Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Manuscript Category

OXFORD

Applications note

Stochastic Variant Graph: Optimal read mapping using A*

Rubber Duck [1,*],

[1]Zoo Pond, Switzerland

[*]To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Algorithmic reliability is an important aspect in many scientific and engineering fields. Despite the advancements in computational genetics, the focus has mostly been on performance while sacrificing the provided guarantees. We resolve the long-standing trade-off between optimality and performance for the read to graph mapping problem. Here we formally define the problem of read to graph mapping and present a provably-optimal practical solution. We developed Probabilistic Variant Graph (PVG) – a generative stochastic model designed for read mapping to general reference graphs. It completely encompasses all of the various available data sources in their probabilistic interpretation: 1) read quality scores, 2) edit distance alignment metric, and 3) variant likelihood statistics. This model captures an unprecedented level of generality while guaranteeing global seed-and-extend mapping optimality. Our novel heuristic precomputation allows an A* shortest path algorithm to reach a near input-size empirical complexity. We demonstrate that aligning using our model improves the mapping accuracy of current state of the art tools while matching the performance on both synthetic and real data. Because of being a generative model, we are able to optimize the alignment probability P(read|path) but also to estimate the marginal likelihood and mapping probabilisty P(path|read). The PVG tool is available on https://github.com/eth-sri/pvg.

1 Introduction

FiXme: make sure to incorporate Comparison of sequence to graph alignment algorithms

Overviews: A History of DNA Sequence Assembly[29], [2].

Finding the origin of newly sequenced genomic sequences is a routine task of bioinformatics, consisting of mapping genomic sequences (e.g. reads directly produced by a sequencing machine) onto linear reference genomes that have been precomputed and humanly curated. However, working with a single reference genome cannot be representative for the whole diversity of an organism and introduces a bias. Instead, mapping on a collection of genomes has been proposed as an extension in order to account for the whole distribution. This can be especially useful when dealing with microbial genomes (>100k or even 1M different reference). Good candidates for representing such genomic variety are the genome graphs which are already heavily used in bioinformatics for tasks like de-novo assembling. Here we will focus on de Bruijn Graphs (DBGs). Various queries need to be translated from linear to graph models (e.g., read mapping and variant calling).

The common query of mapping a genomic sequence is translated as finding such a path (or sometimes several candidate paths) in the DBG that spells a string which best aligns with the query sequence. There are different ways to define the quality of alignment but the commonly used are exact match (which can efficiently implemented by hashing), number of substitutions (Hamming distance), and edit distance (which accounts of the biologically relevant insertions and deletions). In practise, the more general edit distance is rarely applied to the huge number of short reads because of computational burden. Nevertheless, an exact algorithms with optimal worst case time complexity $O(V + mE)$ (m – query sequence length) has been recently developed for general directed graph (cycles allowed).[36] Such a complexity is still too slow for full-length genome graphs but is not prohibitive for smaller graphs like Ig graphs (similar to the ones constructed in [3] with similar complexity algorithms).

The tool vg[1, 32, 12] has been developed as a practical general graph reference (both DNA and RNA). It heavily uses hashing for kmer mapping which is further extended. It "dagifies" a graph (unrolls and unfolds any cycles) to create a directed acyclic graph (DAGs) that are amenable to partial-order alignment.[32] (This can be problematic for long reads mapped on complex cyclic graphs) The authors note that:

"The current version of vg uses heuristics in place of a true probability model for its mapped reads. We expect that true generalizations of traditional models will be a significant advancement for the field, particularly for segregating structural variants."

"The leading linear reference-based variant calling tools in use today are all based on probabilistic models of sequencing data (Nielsen et al. 2011). This approach has several advantages. Modern sequencing technologies all attempt to quantify the uncertainty in their base calls. Probability models provide a natural framework to incorporate this uncertainty into genotype calls, and they allow algorithms to estimate uncertainty about genotype calls for downstream analyses"

Another line of molecular genomics research involves probabilistic models to capture different kinds of uncertainty (both biological and technical) in order to reason more precisely. An example of such a query is variant calling. genotype $G(AA, AT, ...)$ with maximum the posterior probability $P(G|X)$ given the data X. This is done by using Bayes' formula and a direct computation of $P(X|G)$. The ratio between the most probable and the second most probable genome is used as an estimate for the inference uncertainty.[31] FiXme: include the variant calling query?

Particularly interesting use cases for a probabilistic framework are some complex genome regions that are highly variable (e.g. Ig, TCR, BCR, HLA, MHC genes).

We argue that our precise inference approaches scales for many domain-specific cases. The computational complexity for read mapping is close to linear of the read length thanks to the A*/ALT optimization strategy [14].

