Supplementary Materials for "Leveraging known genomic variants to improve detection of variants, especially close-by Indels"

Nam S. Vo [1] and Vinhthuy Phan[2]
[1]Department of Bioinformatics and Computational Biology, The University of Texas MD Anderson Cancer Center, Houston, TX 77030, USA
[2]Department of Computer Science, The University of Memphis, Memphis, TN 38111, USA

1 Iterated randomized algorithm

1.1 Overview of algorithm

Algorithm 1 describes the iterated randomized algorithm mentioned in main text. In this algorithm we aim at finding proper seeds based on long exact matches of the read r and the meta-genome G *with respect to* p.

Algorithm 1 FindSeeds(read r)

```
1: p <− a random position on the read r.
2: for n from 1 to A do
3:    repeat
4:       Start forward search in r from p using FM-index ℑ of meta-genome 𝒢.
5:    until seed length ≥ W OR search reaches the end |r|
6:    if seed found then
7:       return seeds
8:    else
9:       p <− a random position on the read r.
10: Return ∅.
```

In line 1, we set the maximal numbers of iterations to A. If A is too small, there will be many unaligned reads. If A is too large, the algorithm is slow. So, it is important to choose A appropriately. In line 5 and 6, we set minimum seed length to W. If W is too small, we might run into repeats, which lead to false positives. Small W also leads to more seeds and we have to perform more approximate matching for the next step (extending seeds), which is much more time-consuming. Therefore, the choice of W is very important. Our strategy for determining good values of A and W is based on randomization.

1.2 Parameter estimation

Suppose that the correct seed is s. Assume that there are d mismatches between r and s, which divide r into $d+1$ blocks. Let the sizes of the blocks be $m_1, m_2, \cdots, m_{d+1}$, then the length of r is $m = \sum_{i=1}^{d+1} m_i$. Since p is a random position, the seed found by Algorithm 1, as a common substring, would be a random block. This block is selected with probability $p_i = \frac{m_i}{m}$, thus the expected size of block i is $E[S_i] = m_i p_i = \frac{m_i^2}{m}$. Therefore, the expected size of a random block, i.e. the expected length of the seed, is $E[X] = \sum_{i=1}^{d+1} E[X_i] = \sum_{i=1}^{d+1} \frac{m_i^2}{m}$. Use Cauche-Schwarz inequality we have:

$$\left(\sum_{i=1}^{d+1} \frac{1}{d+1}\frac{m_i}{m}\right)^2 \leq \sum_{i=1}^{d+1}(\frac{1}{d+1})^2 \sum_{i=1}^{d+1}(\frac{m_i}{m})^2$$

After simplifying we have $E[S] \geq \frac{m}{d+1}$. In other words, the expected length of the seeds found by Algorithm 1 is at least $\frac{m}{d+1}$. Therefore, we can estimate $W \sim \frac{m}{d+1}$. The distance d can be estimated by the rate of mutations of the given genome and the rate of sequencing errors of the given data. Let b be the rate of mutations or sequencing errors, which we can assume to be distributed by a binomial distribution with mean $\mu = mb$ and variance $\sigma^2 = mb(1-b)$, where m is read length. Then the upper bound of d can be estimated by $\mu + c\sigma$, for some constant c, then the value of W can be derived. In our experimentation, $c = 4$ produces good performance.

The value of A can be selected so that the longest block can be sampled with high certainty. The probability that the longest block is selected (i.e. if a random position p lands inside it) is $\frac{m^*}{m}$, where m^* is the length of the longest block. Using the Pigeonhole Principle we have $m^* \geq \frac{m}{d+1}$, which means $d + 1 \geq \frac{m}{m^*}$. This is the expected number of iterations to make p land inside the longest block. Therefore, we can estimate $A = t + 1 \geq d + 1$. If we assign $A = c \cdot (t + 1)$, then the probability of landing in the longest block is exponentially increased as a function of c. In our experimentation, $c = 2$ produces good performance, in trade-off between accuracy and running time.

