal{L}_i, s_2$.[5] The canonical edits merely result from special cases:

- In a canonical substitution edit, $l_{\mathcal{L}_i} = l_{\mathcal{L}_j} = 1$.
- In a canonical INDEL edit, $l_{\mathcal{L}_i} = 0, l_{\mathcal{L}_j} \geq 1$.

Conceptually, a non-canonical edit occurs when two edits occur within k positions of each other. In this case, we can trivially fall back on a $O(nm)$ local alignment algorithm (e.g., a pairwise HMM or Smith-Waterman, see [7, 20]), *but* we only need to locally realign $\mathcal{L}_i$ against $\mathcal{L}_j$, which reduces the size of the realignment problem. However, we can further limit this bound by limiting the maximum number of INDEL edits to $d = |l_{\mathcal{L}_i} - l_{\mathcal{L}_j}|$. This allows us to use an alignment algorithm that limits the number of INDEL edits (e.g., Ukkonen's algorithm [22]). By this, we can achieve $O(n(d+1))$ cost.

3.2 Implementation

4 Results

4.1 Accuracy

To benchmark Avocado's accuracy, we used the high coverage, PCR-free whole genome sequencing (WGS) run of NA12878 from the 1,000 Genomes project [1]. We chose this dataset because NA12878 has extensive orthogonal verification data that is available through the National Institute for Standards and Time's (NIST's) Genome-in-a-Bottle (GIAB) project [27], and the WGS preparation of NA12878 for the 1,000 Genomes project[6] is more representative of a typical sequencing dataset than the GIAB 300× coverage WGS data for NA12878. We verified our calls using the Global Alliance for Genomics and Health's (GA4GH's) benchmarking suite[7] Since the NA12878 alignment available from 1,000 Genomes represents the final, preprocessed analysis-ready reads, we realigned the reads using BWA, before running the data through Avocado, Samtools/Bcftools Mpileup [5, 11, 13], and the GATK's HaplotypeCaller [6]. The relative accuracy of all three tools is shown in Table 1.

4.2 Performance

Figure 3A demonstrates Avocado's strong scaling. In this experiment, we ran Avocado's INDEL realigner and genotyper on a constant dataset (the high coverage NA12878 dataset from 1,000 Genomes [1]), while varying the size of the cluster that Avocado was run on. This experiment was

[5] This property holds true for $\mathcal{S}_j$ as well.

[6] The high coverage NA12878 data is available from `ftp://ftp-trace.ncbi.nlm.nih.gov/1000genomes/ftp/phase3/data/NA12878/high_coverage_alignment/NA12878.mapped.ILLUMINA.bwa.CEU.high_coverage_pcr_free.20130906.bam`.

[7] Available from `https://github.com/ga4gh/benchmarking-tools`. Also, see Paten et al [17].

Table 1: Accuracy on NA12878

	Mpileup	GATK	Avocado
SNP Recall			97%
SNP Precision			98%
INDEL Recall			55%
INDEL Precision			75%
Runtime			19h55m

run on our in-house cluster, which contains 64 machines connected by a full-bisection bandwidth 10 gigabit ethernet network. Each machine in this cluster runs a 16-core Intel Xeon E5-2670 at 2.6GHz, with 256GB of memory. Each node has four 1TB hard disk drives, and data was stored in the Apache Hadoop Distributed File System (HDFS, [19]). The cluster resources are managed by Apache YARN [23]. We ran Apache Hadoop 2.6.2, and Apache Spark 1.6.2. Due to cluster maintainance, several nodes were out of commission, limiting us to 896 cores for our experiments.

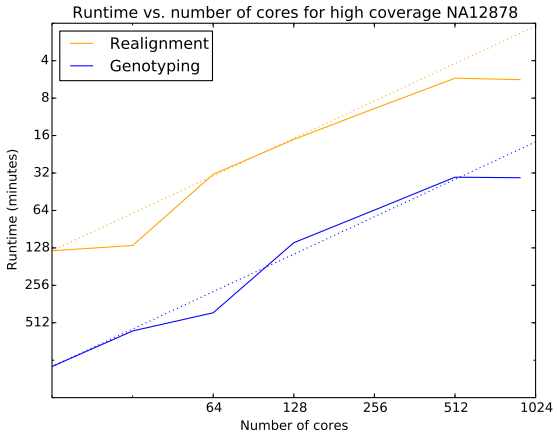

Figure 3: **A, left:** Runtime of the INDEL realignment and genotyping algorithms as the number of cores is changed. Note that the times on the Y-axis are descending; as the number of cores made available to Avocado is doubled, runtime decreases. The dotted lines represent ideal scaling, where runtime halves when the number of cores in the cluster is doubled. **B, right:** Job completion times for the various stages of the Avocado pipeline.

Avocado demonstrates linear strong scaling out to the full size of our cluster. This is due to the even distribution of work across all of the nodes in our cluster. This can be seen in Figure 3B, which shows that task completion times are fairly uniform within each stage of the realignment and genotyping process.

5 Conclusion

In this paper, we have introduced the indexed de Bruijn graph. This extension of the traditional de Bruijn graph allows for $\Omega(n)$ local alignment of two or more sequences, with a hard $\mathcal{O}(n)$ bound for canonical edits and an $\mathcal{O}(l^2)$ bound for non-canonical edits (in genotyping, l is the variant

allele length, which is typically much smaller than n). After describing this structure, we have demonstrated how indexed de Bruijn graphs can be used with a pooled model to locally reassemble haplotypes that contain variants. By using an indexed de Bruijn graph, we are able to reduce the runtime complexity of the local reassembly algorithms that are commonly used for variant discovery and calling.

