

A natural encoding of genetic variation in a Burrows-Wheeler Transform to enable mapping and genome inference

Sorina Maciuca, Carlos delOjo Elias, Gil McVean, and Zamin Iqbal

Wellcome Trust Centre for Human Genetics,
University of Oxford, Roosevelt Drive, Oxford OX3 7BN, UK
{sorina.maciuca,gil.mcvean,zamin.iqbal}@well.ox.ac.uk
{carlos.delajoelias}@ndm.ox.ac.uk

Abstract. We show how positional markers can be used to encode genetic variation within a Burrows Wheeler Transform (BWT), and use this to construct a generalisation of the traditional “reference genome”, incorporating known variation within a species. Our goal is to support the inference of the closest mosaic of previously known sequence to the genome(s) under analysis.

Our scheme results in an increased alphabet size, and by using a wavelet tree encoding of the BWT we reduce the performance impact on rank operations. We give a specialised form of the backward search that allows variation-aware exact matching. We implement this, and demonstrate the cost of constructing an index of the whole human genome with 80 million genetic variants is 31GB of RAM, below that of previous implementations. We also show that inferring a closer reference can close large kilobase-scale holes in mapping pileups in *P. falciparum*

Keywords: pan-genome, Burrows-Wheeler Transform, FM index, genome

1 Introduction

Genome sequencing involves breaking DNA into fragments, identifying substrings (called “reads”) and then inferring properties of the genome. Recently, it has become possible to study within-species genetic variation on a large scale [6–8], where the dominant approach is to match substrings to the canonical “reference genome” which is constructed from an arbitrary individual. This problem (“mapping”) has been heavily studied (see [5]) and the Burrows Wheeler Transform [2] underlies the two dominant mappers - bwa [3] and bowtie [4]. Mapping reads to a reference genome is a very effective means of detecting genetic variation involving single character changes (SNPs - single nucleotide polymorphisms). However this method becomes less effective the further the genome differs from the reference. Many important regions of genomes are highly diverse, and so sequence reads from a random genome can very easily fail to map to the reference.

We want to build a representation of the genome of a species which incorporates N genomes and supports the following inference. We take as input, sequence data from a new sample of our species, and an estimate of how many genomes the sample contains and their relative proportions - e.g. a normal human sample would contain 2 genomes in a 1:1 ratio, a bacterial isolate would contain 1 genome, and a malaria sample might contain 3 genomes in the ratio 10:3:1. We would then infer the sequence of the underlying genomes. In this paper we describe a method for encoding genetic variation designed to enable this approach.

Genomes evolve (broadly speaking) by two processes - mutation (changing a single character, or less frequently, inserting or deleting a few) and recombination (either two chromosomes exchange a chunk of DNA, or one chromosome copies a chunk from another). Thus once we have seen many genomes of a given species, a new genome is likely to look like a mosaic of genomes we have seen before. If we can infer a close mosaic, we have found a “personalised reference genome”, and reads are more likely to exact-match. This approach was first described in [9], applied to the human MHC region. However their implementation was quite specific to the region and would not scale to the whole genome. Valenzuela et al [1] have recently also espoused a find-the-closest-reference approach.

Other “reference graph” approaches have been published [10–12], generally approaching just the alignment step. Siren *et al* developed an method (GCSA [11]) - however construction costs for a whole human genome plus a set of SNPs required more than 1 Tb of RAM. Huang et al [12] developed an FM index encoding of a reference genome-plus-variation (“bwbble”) by extending the genetic alphabet to encode single-character variants with new characters and then concatenating padded indel variants to the end of the reference genome. Bwbble allows reads to align smoothly across multiple SNPs and up to one indel. We do something similar, but treat all variation in an equivalent manner. While completing this paper, the preprint for GCSA2 was published ([13]), which drops RAM usage of human genome index construction to <100GB at the cost of >1Tb of disk I/O.

We show below how to encode a set of genomes, or a reference plus genetic variation, in an FM index which naturally distinguishes alternate alleles (defined below). We extend the well known BWT backward search, and show how read-mapping can be performed in a way that allows reads to cross multiple variants, allowing recombination to occur naturally. Our data structure supports bidirectional search (which underlies the Super Maximal Exact Match algorithms of bwa-mem [3]), but currently we have only implemented exact matching. We show that index construction is relatively cheap, encoding the human genome and 8.3 million genetic variants from the 1000 genomes project using just 36 GB of RAM on a single core in 7 hours. We go on to show how inferring a personalised reference results in better genome inference, looking at a highly challenging region - the MSP3.4 gene of *P. falciparum* where alleles differ by one SNP every 3 bp.