2 Asymmetric alignment algorithm

2.1 Overview of algorithm

The following describes our asymmetric alignment algorithm mentioned in main text. In this algorithm, we aim to compute edit distance with affine gap penalty between $s = s_1 s_2 \cdots s_m$ and $t = t_1 t_2 \cdots t_n$, in which s represents a read and t represents the matching meta-reference:

- Consider the right-most column of alignment between $s = s_1 s_2 \cdots s_i$ $(1 \leq i \leq m)$ and $t = t_1 t_2 \cdots t_j$ $(1 \leq j \leq n)$, we compute three traditional matrixes, D for an alignment (s_i, t_j), IS for an alignment $(s_i, -)$, and IT for an alignment $(-, t_j)$:

  $D(i,j)$: ... s_i / ... t_j $\quad IS(i,j)$: ... s_i / ... $-$ $\quad IT(i,j)$: ... $-$ / ... t_j

- If t_j is a standard character (A, C, G, T, N) then we use the traditional dynamic programming strategy for affine gap alignment.

- If t_j is a variant character V (a known variant location) then t_j can have several possibilities, which can be Indels. The algorithm needs to consider all possibilities of matching s to these possible variants. In the case of Indels (more than one character), we need to look back on the s from s_i to take a equal-length sequence, i.e., the sequence $s_{i-k+1}s_{i-k+2}...s_i$ (k is equal to length of Indel), to match with the Indel.

This algorithm is exploited together with the aforementioned iterated randomized algorithm. After several iterations, if it found only one match with distinct minimum score, it will stop and considers that match as the correct candidate. Our experiment showed that, in many cases, (e.g, on the non-repeat regions of genomes) the number iterations are quite small. In some cases (e.g, on the repeat regions of genomes, which can result in multiple matches with similar score), the number iterations can reach the maximum number of allowed iterations. In this cases, we employed a filter strategy to determine the correct matches, which selects the correct candidate among the matches with lowest scores and proper insert sizes (for paired-end reads).

2.2 Cost scheme

In our experimentation, the cost of substituting a read base s_i and reference base t_j is $-\log f(t_j)$, where $f(t_j)$ is the probability of t_j to be a variant. If t_j is a known variant, $f(t_j)$ is obtained from its probability in the variant profile, which is updated during the alignment of reads. If t_j is not a known variant, $f(t_j)$ is set to a user-defined value, with default value of mismatches, gap opens and gap extensions are 4, 6 and 1, respectively, the same with default values of BWA-MEM.

The alignment is asymmetrical due to the following initial configurations: (1) $IT_{0,j} = 0$, for $j \leq n$: the cost of converting t to an empty string is 0; (2) $IS_{i,0} = O$, for $j \leq n$, O is gap open penalty; (3) otherwise, the values were set to ∞. It is not possible to match reads to nothing from the reference genome.

3 Variant profile update strategy

Initial profiles of known variants are assigned based on the input databases. Probabilities of known variants are assigned based on their allele frequency in the databases. In case of diploid data, they are assigned based on frequency of genotypes.

Initial profiles of unknown variants, which are detected during the alignment process, are assigned experimentally as user-input. For human genomes, default values of prior probability of a new SNP and a new Indel are assigned to 0.001 and 0.00001, respectively.

The variant profile is then updated read-by-read during the alignment process. After all reads are aligned, the final variant profile will be used to call variants based on maximum probabilities of each variants in the updated profiles.

4 Implementation

Our method, IVC, was implemented in the Go programming language, which was designed to have high performance, garbage collection, and primitive support for concurrency. Although Go might not be very common currently and other languages such as C are also good choices for implementing high performance software like this, we think that Go is a better choice in this context. It is easier to write simple, reliable, and efficient software in Go, although the most optimal C code might be faster than those of Go. We also provided several pre-complied binary packages of the tool for several common platforms to help users run the tool without installing Go. These pre-complied binary packages were obtained by using the Go's built-in cross compilation feature.

The source code of IVC can be found at https://github.com/namsyvo/IVC.