In this work, we aim at extending the applicability of DBG and various genomic operations to a more general probabilistic (uncertainty) model that also captures edit operations. This task has been underlined to be of significant practical importance[32].

An alternative to MAP optimization is centroid estimation[4] which may be more appropriate in cases when the MAP is not near one. It has been applied to alignment[16].

Unlike other graph-based reference approaches, pVG smooths the concept of read mapping from mapped vs not mapped to mapping with a probabilistic score. In other words, it will always map a read with the difference in the mapping quality. Or in terms of generation, even the simplest pVG can generate all possible (infinitely many) sequences.

2 Contibutions

As mentioned in different

1. problem definitions and approaches: prob.matching (seq to read) $\rightarrow$ optimal mapping on a PNFA (DP, Dijkstra, A*) $\rightarrow$ construction (from reference seqs or reads) $\rightarrow$ DBG conversion to PNFA $\rightarrow$ queries
2. unifying three sources of uncertainty
3. subsumes ED minimization (by using zero phred scores, zero supersource jumps scores, zero transition scores)
4. subsumes MEM maximization (by using zero phred scores, zero supersource jumps and zero supersink jumps, zero edit scores, negative constant transition scores)
5. jointly solving mapping and aligning tasks
6. implementaion (dp, dijkstra)-testing-comparisons-applications-two metrics (avg and total probability). compare Dijkstra and A* in two regimes (+edit edges; +phread) to VG tool and Minimap2 (instead of BWA-MEM) on ecoli, VDJ data and synthetic data (get Pevzner's VDJ graph or the HMM) for mapping and alignment accuracy and for performance
7. stability checking: varying parameters, abstract interpretation; how much change of edit probs is needed for a different best alignment
8. Get the Ig Graph from Pevzner's paper [3] synthetize graphs and queries with substitution, insertion and deletion errors. Learn the probabilities on the edges (+probabilities on indel and substitution edges) using big data over TCR/BCR. Run TraCeR to obtain sequence candidates (before TCR type annotation per nucl.) Feed the candidates to the T5 algo (implementation) to annotate them (use GAM format from VG tool; and also CIGAR) Validate the T5 annotations using IgBLAST annotations (=TraCeR annotation). Map kmers directly from the input to the graph. Compare with [35]. Compare to vg, HMM, linear state of the art.

2.1 Queries

Several successive tasks are typical for vg:

1. Convert a reference genome or a set of sequences to a DBG (construct or msga commands in vg) – learning an automata
2. Mapping reads (map) on the DBG using kmers hashing or maximal exact matches (MEMs)
3. Calling variants (call and genotype) – predicting the likelihood of variation at each locus after alignment, SNP, short indels
4. Drawing a string from the pDBG (most probable, random or median–minimizing the expected dist to the pDBG)

In the context of pDBGs, the upper tasks will translate to:

1. Construct a pDBG that captures technical noise, population variation, etc. (e.g. $p(u, v) = -log(\#reads(u, v))$)
2. Map reads MEMs with a threshold

2.2 Uncertainty sources

We account for three sources of uncertainty that we want to account for (we shortly call them *variants*, *edits* and *phreds*).

1. **variants**: distinguishing variation represented in the graph (different references): e.g. populational variation like SNPs and highly variable genomic regions (e.g. TCRs)
2. **edits**: variation outside the graph, produced the new genomic sequence (subst, del, ins, transpositions?), able to capture somatic variation that is not represented in the reference graph
3. **phreds**: technical errors while sequencing (substitution errors in the query, aka base calling), especially important when the sequencing depth is small and the error rates are high (e.g. current long read technologies with >10

2.3 Comparison to existing approaches

FiXme: Add columns to the comparison table: complexity, short/long read, single/pair-end, dataset, multiple reads (e.g., extend from seed-and-extend, MSA) FiXme: Add rows to the comparison table: check with https://genome.cshlp.org/content/27/5/665.full.pdf[32]

Multiple Genome Index(MuGI'14)

The Na'09 paper[30] aligns a read to another read. They modify the edit penalties based on quality scores. The evals are just counting how often the standard DP finds exactly the same alignment as the modified DP with the quality scores finds.