2 Background: Compressed Text Indexes

Burrows-Wheeler Transform. The Burrows-Wheeler Transform (BWT) of a string is a reversible permutation of its characters that was originally developed for compression [2]. The BWT of a string $T = t_1 t_2 \dots t_n$ is constructed by sorting its n cyclic shifts $t_1 t_2 \dots t_n, t_2 \dots t_n t_1, \dots, t_n t_1 \dots t_{n-1}$ in lexicographic order. The matrix obtained is called the Burrows-Wheeler Matrix (BWM) and the sequence from its last column is the BWT. Storing the first and last column of the BWM is sufficient for finding the number of exact matches of a query in T . For locating the position of the matches in T an additional data structure is required, the suffix array.

Suffix Arrays. The suffix array of a string T is an array of integers that provides the starting position of T 's suffixes, after they have been ordered lexicographically. Formally, if $T_{i,j}$ is the substring $t_i t_{i+1} \dots t_j$ of T and SA is the suffix array of T , then $T_{SA[1],n} < T_{SA[2],n} < \dots < T_{SA[n],n}$. It is related to the BWT, since looking at the substrings preceding the terminating character \$ in the BWM rows gives the suffixes of T in lexicographical order.

Backward search Any occurrence of a pattern P in text is a prefix for some suffix of T , so all occurrences will be adjacent in the suffix array of T , since suffixes starting with P are sorted together in a SA-interval. Let $C[a]$ be the total number of occurrences in T of characters smaller than a in the alphabet. Then if P' is a suffix of the query P and $[l(P'), r(P')]$ is its corresponding SA-interval, then the search can be extended to aP' by calculating the new SA-interval:

$$l(aP') = C[a] + \text{rank}_{BWT}(a, l(P')) + 1 \quad (1)$$

$$r(aP') = C[a] + \text{rank}_{BWT}(a, r(P')) \quad (2)$$

The search starts with the SA-interval of the empty string, $[1, n]$ and successively adds one character of P in backward order. When the search is completed, it returns a SA-interval $[l, r]$ for the entire query P . If $r \geq l$, there are $r - l + 1$ matches for P and their locations in T are given by $SA[i]$ for $l \leq i \leq r$. Otherwise, the pattern does not exist in T . If the C -array and the ranks have already been stored, the backward search can be performed in $O(|P|)$ time in strings with DNA alphabet.

Wavelet Trees Rank queries scale linearly with the alphabet size by default. The wavelet tree is a data structure designed to store strings with large alphabets efficiently and provide rank calculations in logarithmic time. A wavelet tree converts a string into a balanced binary-tree of bitvectors, whose root is built by taking the sorted alphabet and replacing the lower half of smaller symbols with a 0, and the other half of larger symbols with a 1 in the string. This

creates ambiguity initially, but at each tree level, each half of the parent node's alphabet is re-split into 2 and re-encoded, so the ambiguity lessens as the tree is traversed in depth. At the leaves, there is no ambiguity at all. The tree is defined recursively: take the lexicographically ordered alphabet, split it into 2 equal halves; in the string corresponding to the current node (start with original string at root), replace the first half of letters with 0 and the other half with 1; the left child node will contain the 0-encoded symbols and the right child node will contain the 1-encoded symbols, preserving their order from the original string; reapply the first step for each child node recursively until the alphabet left in each node contains only one or two symbols (so a 0 or 1 determines which symbol it is).

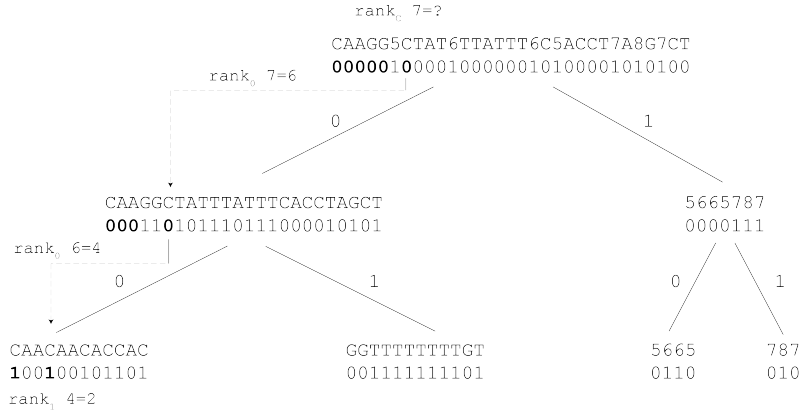


Fig. 1. Wavelet tree encoding of a string; we return to this string in Figure 2 below, as an encoding of a PRG. Calculating the rank of the marked C is performed by repeated `rank()` calls moving down the binary tree until the alphabet remaining is just 2 characters.

