# Supplementary Materials for Fast genotyping of known SNPs through approximate $k$-mer matching

Ariya Shajii, Deniz Yorukoglu, Y. William Yu, & Bonnie Berger*

## A Algorithms

Note, we will only write QUERY($D_{\text{ref}}, K$) and QUERY($D_{\text{SNP}}, K$) to represent queries in $D_{\text{ref}}$ and $D_{\text{SNP}}$, respectively; $J_{\text{ref}}$ and $J_{\text{SNP}}$ are implied.

**Input:** $R$
**Output:** $D_{\text{ref}}$
  $D_{\text{ref}} \leftarrow \{\}$;
  **for** $i \in \{1, 2, \ldots, \text{length}(R) - k + 1\}$ **do**
    $D_{\text{ref}}$.append($(R[i : i + k], i)$); {append ($k$-mer, index) tuple}
  sort($D_{\text{ref}}$); {sort by $\xi(k\text{-mer})$}
  uniq($D_{\text{ref}}$); {only keep one instance of each $k$-mer (which to keep is unimportant)}
  **return** $D_{\text{ref}}$;

Algorithm S.1: Generation of reference dictionary $D_{\text{ref}}$ from reference sequence $R$.

**Input:** $D_{\text{ref}}$
**Output:** $J_{\text{ref}}$
  $J_{\text{ref}} \leftarrow [0, 0, \ldots, 0] \in \mathbb{N}^{2^{32}}$; {for simplicity, assume $J_{\text{ref}}$ is 0-indexed}
  $u_{\text{prev}} \leftarrow 0$;
  **for** $i \in \{1, 2, \ldots, \text{length}(D_{\text{ref}})\}$ **do**
    $K, V \leftarrow D_{\text{ref}}[i]$; {$K$ is the 32-mer, $V$ is its index in the reference}
    $u \leftarrow (\xi(K) \gg 32)$;
    **if** $u \neq u_{\text{prev}}$ **then**
      **for** $j \in \{u_{\text{prev}} + 1, u_{\text{prev}} + 2, \ldots, u\}$ **do**
        $J_{\text{ref}}[j] \leftarrow i$;
    $u_{\text{prev}} \leftarrow u$;
  **for** $i \in \{u_{\text{prev}} + 1, u_{\text{prev}} + 2, \ldots, 2^{32} - 1\}$ **do**
    $J_{\text{ref}}[i] \leftarrow \text{length}(D_{\text{ref}}) + 1$;
  **return** $J_{\text{ref}}$;

Algorithm S.2: Generation of secondary hash table $J_{\text{ref}}$ from reference dictionary $D_{\text{ref}}$.

*Corresponding author: bab@mit.edu

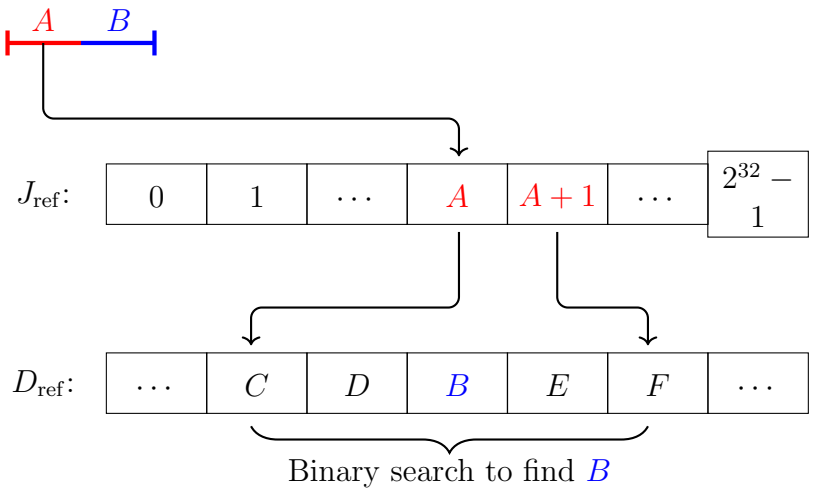


Figure S.1: Simplified visualization of querying $D_{\text{ref}}$ with some 32-mer. The encoded 32-mer has high bits $A$ (red) and low bits $B$ (blue). We look into $J_{\text{ref}}$ at indices $A$ and $A+1$ to obtain the bounds for our search in $D_{\text{ref}}$. Then, since $D_{\text{ref}}$ is sorted by the numerical values of the encoded 32-mers, we perform a binary search on this interval for the 32-mer whose encoding has low bits $B$. Since all 32-mers in the interval have high bits $A$ (by design of $J_{\text{ref}}$), once we find an encoding with low bits $B$, we know we have found our initial 32-mer in the dictionary.

**Input:** $D_{\text{ref}}, J_{\text{ref}}, K$
**Output:** QUERY$(D_{\text{ref}}, J_{\text{ref}}, K)$
  $u \leftarrow \lfloor \xi(K)/2^{32} \rfloor$;
  $a \leftarrow J_{\text{ref}}[u]$;
  **if** $a >$ length$(D_{\text{ref}})$ **then**
    **return** Null;
  **if** $u < 2^{32}$ **then**
    $b \leftarrow J_{\text{ref}}[u+1]$;
  **else**
    $b \leftarrow$ length$(D_{\text{ref}}) + 1$;
  **return** $D_{\text{ref}}[a:b]$.bsearch$(K)$; {binary search from $a$ (inclusive) to $b$ (exclusive)}

Algorithm S.3: Querying of $D_{\text{ref}}$ with some 32-mer $K$.