LAST'10[11] aligns a read to a genome allowing only for mismatches because of sequencing or wrong mapping. The evals include both synthetic and real data. In the simulated tests, they sample 36bp fragments with different levels of random substitutions (modeling the biological variation) and add noise from real phred values. The real setting is xeno-mapping of reads from one specie to the genome of another closely related: firstly

tool	**reference graph**	**variants**	**edits**	**phred**	**quality score**	**probabilistic**	**no adhoc**	**efficiency**	**evals**
Smith-Waterman'81 [39]	linear	-	✓	-	-	-	✓	✓	?
MAQ'08 [24]	linear	-	(-)	✓	✓	✓(approx.)	✓	✓	?
Bowtie2'12 [23]	linear	-	✓	-	?	- (ED)	?	✓	?
Na'09 [30]	linear	-	✓	✓	?	✓?	?	?	exact alignment when changed DP scores
LAST'10 [11]	linear	-	-	✓	✓	✓?	-	?	synth (subst+phred), real (on close specie)
BWA-MEM	linear	-	-	-	?	- (MEM)	?	?	?
ERG'12 [40]	linear+variants?	?	?	?	?	?	?	?	RNA
BlastGraph'12 [18]	✓?	✓	(✓)	-	-	-	-	-	-
BWBBLE'13 [19]	many linear?	✓	?	?	?	?	?	?	?
MuGI'14 [5]	many linear	✓	?	?	?	?	?	?	?
GCSA'14 [38]	DAG*	✓	(✓) ≤ 3	-	-	-	✓	(✓) when ED is small	mapping and variation on HG
HISAT2'15 [22, 38, 23]	DAG*?	✓	?	-	?	?	?	✓	-?
PRG'15 [7]	DAG	✓	(-)	-	-	?	-	-	only MHC genotypes
deBGA'16 [26]	de Bruijn Graph	✓	-	-	?	-	?	✓	sim and real on HG
BGREAT'16 [25]	DAG	✓	subst	-	-	-	-	✓	too simple
Gramtools'16 [27]	DAG	✓	-	-	-	-	-	✓	-
partis'16 [35]	linear HMMs	✓	subst	-	-	✓	(✓)	-	VDJ annotation
DP'17 [36] (no tool)	✓	✓	✓	-	-	- (ED)	✓	-	-
Graphtyper'17 [8]	DAG	✓	1subst	-	-	max.match	?	✓	only genotypes
VG'17 [1, 32, 12]	✓	✓	-	-	?	- (MEM)	?	✓	HG?
IGoR'18 [28]	?	✓	subst	-	?	- (ED)	?	?	VDJ
BrownieAligner'18 [17]	✓	✓	✓	-	?	?	?	?	
pVG	✓	✓	✓	✓	✓	✓	✓	Dijkstra (LN nodes, LM edges)	VDJ

Table 1. Comparison of existing mapping approaches (separate aligning steps not included). Edits are insertions, deletions and substitutions. MAQ accounts for edits after mapping. PRG accounts for edits after mapping and linearization. Only evaluated features are included. The DAG* can be extended to general graph.

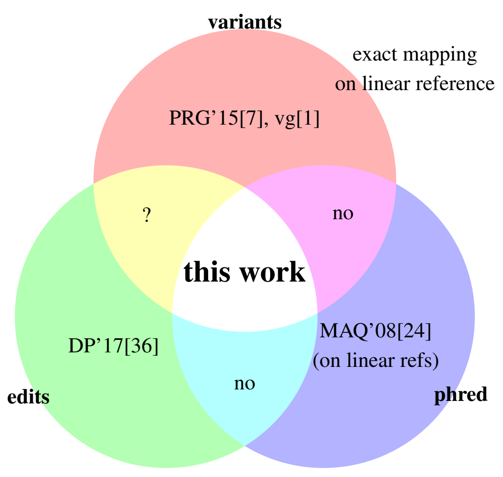

variants
exact mapping on linear reference
PRG'15[7], vg[1]
?
no
this work
DP'17[36]
MAQ'08[24] (on linear refs)
no
edits
phred

Fig. 1. Comparison of existing mapping approaches based on supported sources of uncertainty: different weighting of reference variants, edit distance minimization to reference, and query phred values

different sets of edit penalties are estimated, then three experiments are conducted: accounting for phreds, phreds+edits, phreds+edits+gaps.

The BlastGraph'12[18] tool maps using seed hashing and aligns to the left and right of the seeds using DP minimizing edit distance. It is evaluated on DBGs built using a 10K, 100K and 100K reads out of approx. 4M Illumina single-end 36nb reads (sra:DRR000096). The edits compensate for technical noise but not for biological noise. The eval does not include bio variability.

Generalized compressed suffix array'14 (GCSA) is a BWT extention to graphs. In order to support edits, full set of possible edits should be generated in the index. It is built from a reference genome + known variation or by a multiple sequence alignment. Indexes supporting ED of ≤ 0, 1, 2 and 3 are compared to RLCSA (Find and Locate) and BWBBLE (Find) on HG. Evaluated on the Variation dataset'13[]. They find out that with increasing the allowed ED, mapping on the graph stops finding more or more unique mappings than on the linear reference (allowing for the same ED).