5 Data

- Reference genome: the NCBI Genome Reference Consortium, GRCh37 p.13.
  http://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.25/
- Known variant profile: 1000 Genomes Project Data (Phase 1).
  ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase1/analysis_results/integrated_call_sets/
- Known variant profile: dbSNP (build 149).
  ftp://ftp.ncbi.nih.gov/snp/organisms/human_9606_b149_GRCh37p13/VCF/

- Sequencing reads, Illumina HiSeq 2000 paired-end, sample ERR194147 (individual NA12878), read length 100, coverage ∼50x:
  http://www.illumina.com/platinumgenomes
  http://www.ebi.ac.uk/ena/data/view/ERR194147
- GIAB data: The Genome-In-A-Bottle Consortium.
  ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/NA12878_HG001/NISTv2.19
- Simulated reads: generated from DWGSIM.
  https://github.com/nh13/dwgsim
- Simulated genomes: generated from IVC genome simulator.
  https://github.com/namsyvo/varcall-tools/tree/master/ivc-tools/genome-simulator

6 Variant callers, used command and settings

Scripts to run all tools are available publicly in our github repositories:

- For BWA: https://github.com/namsyvo/varcall-tools/tree/master/bwa-tools
- For GATK-UG/HC: https://github.com/namsyvo/varcall-tools/tree/master/gatk-tools
- For SAMtools: https://github.com/namsyvo/varcall-tools/tree/master/sam-tools
- For Atlas2: https://github.com/namsyvo/varcall-tools/tree/master/atlas-tools
- For IVC: https://github.com/namsyvo/varcall-tools/tree/master/ivc-tools

Commands and parameters to run all tools:

- IVC: running with default parameters, version 0.8.1.

  ivc-index -R ref.fasta -V known_variants.vcf -I index_dir

  ivc -R ref.fasta -V known_variants.vcf -I index-dir -O ivc_var_call.vcf -1 read1.fastq -2 read1.fastq

- Other methods:
  - Alignment: running BWA-MEM version 0.7.7 with default parameters, excluding the number of threads for parallel running (32).
    Input FASTA file: $1
    Output SAM files: $2
    * Indexing the reference: bwa index index_dir/$1.fasta
    * Aligning reads to the reference: bwa mem -t 32 index_dir/$1.fasta data_dir/read1.fastq data_dir/read2.fastq >result_dir/$2.sam 2 >result_dir/log.txt
  - Post-alignment processing:
    Input FASTA file (without file extension .fasta): $1
    Input SAM files (without file extension .sam): $2
    genome-tools path: $3

* Converting SAM to BAM: software version is indicated in the script.
  $3/samtools-0.1.19/samtools view -bS -t $1.fasta -@ 32 $2.sam >$2.bam
* Creating reference dictionary: software version is indicated in the script.
  java -jar $3/picard-tools-1.109/picard-tools-1.109/CreateSequenceDictionary.jar R=$1.fasta O=$1.dict
* Creating fasta index file: software version is indicated in the script.
  $3/samtools-0.1.19/samtools faidx $1.fasta
* Sorting the BAM file: software version is indicated in the script.
  java -jar $3/picard-tools-1.109/picard-tools-1.109/SortSam.jar INPUT=$2.bam OUTPUT=$2-sorted.bam SORT-ORDER=coordinate TMP-DIR=/tmp
* Creating BAM index file: software version is indicated in the script.
  $3/samtools-0.1.19/samtools index $2-sorted.bam $2-sorted.bam.bai
  java -jar $3/picard-tools-1.109/picard-tools-1.109/AddOrReplaceReadGroups.jar I=$2-sorted.bam O=$2-sorted-RG.bam SORT-ORDER=coordinate RGID=foo RGLB=bar RGPL=illumina RGPU=run RGSM=ivc-mutant CREATE-INDEX=True

– Indel realignment:

  Input FASTA file: $1

  Input BAM files: $2

  Input VCF files: $3

  Output VCF files: $4

  genome-tools path: $5

  * Creating Indel Realignment intervals: running IndelRealigner with default parameters, excluding the number of threads for parallel running (32). Software version is indicated in the script.
    java -jar $5/GenomeAnalysisTK-3.1-1/GenomeAnalysisTK.jar -l INFO -nt 32 -T RealignerTargetCreator -R $1 -I $2 -o "$4.forIndelRealigner.intervals"
  * Realigning Indels: running with default parameters, software version is indicated in the script.
    java -jar $5/GenomeAnalysisTK-3.1-1/GenomeAnalysisTK.jar -l INFO -T IndelRealigner -R $1 -I $2 -o $4 -targetIntervals "$4.forIndelRealigner.intervals"

– Variant calling:

  Running each tool with the specified parameters as in the following commands, using input/output file notations as in Indel realignment step, software version is indicated in the script.