In order to answer a rank query over the original string with large alphabet, repeated rank queries over the bitvectors in the wavelet tree nodes are used as a guide to the subtree that contains the leaf where the queried symbol is non-ambiguously encoded. The rank of the queried symbol in this leaf is equal to its rank in the original string. The number of rank queries needed to reach the leaf is equal to the height of the tree, i.e. $\log_2 |\Sigma|$ if we let Σ be the set of symbols in the alphabet. Computing ranks over binary vectors can be done in constant time, so a rank query in a wavelet tree-encoded string has complexity $O(\log_2 |\Sigma|)$.

3 Encoding a variation-aware reference structure

3.1 Terminology

A *variant site* or *site* is a region of the chromosome where there are a number of alternative options for what sequence can be present. These alternatives are termed *alleles* and might be as short as a single character, or could be many hundreds of characters long. A *pan-genome* means a representation of a number (greater than 1) of genomes within a species.

3.2 PRG Encoding

Following [9], we use a linear PRG conceptually equivalent to a directed, acyclic, partial order graph, that is generated from a reference sequence and a set of alternative sequences at given variation loci. The graph is linearised into a long string over an alphabet extended with new symbols marking the variants, for which the FM-index can be constructed.

ref	CAAGC	TAT	--	ACCT	A	T
alt1	CAAGG	T	TATTT	T	ACCT	G
alt2	CAAGG	C	----	ACCT	A	T



CAAGG1	C	T	A	T	2	T	T	A	T	T	2	C	1	A	C	T	3	A	4	G	3	C	T
--------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Fig. 2. A simple PRG linearised according to our encoding. The first site has 3 alleles, which do not here look at all similar, and the second is a SNP. This example will be used in subsequent figures.

Building this data structure requires multiple steps.

1. First, corresponding regions of shared (exact match) sequence between the input genomes must be identified. These must be of size k at least (where k is pre-defined), and act as anchors for the coordinates of the variation sites.
2. Second, for any locus between two anchor regions, the set of possible haplotypes must be determined from the input genomes, but do not need to be aligned. Indels are naturally supported by haplotypes of different lengths.
3. Each variation locus is assigned two unique numeric identifiers, one even and one odd. The odd identifiers will mark locus boundaries and the even identifiers will mark alternative allele boundaries.
4. For each variation locus, its left anchor is added to the linear PRG, followed by its odd identifier. Then each sequence coming from that locus, starting with the reference sequence, is successively added to the linear PRG, followed by the even locus identifier, except the last sequence, which is followed by the odd identifier.
5. Convert the linear PRG to integer alphabet ($A \rightarrow 1$, $C \rightarrow 2$, $G \rightarrow 3$, $T \rightarrow 4$, variation locus identifiers $\rightarrow 5, 6, \dots$)

6. The FM-index (suffix array, BWT, wavelet tree over BWT) of the linear PRG is constructed and we will call this the vBWT.

An illustration of these steps on a toy example is given in Figure 2.

Importantly, *the markers force the ends of alternative sequences coming from the same site to be sorted together in a separate block in the Burrows-Wheeler matrix, even if they do not have high sequence similarity*. Therefore, alternative alleles from each site can be queried concurrently.

3.3 Graph structure: constraints

We show in Figure 3a two chromosomes which differ by 3 SNPs, and in 3b) and 3c) give two graph encodings. Both represent the sequence content equally well, and we allow both. However, we show in 3d) an example where a long deletion lies “over” two other alleles. We would encode this in our PRG as shown in 3e). This works, but results in many alternate alleles. An alternative would be to allow “nested” variation, where variants lie on top of other alleles, as shown in Figure 3f). This could certainly be encoded in our system, but we do not allow it for our initial implementation, as it would potentially impact mapping speed. Note that this constraint is shared by bwbb.