PRG'15[7] constructs a graph with edge probabilities by chopping reads to kmers. Then the whole reads are mapped to the graph and the two most probable haplotypes. This means that the reads are not mapped to an independent reference genome.

Gramtools'16[27] uses BWT to represent variations of the reference. Evaluated for speed not accuracy.

deBGA'16 [26] indexes a DBG and introduces a graph-based seed-and-extension algorithm. "Generally, seed-and-extend aligners consist of two steps: (i) seeding: the inference of putative read positions (PRPs) from the matches (hits) of tokens (seeds) between the read and reference genome; and (ii) extension: alignment of the read to the region surrounding each PRP to determine the most likely read position(s)" "One of the major bottlenecks faced by this approach is how to handle repetitive genomic regions, even in the context of a single genome, for example, over 50% of the human genome comprised repeats" The evals measure the sensitivity and accuracy for read alignment against a single genome using simulated and real pair-end reads. Compared to Bowtie2, BWA, BWA-MEM, STAR, SeqAlto and GEM.

ERG'16 [40].

BGREAT'16[25] maps single-end reads on DBGs using an exhaustive solution and a greedy speedup. Evals consist of comparing mapping percentable on contigs vs the graph, calculating percentage of reads that map on branching paths. The accuracy evals are only with sythetic reads with sequencing errors from HG being mapped on the same HB DBG.

Could compare only to BlastGraph but it was too unstable. We subsume this method as the DBG can be converted to pVG.

Partis'16[35] precomputes the structure of a set of separate linear HMM for each VDJ recombination. Then, based on the set of input sequences (e.g., reads) it estimates allele usage probs, transition probs and emission probs and annotates each sequence by VDJ alleles. Given a VDJ distribution and priors on gap size, naive sequences are simulated after which they are mutated (similar to BCR) into a set of sequences of a clonal family (using TreeSim). Partis does inference by finding a path in the HMM that has maximum probability to generate a single or multiple sequences (multi-HMM). The V, D and J allele correctness is compared to IMGT, IgBLAST, iHMMunealign and ighutil. Uses SW alignment.

Graphtyper'17[8] creates a DAG out of a reference and a set of variants. Mapping is by hashing kmers (seed-and-extend) which are extracted from a read with 1bp overlap, then extending the seeds of overlapping kmers on the reference, then BFS through the graph structure of overlapping kmers (without any mistakes for now), then extracting kmers with 1 mistake from a read (and its reverse complement) that overlaps by 1bp. Evaluated are only genotype calls but not the alignments.

FORGe'18[34] studies the drawbacks and trade-offs between accuracy and overhead when adding more reference genomes to the graph. They suggest a method for scoring variants based on the effect on accuracy and the computational overhead and deciding which to add to the graph. Compared to HISAT2, ERG,

IGoR'18[28] has 3 regimes: SW alignment, infer a model and evaluating sequence statistics. Compares to Partis and MiXCR using their own simulated data (instead of reusing). It does SW alignment.

BrownieAligner'18[17] works on de Bruijn graphs. Claims alignment optimality but depends on initial heuristic seeding. A Markov Model is built by aligning the reads on the graph, and then used for heuristically pruning the spurious paths. It captures more long-distance relationships than k. Seems to use a cutoff of 2 indels for a banded DP. Its scores (+1,-1,-3) are not general and what is most important, the matching score should be >0 (as commented in the code). No phreds support. No probabilistic interpretation. In the implementation, they perform DFS with a stack instead of priority queue (as shown in the paper). They evaluate on simulated and on real reads from one genome: reads are simulated using ART tool[20], and the real reads true locations are assumed to be the ones found by bwa (no variant data). B&B speeds-up DFS between 1.1 times and 4-5 times.

In[12] VG is evaluated by mapping simulated reads from a 150bp pair-end reads

As visualized on figure 1, existing approaches regard some of the mentioned uncertainty sources but do not unify them in order to extract full advantage of the available data.

3 Problem Statement

pVG phrases alignment as a probabilistic inference task that we introduce and motivate in this section. The inference task is based on a generative model that specifies how reads are generated. A mathematically precise formulation of this generative model ensures that pVG solves the correct alignment task.

Notation. Table 2 provides a summary of notation used in this work. The first two parts of Table 2 provide a quick reference for the notation introduced in this section. For now, we only note that we distinguish sets of instances (e.g., $\mathcal{G}$), random variables inducing a distribution over these sets (e.g., G), and instances of random variables (e.g., g).

The last part of Table 2 summarizes notation we use without further comments. The only non-standard notation is $g[\![i]\!]$ (and $g[\![i{:}j]\!]$), which we use to indicates the positions at which a given read could align. Formally, we can interpret $g[\![i]\!]$ as the tuple (g, i).