  * Calling variants with GATK-UnifiedGenotyper:
    java -jar $5/GenomeAnalysisTK-3.1-1/GenomeAnalysisTK.jar -l INFO -nct 32 -T UnifiedGenotyper -R $1 -I $2 –dbsnp $3 -o $4 –output-mode EMIT-VARIANTS-ONLY -glm BOTH -rf BadCigar -stand-call-conf 20.0 -stand-emit-conf 20.0 -dcov 200
  * Calling variants with GATK-HaplotypeCaller:
    java -jar $5/GenomeAnalysisTK-3.1-1/GenomeAnalysisTK.jar -l INFO -nct 32 -T HaplotypeCaller -R $1 -I $2 -o $4 –genotyping-mode DISCOVERY -rf BadCigar -stand-call-conf 20.0 -stand-emit-conf 20.0 -dcov 200
  * Calling variants with SAMtools:

$5/samtools-0.1.19/samtools mpileup -uf $1 $2 | $5/bcftools/bcftools view -vcg -> $4
$5/bcftools/bcftools norm -f $1 -m -any -o st-var-call-norm.vcf -O v $4

* Calling variants with Altas2:
$5/Atlas2_v1.4.3/Atlas-SNP2/Atlas-SNP2.rb -r $1 -i $2 -o $4 -n dwgsim -y 2 –Illumina
$5/Atlas2_v1.4.3/Atlas-Indel2/Atlas-Indel2.rb -r $1 -b $2 -o $4 -s dwgsim -I
* Calling variants with Scalpel:
$5/scalpel-0.5.3/scalpel-discovery –single –ref $1 –bed exome_regions.bed –bam $2 –dir scalpel_results –window 600 –numprocs 32

– Filtering variants:

In addition to parameters specified by the above commands, all variants are filtered by a threshold of quality 20 or more. They are filtered more by setting thresholds for read depth before evaluation.

7 Raw values of precision and recall in Results section

Table below shows raw values of precision and recall on simulated data at coverages from 3x to 50x, for SNP and Indel calling in known and unknown variant locations before and after indel realignment. Tools: IR: GATK IndelRealigner, UG: GATK UnifiedGenotyper, HC: GATK HaplotypeCaller, ST: SAMtools, AT: Atlas2.