**Input:** $D_{\text{SNP}}$
**Output:** $P$
  $P \leftarrow \text{Array}(\text{length} = \max\{V.\text{index} : (K, V) \in D_{\text{SNP}}\})$;
  **for** $(K, V) \in D_{\text{ref}}$ **do**
    $p \leftarrow V.\text{index}$;
    $P[p].\text{ref\_allele} \leftarrow V.\text{ref\_allele}$;
    $P[p].\text{alt\_allele} \leftarrow V.\text{alt\_allele}$;
    $P[p].\text{ref\_allele\_freq} \leftarrow V.\text{ref\_allele\_freq}$;
    $P[p].\text{alt\_allele\_freq} \leftarrow V.\text{alt\_allele\_freq}$;
    $P[p].\alpha \leftarrow 0$;
    $P[p].\beta \leftarrow 0$;
  **return** $P$

Algorithm S.4: Initialization of $P$.

**Input:** $D_{\text{ref}}, D_{\text{SNP}}, Q$
**Output:** $\text{TARGET}(D_{\text{ref}}, D_{\text{SNP}}, Q)$
  indices $\leftarrow$ Array();
  kmers $\leftarrow$ Array();
  **for** $(K, \text{offset}) \in \mathcal{S}(Q)$ **do**
    **for** $K' \in \mathcal{N}(K)$ **do**
      $V_1 \leftarrow \text{QUERY}(D_{\text{ref}}, K')$;
      $V_2 \leftarrow \text{QUERY}(D_{\text{SNP}}, K')$;
      **if** $V_1 \neq \text{Null}$ **then**
        indices.append($V_1 - \text{offset}$);
        kmers.append($(K', V_1 - \text{offset})$);
      **if** $V_2 \neq \text{Null}$ **then**
        indices.append($V_2.\text{index} - \text{offset}$);
        kmers.append($(K', V_2.\text{index} - \text{offset})$);
  target $\leftarrow$ highest_multiplicity_element(indices);
  **return** (target, kmers);

Algorithm S.5: Finding target index in reference sequence at which read likely originated. Assume "highest_multiplicity_element" returns the element of highest multiplicity in its argument array if said element is unique and has multiplicity greater than one, and returns Null otherwise. This can, in practice, be implemented in linear runtime using a hash table that maps indices to frequencies.

Note that, in Algorithm S.5, we must always check the following two conditions for each $k$-mer $K' \in \mathcal{N}(K)$ that we query before adding the results of this query to our array of potential target indices:

- Based on the results of the $D_{\text{ref}}$ query, $K'$ must not: differ from $K$ in a position where there exists a SNP *and* have the alternate allele for that SNP.
- Based on the results of the $D_{\text{SNP}}$ query, $K'$ must not: differ from $K$ in a position where there exists a SNP *and* have the reference allele for that SNP.

These two conditions prevent us from changing what is actually the alternate allele of a SNP in a read to the reference allele (via $\mathcal{N}(K)$) and incorrectly obtaining a successful query result in $D_{\text{ref}}$, or vice versa for $D_{\text{SNP}}$.

**Input:** $D_{\text{ref}}, D_{\text{SNP}}, Q, P$
**Output:** –
  target, kmers $\leftarrow$ TARGET$(D_{\text{ref}}, D_{\text{SNP}}, Q)$;
  **for** $(K, \text{normalized\_index}) \in$ kmers **do**
    **if** normalized_index = target **then**
      **for** $i \in \{1, 2, \ldots, k\}$ **do**
        **if** $P[\text{target} + i - 1] \neq \text{Null}$ **then**
          **if** $K[i] = P[\text{target} + i - 1].\text{ref\_allele}$ **then**
            $P[\text{target} + i - 1].\alpha \leftarrow P[\text{target} + i - 1].\alpha + 1$;
          **else if** $K[i] = P[\text{target} + i - 1].\text{alt\_allele}$ **then**
            $P[\text{target} + i - 1].\beta \leftarrow P[\text{target} + i - 1].\beta + 1$;

Algorithm S.6: Updating pileup table for read $Q$.

# B Full Experiment Timings

Table S.1: Full LAVA timing results compared to other genotyping pipelines, corresponding to Fig. 4 in the main text. All times are given in minutes.

| Method | Mapping | Indexing/Sorting | Genotyping | Total Time |
|---|---|---|---|---|
| LAVA (dbSNP) | 0 | 0 | 294.4 | 294.4 |
| LAVA (Affy) | 0 | 0 | 184.8 | 184.8 |
| LAVA Lite (dbSNP) | 0 | 0 | 367.7 | 367.7 |
| LAVA Lite (Affy) | 0 | 0 | 247.0 | 247.0 |
| Bowtie 2 + mpileup (dbSNP) | 781.3 | 68.6 | 446.1 | 1296.0 |
| Bowtie 2 + mpileup (Affy) | 781.3 | 68.6 | 446.1 | 1296.0 |
| BWA + mpileup (dbSNP) | 1193.9 | 68.6 | 437.5 | 1700.0 |
| BWA + mpileup (Affy) | 1193.9 | 68.6 | 437.5 | 1700.0 |
| Bowtie 2 + GATK HC (dbSNP) | 781.3 | 0 | 456.1 | 1237.4 |
| Bowtie 2 + GATK HC (Affy) | 781.3 | 0 | 211.8 | 993.1 |
| BWA + GATK HC (dbSNP) | 1193.9 | 0 | 585.7 | 1779.6 |
| BWA + GATK HC (Affy) | 1193.9 | 0 | 223.9 | 1417.8 |
| SNAP + GATK HC (dbSNP) | 129.2 | 43.5 | 816.4 | 989.1 |
| SNAP + GATK HC (Affy) | 129.2 | 43.5 | 227.4 | 400.1 |