	Set	Random Variable	Instance
Genome	$\mathcal{G}, \mathcal{G}^{\odot}, \mathcal{G}^{🔧}$	$G, G^{\odot}, G^{🔧}$	$g, g^{\odot}, g^{🔧}$
Alignment	$\mathcal{A}$	A	$a, a^{\star}$
Genome Segment	$\mathcal{S}, \mathcal{S}^{👁}$	$S, S^{👁}$	$s, s^{👁}$
Start of segment	$\{0, \dots, \lvert g\rvert - 1\}$	$B, B^{🔧}$	$b, b^{🔧}$

Superscript indicating core genome	$\odot$
Superscript indicating mutated genome	🔧
Superscript indicating read of genome segments	👁

Letter at i^{th} position of genome g (0-based indexing)	$g[i]$
Position i in genome g	$g[\![i]\!]$
Letters/Positions i (inclusive) to j (exclusive) of genome g	$g[i{:}j]$, $g[\![i{:}j]\!]$
Length of genome or segment	$\lvert g\rvert$, $\lvert s\rvert$
Set of paths of length L in graph $\mathbb{G}$	$\text{Paths}_L(\mathbb{G})$

Table 2. Summary of notation.

Inspiration: Complete Genomes Model. Our generative model is inspired by a model of biological reality, depicted in the first column of Fig. 7.

It starts from a random variable G over the set $\mathcal{G} = \{g_i\}_{i \in \mathcal{I}}$ of haploid genomes occurring in the population under consideration. The genomes in $\mathcal{G}$ exhibit both small-scale and large-scale variations appearing in the population. Small-scale variations include de-novo mutations and other rare mutations, while large-scale variations capture common mutations in the population that typically span multiple base pairs. The distribution of the random variable G captures the probability of each genome g. We note that this distribution is typically not uniform, as some variations are more common than others. As a consequence, ignoring the distribution of G (as done, e.g., by the Smith-Waterman algorithm) will necessarily lead to loss of information.

The model first samples a genome g from G. Then, it sequences g, resulting in a read $s^{👁}$ of a predetermined length L. Sequencing consists of (i) randomly chopping the genome into smaller segments, (ii) discarding segments shorter than L and selecting one remaining segment s of length at least L to read, and (iii) reading the first L letters in s, resulting in $s^{👁}$. In general, $s \neq s^{👁}$ even if the length of s is L, as reading is noisy and may introduce *read errors*.

We assume the model does not only report the read $s^{👁}$, but also the probability of reading each letter correctly. Typically, this probability is represented as a phred quality score [9]:

$$Q_i := -10 \cdot \log_{10} \Pr\left[\text{error reading } i^{\text{th}} \text{ letter}\right],$$

However, we work with phred probabilities instead, denoted by

$$\text{phred}_i := \Pr\left[\text{correctly reading } i^{\text{th}} \text{ letter}\right].$$

3.1 Our Model: Core Genomes Model

Instantiating the Complete Genomes Model in practice is impossible; we typically cannot even enumerate all genomes in $\mathcal{G}$. Instead, we introduce the Core Genomes Model, which starts from a random variable $G^{\odot}$ over a set $\mathcal{G}^{\odot}$ of *core genomes*. In contrast to $\mathcal{G}$, the core genomes in $\mathcal{G}^{\odot}$ only capture the large-scale variations appearing in the population.

Mutations. Our model introduces small-scale variations by first sampling a core genome $g^{\odot}$ from $G^{\odot}$ and then mutating it. To mutate $g^{\odot}$, our model copies it letter by letter, generating a mutated genome $g^{🔧}$. When the model is about to copy the i^{th} letter of $g^{\odot}$, it probabilistically decides to (i) insert a new letter (insertion), (ii) skip this letter (deletion), (iii) copy the letter incorrectly (edit), or (iv) copy the letter correctly (copy).

```
1  def mutate_letter(letter):
2    actions = [(p_ed / 3, 'edit', l) for l in letters if l != letter] + \
3          [(p_ins / 4, 'ins', l) for l in letters] + \
4          [(p_del, 'del', ε)] + \
5          [(p_copy, 'copy', letter)]
6    return select_action(actions)
7
8  def mutate(g⊙: Genome):
9    g🔧 = Genome('') # start from empty genome
10   i = 0
11   while i < |g⊙|:
12     action, letter = mutate_letter(g⊙[i])
13     # record the mutations
14     g🔧.mutations += [(Pos(g⊙, i), action, letter)]
15     if action != 'del':
16       g🔧 += letter # append letter to genome
17     if action != 'ins':
18       i += 1
19   return g🔧
```

Listing 1. Code for mutating a core genome.

```
1  def read_letter(letter):
2    actions = [(1 - p_error, 'read', letter)] + \
3             [(p_error / 3, 'error', l) for l in letters if l != letter]
4    return select_action(actions)
5
6  def read_segment(s: Segment):
7    for i in range(|s|):
8      action, letter = read_letter(s[i])
9      s.errors += [(Pos(s, i), action, letter)]
10     s[i] = letter
11   return s
```

Listing 2. Code for reading a sequence.