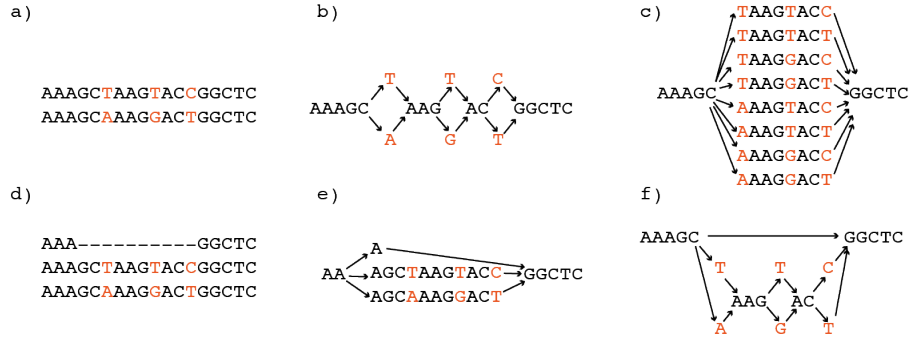


Fig. 3. PRG graph structure. The sequences shown in Figure 3a) could be represented either as 3 separate mutations (shown in b)), or enumerated as 8 small haplotypes, shown in c). Both are supported by our encoding. Similarly, the sequences in d) could be represented in our implementation as shown in e). However, we do not support “nesting” of alleles, as shown in f).

4 Variation aware backward search in vBWT

In this section, we present a modified backward search algorithm for exact matching against the vBWT that is aware of alternative sequence paths. When reads

align to the non-variable part of the linear PRG or when a variant locus is long enough to enclose the entire read, the usual backward search algorithm can be used. Otherwise, when the read must cross variation site junctions in order to align, site identifiers and some alternative alleles must be ignored by the search. This means a read can align to multiple substrings of the linear PRG that may not be adjacent in the BWM, so the search can return multiple SA-intervals. We give pseudocode in Algorithm 1 below, and outline the idea in Figure 4.

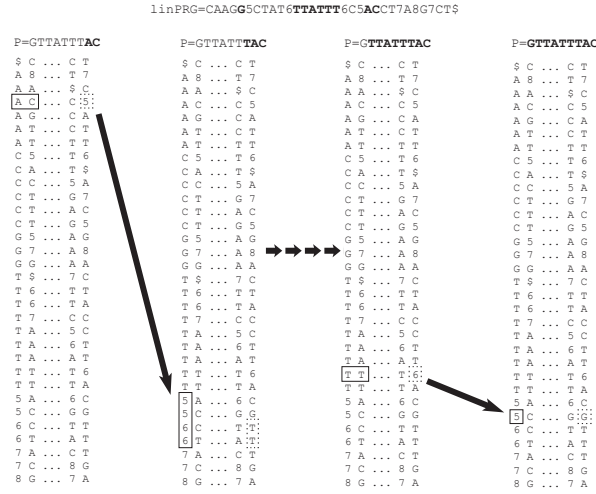


Fig. 4. Backward search across vBWT. We start at the right-hand end of the read $GTTATTTAC$, with the character C , and as we extend we hit the character 5 , signalling the start or end of a variation locus. We check with a mask, and find it is the end. Therefore the read must now continue into one of the alleles, signalled by the number 6 . Continuing in this manner (the shorter arrows signify multiple intermediate steps not shown) we are able to align across the site.

At each step in backward search, before extending to the next character, we need to check whether the current matched read substring is preceded by a variation marker anywhere in the linear PRG. A scan for symbols larger than 4 (“range_search_2d” in the pseudocode) must be performed in the BWT within the range given by the current SA-interval. If a variation marker is found and it is an odd number, the read is about cross a site boundary, i.e. is about to walk in or walk out of a site. The suffix array can be queried to find the position of the two odd numbers in the linear PRG: the number occurring at a smaller position will mark the beginning of site and the other one will mark the end of site.