		Known variant locations							Unknown variant locations						
		3x	5x	10x	15x	20x	25x	50x	3x	5x	10x	15x	20x	25x	50x
		SNPs													
IVC	Prec	0.9995	0.9996	0.9997	0.9996	0.9996	0.9996	0.9997	0.9963	0.9949	0.9962	0.9975	0.9978	0.9978	0.9976
	Rec	0.9493	0.9932	0.9998	0.9998	0.9998	0.9998	0.9999	0.8155	0.9648	0.9956	0.9985	0.9984	0.9985	0.9987
UG	Prec	0.9998	0.9996	0.9994	0.9993	0.9992	0.9991	0.9990	0.9891	0.9852	0.9859	0.9857	0.9842	0.9833	0.9804
	Rec	0.8147	0.9633	0.9980	0.9987	0.9988	0.9990	0.9991	0.8150	0.9647	0.9981	0.9987	0.9988	0.9990	0.9991
UG-IR	Prec	0.9999	0.9998	0.9997	0.9997	0.9997	0.9996	0.9997	0.9911	0.9892	0.9926	0.9943	0.9942	0.9941	0.9934
	Rec	0.8147	0.9633	0.9980	0.9987	0.9989	0.9990	0.9991	0.8150	0.9647	0.9981	0.9987	0.9988	0.9989	0.9991
HC	Prec	0.9993	0.9993	0.9995	0.9995	0.9995	0.9994	0.9994	0.9919	0.9908	0.9942	0.9969	0.9970	0.9969	0.9967
	Rec	0.7932	0.9542	0.9964	0.9975	0.9977	0.9979	0.9980	0.7938	0.9555	0.9965	0.9975	0.9977	0.9978	0.9979
HC-IR	Prec	0.9993	0.9993	0.9995	0.9995	0.9995	0.9995	0.9994	0.9919	0.9908	0.9942	0.9969	0.9970	0.9969	0.9967
	Rec	0.7932	0.9542	0.9964	0.9975	0.9977	0.9979	0.9980	0.7938	0.9555	0.9965	0.9975	0.9977	0.9978	0.9979
ST	Prec	0.9982	0.9969	0.9943	0.9943	0.9970	0.9989	0.9995	0.9949	0.9933	0.9901	0.9888	0.9908	0.9918	0.9905
	Rec	0.7838	0.9487	0.9907	0.9918	0.9947	0.9968	0.9976	0.7839	0.9495	0.9911	0.9921	0.9949	0.9967	0.9976
ST-IR	Prec	0.9982	0.9970	0.9944	0.9944	0.9971	0.9990	0.9996	0.9953	0.9943	0.9921	0.9915	0.9940	0.9955	0.9960
	Rec	0.7838	0.9487	0.9906	0.9917	0.9947	0.9967	0.9976	0.7839	0.9495	0.9911	0.9921	0.9949	0.9967	0.9976
AT-IR	Prec	1.0000	1.0000	0.9999	0.9997	0.9994	0.9991	0.9978	0.9973	0.9975	0.9962	0.9947	0.9929	0.9898	0.9871
	Rec	0.2852	0.4842	0.4032	0.2794	0.2113	0.1679	0.0926	0.2839	0.4828	0.4029	0.2801	0.2116	0.1675	0.0925
		Indels													
IVC	Prec	0.9820	0.9819	0.9827	0.9827	0.9827	0.9828	0.9820	0.9021	0.8934	0.9078	0.9062	0.9063	0.9072	0.9061
	Rec	0.9352	0.9773	0.9831	0.9831	0.9832	0.9833	0.9841	0.6974	0.8561	0.9064	0.9247	0.9286	0.9312	0.9338
UG	Prec	0.9262	0.9499	0.9615	0.9623	0.9626	0.9623	0.9617	0.8682	0.8964	0.8987	0.8909	0.8867	0.8851	0.8855
	Rec	0.1134	0.3899	0.8130	0.8780	0.8868	0.8876	0.8886	0.1167	0.4047	0.8173	0.8770	0.8840	0.8859	0.8852
UG-IR	Prec	0.9454	0.9602	0.9638	0.9634	0.9634	0.9634	0.9630	0.8804	0.9037	0.9022	0.8987	0.8985	0.8985	0.8955
	Rec	0.1341	0.4334	0.8352	0.8843	0.8895	0.8900	0.8909	0.1372	0.4495	0.8371	0.8831	0.8865	0.8876	0.8865
HC	Prec	0.9654	0.9653	0.9655	0.9658	0.9659	0.9659	0.9658	0.8870	0.8906	0.8992	0.9005	0.9000	0.9002	0.8995
	Rec	0.6859	0.8430	0.8951	0.8972	0.8977	0.8983	0.8984	0.6807	0.8395	0.8926	0.8954	0.8958	0.8964	0.8960
HC-IR	Prec	0.9654	0.9653	0.9655	0.9658	0.9659	0.9659	0.9658	0.8870	0.8906	0.8992	0.9003	0.8996	0.9000	0.8995
	Rec	0.6867	0.8435	0.8951	0.8972	0.8977	0.8983	0.8985	0.6818	0.8393	0.8926	0.8954	0.8958	0.8964	0.8960
ST	Prec	0.9579	0.9602	0.9623	0.9622	0.9616	0.9608	0.9573	0.8614	0.8631	0.8490	0.8307	0.8118	0.7968	0.7226
	Rec	0.5768	0.7164	0.8402	0.8689	0.8759	0.8788	0.8798	0.5710	0.7289	0.8417	0.8719	0.8790	0.8818	0.8835
ST-IR	Prec	0.9585	0.9610	0.9630	0.9635	0.9634	0.9632	0.9604	0.8735	0.8752	0.8772	0.8685	0.8573	0.8488	0.8003
	Rec	0.5945	0.7275	0.8428	0.8714	0.8779	0.8806	0.8810	0.5882	0.7362	0.8442	0.8729	0.8799	0.8822	0.8826
AT-IR	Prec	0.9315	0.9121	0.9297	0.9369	0.9528	0.9621	0.9681	0.8120	0.8440	0.8314	0.8490	0.8780	0.9003	0.9073
	Rec	0.0165	0.0595	0.1995	0.3435	0.5488	0.7268	0.8859	0.0177	0.0736	0.2096	0.3563	0.5434	0.7241	0.8829