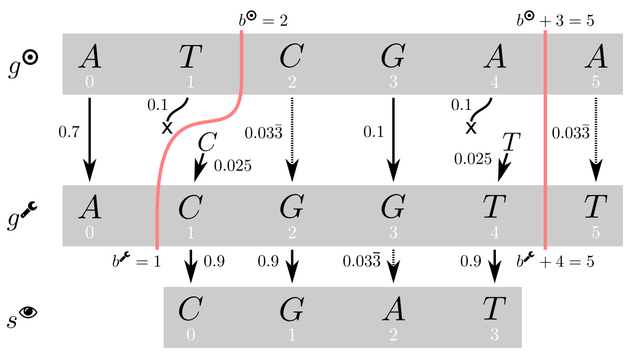


Fig. 2. Generation of a read $s^{👁} = CGAT$ from core genome $g^{\odot} = ATCGAA$ by editing and sequencing. Arrows indicate copied letters ($p_{\text{copy}} = 0.7$) or edited (dashed arrow; $p_{\text{ed}} = 0.1$), inserted (arrow starting from a new letter; $p_{\text{ins}} = 0.1$), or deleted (arrow to x; $p_{\text{del}} = 0.1$). During sequencing, we assume a phred probability of $\text{phred}_i = 0.9$ at every position i.

Lst. 1 shows an algorithmic description of this mutation process. It iterates over the positions `i` in $\mathcal{G}^{\odot}$ and and in every step determines all possible actions (Lin. 2–6). Each action consists of a probability for that action, an action type (insertion, deletion, edit, copy), and the letter produced by this action (ε indicates no letter, for deletions). Unless the action is a deletion, Lst. 1 appends the new letter to the mutated genome (Lin. 15–16). Then, unless the action is an insertion, Lst. 1 advances the read pointer (Lin. 17–18).

For simplicity of presentation, Lst. 1 assumes that each action type occurs with a constant probability at every position. In addition, it assumes that the new letter for insertions and edits is selected uniformly at random from all letters, i.e., each letter has a probability of $p_{\text{ins}}/4$ to be inserted. Likewise, the process selects the new letter for edits uniformly at random from all letters different from the current one, e.g., copying letter A yields (edited) letter C, G, or T with a probability of $p_{\text{ed}}/3$, respectively. It is straightforward to extend Lst. 1 to allow for position-dependent action probabilities, and for a selection of new letters according to a different distribution.

In Lin. 19, Lst. 1 does not only return the mutated genome $g^{🔧}$, but also its mutation history `mutations`, constructed in Lin. 14. For every step in the mutation process, the history records (i) the current position of the read pointer consisting of $g^{\odot}$ and `i`, (ii) the action type (`action`), and (iii) the generated letter (`letter`).

Sequencing. After determining $g^{🔧}$, our model sequences it according to the following steps (these steps formalize the sequencing process described in the Complete Genomes Model).

First, it selects a starting point b in $g^{🔧}$ by sampling from a random variable B describing the distribution over all possible starting points. To avoid reads of length less than L, B must never select one of the last $L-1$ positions of g.

Second, our model determines the segment $s := g[b{:}b+L]$. We note that unlike the Complete Genomes Model, our model does not produce longer segments than L at this stage. Since the read only depends on the first L letters of s, this simplification does not affect the behavior of our model.

Finally, our model reads s letter by letter in order to produce $s^{👁}$. We provide an algorithmic description of the read process in Lst. 2. For every letter, the model may (i) read it correctly with probability `1-p_error`, or (ii) read it incorrectly with probability `p_error`. Analogously to the mutation process, for simplicity of presentation, Lst. 2 assumes constant error probabilities and a selects erroneous letters uniformly at random. Likewise, Lin. 11 does not only return the sequenced read $s^{👁}$, but also its read error history `errors`, constructed in Lin. 9. For every read letter, the history records (i) the current position of the read pointer, (ii) the action type, and (iii) the generated letter.