Algorithm 1 Modified backward search**Input:** pattern $P[0, m]$ and FM-index of PRG in integer alphabet**Output:** list of SA intervals corresponding to matches of P

```

1:  $l \leftarrow C(P[m])$ 
2:  $r \leftarrow C(P[m] + 1)$ 
3:  $i \leftarrow m - 1$ 
4:  $\text{SA\_int} = [l, r)$ 
5:  $\text{Extra\_int} = \emptyset$  ▷ keep list of SA intervals
6: while  $i > 0$  and  $\text{SA\_int} \neq \emptyset$  do
7:   for all  $[l, r) \in \text{SA\_int}$  do
8:      $M \leftarrow \text{WT.range\_search\_2d}(l, r - 1, 5, |\Sigma|)$  ▷ find variation site markers
9:     for all  $(idx, num) \in M$  do ▷  $idx \in [l, r), num \in [5, \Sigma]$ 
10:      if  $num \% 2 = 0$  then
11:         $\text{odd\_num} = num - 1$ 
12:      else
13:         $\text{odd\_num} = num$ 
14:      if  $\text{CSA}[C(\text{odd\_num})] < \text{CSA}[C(\text{odd\_num}) + 1]$  then
15:         $\text{start\_site} \leftarrow C(\text{odd\_num}), \text{end\_site} \leftarrow C(\text{odd\_num}) + 1$ 
16:      else
17:         $\text{start\_site} \leftarrow C(\text{odd\_num}) + 1, \text{end\_site} \leftarrow C(\text{odd\_num})$ 
18:      if  $num \% 2 = 1$  and  $\text{CSA}[idx] = \text{CSA}[\text{site\_end}] + 1$  then
19:         $\text{Extra\_int} = \text{Extra\_int} \cup \{[C(num), C(num + 2))\}$ 
20:      else
21:         $\text{Extra\_int} = \text{Extra\_int} \cup \{[C(\text{start\_site}), C(\text{start\_site}) + 1)\}$ 
22:     $i \leftarrow i - 1$ 
23:   $\text{SA\_int} = \text{SA\_int} \cup \text{Extra\_int}$ 
24:  for all  $[l, r) \in \text{SA\_int}$  do
25:     $l = C(P[i]) + \text{rank}_{\text{BWT}}(P[i], l - 1)$ 
26:     $r = C(P[i]) + \text{rank}_{\text{BWT}}(P[i], r)$ 

```

If the search cursor is next to the start of the site, it is just the site marker that needs to be skipped so the SA-interval (size 1) of the suffix starting with that marker needs to be added to the set of intervals that will be extended with the next character in the read. If the search cursor is next to the end of a site, all alternative alleles from that site need to be queried. Their ends are sorted together in the BWM because of the markers, so they can be queried concurrently by adding the SA-interval of suffixes starting with all numbers marking that site (even and odd).

If the variation marker found is an even number, the read is about to cross an allele boundary, which means its current suffix matches the beginning of an alternative allele and the read is about to walk out of a site, so the search cursor needs to jump to the start of site. As previously described, the odd markers corresponding to that site can be found in the sorted first column of the BWM, and then querying the suffix array decides which one marks the start of site. Then the SA-interval (size 1) for the BWM row starting with this odd marker is recorded. Once the check for variation markers is finished and all candidate SA-

intervals have been added, each interval can be extended with the next character in the read by using equations 1 and 2.

5 Performance

5.1 Construction cost : the human genome

We constructed a PRG from the human reference genome (GRC37 without “alt” contigs) plus the 1000 genomes final VCF (12GB in size) [6], as described below. We first excluded structural variants which did not have precisely specified alleles, and variants with allele frequency below 5% (rare variation offers no benefit as use the PRG to find a mosaic reference which differs ideally by isolated SNPs only). If two variants occurred at consecutive bases, they were merged into one, and all possible haplotypes enumerated. If the VCF contained two consecutive records which overlapped, the second was discarded. This resulted in a dataset of 7.4 million SNPs and 978000 indels. We give construction costs in Table 1, along with comparative figures for Bwbbble with identical input.

Table 1. FM index construction costs for human reference genome plus 1000 genomes variants

	Software Memory(GB)	Time (hrs/mins)
vBWT	36	7h5m
Bwbbble	60	1h5m

For comparison, using common Finnish SNPs, GCSA took over 1TB of RAM building chromosomes separately and pruning the graph in high diversity regions. GCSA2 reduces the memory footprint to below 128GB RAM, running in 13 hours with 32 cores, and using over 1Tb of I/O to fast disk. Our vBWT construction has a lower memory cost than GCSA, GCSA2 and bwbbble, is faster than GCSA/GCSA2, has no (significant) I/O burden, but is significantly slower than bwbbble.

5.2 Inferring a Closer Reference Genome

P. falciparum is a haploid parasite that undergoes recombination when inside a mosquito. It has an unusually repetitive genome that contains more indels than SNPs [15]. There are several regions that present challenges to mapping because samples often diverge strongly from the reference. For example, the merozoite surface protein gene MSP3.4 is known to have two highly diverged lineages at a 500bp region (the DBL domain), which differ by around 1 SNP every 3 bases.