References

[1] 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature 526*, 7571 (2015), 68–74.

[2] Albers, C. A., Lunter, G., MacArthur, D. G., McVean, G., Ouwehand, W. H., and Durbin, R. Dindel: accurate indel calls from short-read data. *Genome research 21*, 6 (2011), 961–973.

[3] Bao, R., Huang, L., Andrade, J., Tan, W., Kibbe, W. A., Jiang, H., and Feng, G. Review of current methods, applications, and data management for the bioinformatics analysis of whole exome sequencing. *Cancer informatics 13*, Suppl 2 (2014), 67.

[4] Bloniarz, A., Talwalkar, A., Terhorst, J., Jordan, M. I., Patterson, D., Yu, B., and Song, Y. S. Changepoint analysis for efficient variant calling. In *Research in Computational Molecular Biology* (2014), Springer, pp. 20–34.

[5] Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., et al. The variant call format and VCFtools. *Bioinformatics 27*, 15 (2011), 2156–2158.

[6] DePristo, M. A., Banks, E., Poplin, R., Garimella, K. V., Maguire, J. R., Hartl, C., Philippakis, A. A., del Angel, G., Rivas, M. A., Hanna, M., et al. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics 43*, 5 (2011), 491–498.

[7] Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge Univ Press, 1998.

[8] Fang, H., Wu, Y., Narzisi, G., O'Rawe, J. A., Barrón, L. T. J., Rosenbaum, J., Ronemus, M., Iossifov, I., Schatz, M. C., and Lyon, G. J. Reducing INDEL calling errors in whole genome and exome sequencing data. *Genome Med 6* (2014), 89.

[9] Garrison, E., and Marth, G. Haplotype-based variant detection from short-read sequencing. *arXiv preprint arXiv:1207.3907* (2012).

[10] Kingsford, C., Schatz, M. C., and Pop, M. Assembly complexity of prokaryotic genomes using short reads. *BMC bioinformatics 11*, 1 (2010), 21.

[11] Li, H. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics 27*, 21 (2011), 2987–2993.

[12] Li, H. Towards better understanding of artifacts in variant calling from high-coverage samples. *arXiv preprint arXiv:1404.0929* (2014).

[13] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., et al. The sequence alignment/map format and SAMtools. *Bioinformatics 25*, 16 (2009), 2078–2079.

[14] Massie, M., Nothaft, F., Hartl, C., Kozanitis, C., Schumacher, A., Joseph, A. D., and Patterson, D. A. ADAM: Genomics formats and processing patterns for cloud scale computing. Tech. rep., UCB/EECS-2013-207, EECS Department, University of California, Berkeley, 2013.

[15] Narzisi, G., O'Rawe, J. A., Iossifov, I., Fang, H., Lee, Y.-h., Wang, Z., Wu, Y., Lyon, G. J., Wigler, M., and Schatz, M. C. Accurate de novo and transmitted indel detection in exome-capture data using microassembly. *Nature methods 11*, 10 (2014), 1033–1036.

[16] Nothaft, F. A., Massie, M., Danford, T., Zhang, Z., Laserson, U., Yeksigian, C., Kottalam, J., Ahuja, A., Hammerbacher, J., Linderman, M., Franklin, M., Joseph, A. D., and Patterson, D. A. Rethinking data-intensive science using scalable analytics systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)* (2015), ACM.

[17] Paten, B., Diekhans, M., Druker, B. J., Friend, S., Guinney, J., Gassner, N., Guttman, M., Kent, W. J., Mantey, P., Margolin, A. A., et al. The NIH BD2K center for Big Data in Translational Genomics. *Journal of the American Medical Informatics Association (JAMIA) 22*, 6 (2015), 1143–1147.

[18] Rimmer, A., Phan, H., Mathieson, I., Iqbal, Z., Twigg, S. R., Wilkie, A. O., McVean, G., Lunter, G., WGS500 Consortium, et al. Integrating mapping-, assembly- and haplotype-based approaches for calling variants in clinical sequencing applications. *Nature genetics 46*, 8 (2014), 912–918.

[19] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The Hadoop distributed file system. In *Proceedings of the Symposium on Mass Storage Systems and Technologies (MSST '10)* (2010), IEEE, pp. 1–10.

[20] Smith, T. F., and Waterman, M. S. Identification of common molecular subsequences. *Journal of molecular biology 147*, 1 (1981), 195–197.

[21] Talwalkar, A., Liptrap, J., Newcomb, J., Hartl, C., Terhorst, J., Curtis, K., Bresler, M., Song, Y. S., Jordan, M. I., and Patterson, D. SMASH: A benchmarking toolkit for human genome variant calling. *Bioinformatics* (2014), btu345.

[22] Ukkonen, E. Algorithms for approximate string matching. *Information and control 64*, 1 (1985), 100–118.

[23] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al. Apache Hadoop YARN: Yet another resource negotiator. In *Proceedings of the Symposium on Cloud Computing (SoCC '13)* (2013), ACM, p. 5.

[24] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M., Shenker, S., and Stoica, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI '12)* (2012), USENIX Association, p. 2.

[25] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: cluster computing with working sets. In *Proceedings of the USENIX Conference on Hot Topics in Cloud Computing (HotCloud '10)* (2010), p. 10.

[26] Zerbino, D. R., McEwen, G. K., Margulies, E. H., and Birney, E. Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. *PloS one 4*, 12 (2009), e8407.

[27] Zook, J. M., Chapman, B., Wang, J., Mittelman, D., Hofmann, O., Hide, W., and Salit, M. Integrating human sequence data sets provides a resource of benchmark SNP and INDEL genotype calls. *Nature 201*, 4.