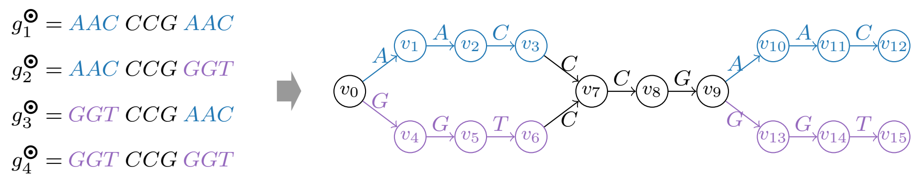
Example. Fig. 2 shows the full process of generating $s^{👁}$ on a concrete example. The model first selects a core genome $g^{\odot} = ATCGAA$. Then, it copies $g^{\odot}$ letter by letter, generating the mutated genome $g^{🔧} = ACGGTT$. The numbers in Fig. 2 indicate the probabilities of each action. Overall, the probability of generating $g^{🔧}$ from $g^{\odot}$, according to the depicted actions, is $0.7 \cdot 0.1 \cdot 0.025 \cdot 0.03\bar{3} \cdot 0.1 \cdot 0.1 \cdot 0.025 \cdot 0.03\bar{3} \approx 5 \cdot 10^{-10}$. We note there are other actions that can also generate $g^{🔧}$ from $g^{\odot}$, e.g., instead of editing C to G, the model could delete C and insert G.

After generating $g^{🔧}$, the model selects a starting position $b^{🔧} = 1$ and copies the $L = 4$ letters $g^{🔧}[1{:}5]$ to generate $s^{👁}$. The probability of generating $s^{👁}$ from $g^{🔧}$ is $0.9 \cdot 0.9 \cdot 0.03\bar{3} \cdot 0.9 \approx 2.4 \cdot 10^{-2}$.

3.2 Alignment as an Inference Task

In this section, we define the alignment task as an inference task in the generative model introduced in §3.1. Our generative model induces a genome alignment task: Given a read $s^{👁}$, determine its generation history consisting of the mutation history and the read error history leading to $s^{👁}$.

Generally, given the mutation history and the read $s^{👁}$, the read error history can be reconstructed. In addition, the mutation history describes the generation of the full $g^{🔧}$, even though $s^{👁}$ may only provide information about the generation of $g^{🔧}[b{:}b+L]$. Therefore, we define the alignment a of $s^{👁}$ to be its mutation history, restricted to positions $b^{🔧}$ to $b^{🔧} + L$.



$g_1^{\odot} = AAC\ CCG\ AAC$
$g_2^{\odot} = AAC\ CCG\ GGT$
$g_3^{\odot} = GGT\ CCG\ AAC$
$g_4^{\odot} = GGT\ CCG\ GGT$
v_0
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}
v_{13}
v_{14}
v_{15}
A
C
G
T

Fig. 3. Variation graph capturing genomes $\{g_1, g_2, g_3, g_4\}$.

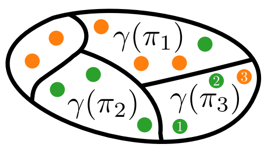


Fig. 4. Concretization of paths of length L.

Example. In Fig. 2, alignment only considers the mutations occurring between the red lines. Concretely, Fig. 2 induces alignment a, given by

$$(g^{\odot}[\![2]\!], \text{ins}, C), (g^{\odot}[\![2]\!], \text{edit}, G), (g^{\odot}[\![3]\!], \text{copy}, G),$$
$$(g^{\odot}[\![4]\!], \text{del}, \varepsilon), (g^{\odot}[\![5]\!], \text{ins}, T).$$

We note that a does not include the deletion of T, as it occurred before generating the first letter from $s^{👁}$. Likewise, it does not include deletions that occur after generating the last letter from $s^{👁}$.

MAP Alignment. Given only $s^{👁}$ from Fig. 2, there are more likely alignments than a. For example, instead of deleting a from $g^{\odot}$ and inserting T into $g^{🔧}$, editing A to T would be a more likely explanation. Generally, we can not hope to always identify the true alignment of a read, but we can determine the most likely alignment $a^\star$, given the read $s^{👁}$. This corresponds to Maximum a Posteriori Probability (MAP) estimation, and can be written as:

$$a^\star = \arg\max_a \Pr\left[A = a \mid S^{👁} = s^{👁}\right] \qquad (1)$$

In Eq. (1), we are using random variables A and $S^{👁}$ which are correlated: A is the alignment of $S^{👁}$.

4 Graph Model

In this section, we introduce a graph representation of the core genomes that enables us to efficiently solve the inference problem introduced in §3.2. The third column in Fig. 7 shows the resulting generative model.

Variation graphs provide a succinct encoding of all possible segments of core genomes. Conventional variation graphs have shown great promise in capturing large-scale variations that linear references cannot represent [1, 32, 12]. Here, we generalize these variation graphs to our probabilistic setting.