Thus for many samples a “pileup” of mapped reads simply shows a hole (see Figure 6). We constructed a catalog of MSP3.4 variation from Cortex [16] and GATK [17] variant calls from 700 *P. falciparum* samples from the Ghana, Laos

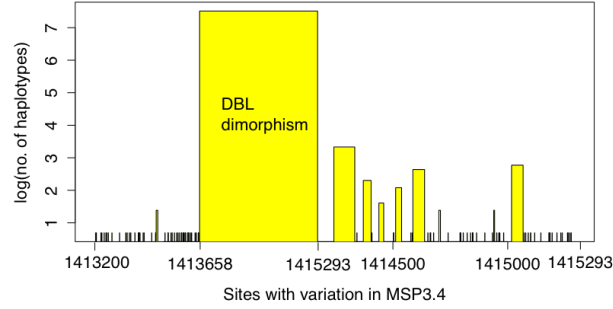


Fig. 5. Histogram of number of alleles at each site in MSP3.4 plotted above the chromosome coordinate.)

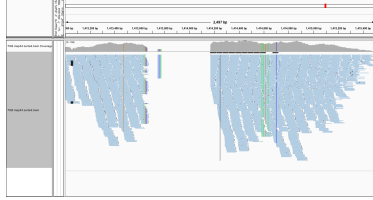


Fig. 6. Mapping reads from sample 7G8 to *P. falciparum* 3D7 reference genome results in a gap.

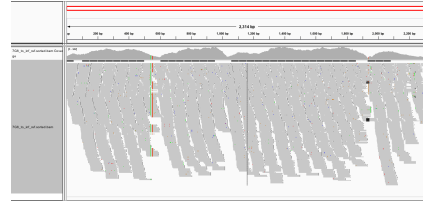


Fig. 7. Mapping reads from sample 7G8 to our vBWT-inferred genome removes the gap, leaving isolated variants easy to detect with standard methods

and Cambodia, and built a PRG just for that gene embedded in its chromosome. We aligned Illumina reads from a well-studied sample that was not used in graph construction (named 7G8) to the PRG using backward search (exact matching), and collected counts on the number of reads supporting each allele. We used a simple “heaviest path” method similar to that outlined by Valenzuela et al [1] to choose the path through the graph which best supported the data. This was our graph-inferred personalised reference for this sample. We then mapped the reads (using `bwa_mem` [18]) to the inferred genome. As can be seen in Figure 7, this gives dramatically better results.

5.3 Further performance comments

One consequence of our approach, shared with `bwbb`, is that a single pattern can match multiple intervals in the BWT. Thus for mapping we store a list of intervals, along with which sites and alleles are crossed. Our current implementation of this is naive (using C++ `std::unordered_map`). We also store an integer array (mask) allowing us to determine if a position in the PRG is in a site, and if

so, which allele - again naively encoded (in `std::vector`). For the human genome example these cost us around 12GB of RAM. This sparse array, which contains a zero at every non-variable site in the chromosome, could be stored much more compactly.

This encoding results in an alphabet of size $4 + 2n$ where n is the number of variant sites. We used the wavelet tree to mitigate this cost from $O(n)$ to $O(\log(n))$, but to get an estimate of how this impacts performance in practise, we measure speed of exact matching of simulated perfect reads from *P. falciparum* chromosome 10 with only MSP3.4 variation inserted. To avoid unnecessarily repeatedly shrinking the same BWT intervals, we store in a hash the BWT intervals corresponding to all 9-mers in the PRG. This results in read-matching at a rate of 277/second.

Finally, there is one performance improvement which we have yet to implement - precalculating and storing an array of ranks at marker positions across the BWT - just as in a standard FM-index. This is not normally done for large alphabet wavelet-tree-based suffix arrays, but we can ignore the numeric characters, and store only for A,C,G,T.

6 Discussion

By extending the alphabet and placing positional markers, we are able to ensure that alternate alleles sort together in the BWT matrix, allowing mapping across sites and recombination. In fact, we could encode quite general graph structures in this manner, but for simplicity we imposed a constraint on graph construction, and did not allow nesting of sites. For haploids, the idea of systematically choosing a closer reference, and then using the standard tools is easily communicated and can be directly integrated into standard analyses. Secondly, for other ploidies, our implementation readily lends itself to collecting phase-informative read-data, running an HMM, and finally MEM-based graph alignment. We have made a choice to encode genetic variation as a Directed Acyclic Graph - thus inversions or rearrangements must be encoded as explicit alternate alleles. Our choice was deliberate - we are aiming for a more limited model, but avoiding heuristics in the details of the graph construction, especially in regions of high diversity.

Software We have implemented the vBWT twice. First as a simple prototype, which provided the results on *P. falciparum* above. Secondly, a more careful implementation that provided all other results, and which is available here: <http://github.com/iqbal-lab/gramtools>. We make the prototype available here purely for sake of reproducibility URL, but all future development (and maintenance) is on the second implementation.

Acknowledgments. We would like to thank Jerome Kelleher, Heng Li, Rayan Chikhi and Jouni Siren for discussions, and Lin Huang and Victoria Popic for

help running bwbbble. Above all we would like to thank Simon Gog and the SDSL developers for providing a valuable and well-maintained resource. `FUNDING`

References

1. Valenzuela, D., Valimaki, N., Pitkanen, E., Makinen, V. On enhancing variation detection through pan-genome indexing. *Biorxiv*. <http://dx.doi.org/10.1101/021444>
2. Burrows, M., Wheeler, D.J. :A block sorting lossless data compression algorithm. Digital Equipment Corporation, Tech. Rep. 124, 1994. Available: <http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.html>
3. Li, H., Durbin, R.: Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics* 25 (14): 1754-1760 (2009)
4. Langmead, B., Salzberg, S.: Fast gapped-read alignment with Bowtie 2. *Nature Methods*. Mar 4;9(4):357-9 (2012)
5. Reinert, K., Langmead, B., Weese, D., et al: Alignment of Next-Generation Sequencing Reads. *Annu Rev Genomics Hum Genet.* 2015;16:133-51
6. The 1000 Genomes Project Consortium: A global reference for human genetic variation. *Nature* 526, 68-74
7. Ossowski, S., Schneeberger, K., Clark, R.M., et al. Sequencing of natural strains of *Arabidopsis thaliana* with short reads. *Genome Research* 18, 2024-2033 (2008)
8. Jeffares, D.C., Rallis, C., Rieux, A., et al. The genomic and phenotypic diversity of *Schizosaccharomyces pombe*. *Nature Genetics* 47, 235-241 (2015)
9. Dilthey, A., Cox, C., Iqbal, Z., et al: Improved genome inference in the MHC using a population reference graph. *Nature Genetics* 47, 682-688 (2015)
10. Schneeberger, K., Hagmann, J., Ossowski, S., et al. Simultaneous alignment of short reads against multiple genomes. *Genome Biol.* 10, R98 (2009).
11. Siren, J., Valimaki, N., Makinen, V. Indexing Graphs for Path Queries with Applications in Genome Research. *IEEE/ACM Trans Comput Biol Bioinform.* 2014 Mar-Apr;11(2):375-88
12. Huang, L., Popic, V., Batzoglou, S. Short read alignment with populations of genomes. *Bioinformatics*. Jul 1;29(13):i361-70 (2013)
13. Siren, J. Indexing Variation Graphs. *arXiv:1604.06605*
14. Na, J.C., Crochemore, M., Park, H., et al: Suffix tree of alignment: an efficient index for similar data. *Combinatorial Algorithms ? Revised Selected Papers of the 24th International Workshop, IWOCA 2013, Rouen, France, July 10-12, 2013* (2013), pp. 337-348
15. Miles, A., Iqbal, Z., Vauterin, P., et al. : Genome variation and meiotic recombination in *Plasmodium falciparum*: insights from deep sequencing of genetic crosses *Biorxiv*. <http://dx.doi.org/10.1101/024182> (2015)
16. Iqbal, Z., Caccamo, M., Turner, I., et al: De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics* 44, 226-232 (2012)
17. DePristo, M., Banks, E., Poplin, R.E., et al: A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature Genetics* 43(5): 491-498 (2011)
18. Li, H. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv:1303.3997*

7 Checklist of Items to be Sent to Volume Editors

Here is a checklist of everything the volume editor requires from you:

- ☐ The final L^AT_EX source files
- ☐ A final PDF file
- ☐ A copyright form, signed by one author on behalf of all of the authors of the paper.
- ☐ A readme giving the name and email address of the corresponding author.