Variation Graphs. Fig. 3 shows a set of genomes $\mathcal{G}$ (left) encoded by a conventional variation graph $\mathbb{G}$ (right). The graph $\mathbb{G} = (V, E)$ consists of vertices V and edges E and is equipped with edge labels $\ell\colon E \to \{A, C, G, T\}$. Thus, each path $\pi = e_1, \dots, e_n$ in $\mathbb{G}$ spells a genome segment by $\ell(\pi) = \ell(e_1)\cdots\ell(e_n)$, where $\cdot$ denotes string concatenation.

In addition, $\mathbb{G}$ is equipped with a position map $m\colon E \to \mathcal{P}(\mathcal{G} \times \mathbb{N})$ which annotates each edge with a set of positions in the genomes. For example, $m((v_0, v_1)) = \{g_1[\![0]\!], g_2[\![0]\!]\}$, as the edge between v_0 and v_1 represents the first letter of the first two genomes. In practice, we may not know the positions referred to by each vertex v. In this case, we simply annotate v with basic information about its position, such as its approximate location in a reference genome.

When given a read $s^{👁}$, existing graph alignment tools aim to find a path π in $\mathbb{G}$ that spells $s^{👁}$, or a slightly edited version of it. The annotations on the graph's vertices then allow us to identify the (set of) positions $s^{👁}$ most likely originates from.

Concrezitation of Paths. Every path π in $\mathbb{G}$ represents one or multiple segments of genomes. We capture these segments by a concretization function γ taking as argument a path π consisting of edges e_i. Given π, γ returns the set of all possible segments π may represent, where each segment is encoded as a sequence of positions, determined by the position map m:

$$\gamma(e_1, \dots, e_L) = \{p_1, \dots, p_L \mid p_i \in m(e_i)\}$$

Compression by Variantion Graphs. Fig. 4 visualizes the concretization of paths of length L. Green circles (●) represent segments present in $\mathcal{G}$, while orange circles (●) represent spurious segments. Fig. 4 shows that the concretization of every path of length L contains at least one segment from $\mathcal{G}$. This is necessary when aligning reads of length L since otherwise, alignment on $\mathbb{G}$ may select a spurious path that does not reflect an existing genome segment. Likewise, Fig. 4 does not show any genome segment of length L that is not captured by concretizing some path. Otherwise, some reads of length L could never be aligned correctly in $\mathbb{G}$.

When $\mathbb{G}$ captures these two properties for every length $L' < L$, we say $\mathbb{G}$ *non-deterministically L-compresses* the set of genomes $\mathcal{G}$. This ensures that (i) any path selected by alignment can be interpreted in the genomes and (ii) any read can be aligned on $\mathbb{G}$. The formal definition is that for all lengths $L' \leq L$:

$$\forall \pi \in \text{Paths}_{L'}(\mathbb{G}).\exists g \in \mathcal{G}, b < |g| - L'.g[\![b{:}b + L']\!] \in \gamma(\pi) \qquad (2)$$
$$\forall g \in \mathcal{G}, b < |g| - L'.\exists \pi \in \text{Paths}_{L'}(\mathbb{G}).g[\![b{:}b + L']\!] \in \gamma(\pi) \qquad (3)$$

Probabilistic Variation Graphs. In order to account for our generative model, we generalize variation graphs to a probabilistic setting. To this end, we extend $\mathbb{G}$ by transition probabilities $p\colon E \to [0, 1]$ which must induce a probability distribution over all outgoing edges, i.e., for all $v \in V$, $\sum_{(v,v')\in E} p(v, v') = 1$. Then, the probability of selecting π is given by $p(\pi) = \prod_{i=1}^{n} p(e_i)$.

In addition, we mark a node $b \in V$ as a distinguished starting node. Then, all paths $\pi = e_1, \dots, e_n$ in $\mathbb{G}$ must start from b, i.e., $e_1 = (b, v)$ for some v.

Compression by Probabilistic Variation Graphs. To enable alignment on probabilistic variation graphs, it does not suffice if $\mathbb{G}$ non-deterministically L compresses $\mathcal{G}$. Instead, we must additionally ensure that $\mathbb{G}$ correctly captures the distribution of G. For example, in Fig. 4, we require that

$$p(\pi_3) = \Pr\left[G[\![B{:}B + L']\!] \in \{❶, ❷, ❸\}\right].$$

Formally, a probabilistic variation graph $\mathbb{G}$ *(probabilistically) L-compresses* a distribution over genomes G if the probability of every path π of length $L' \leq L$ in $\mathbb{G}$ matches the probability of the concretization of this path:

$$p(\pi) = \Pr\left[G[\![B{:}B + L']\!] \in \gamma(\pi)\right] \qquad \forall \pi \in \text{Paths}_{L'}(\mathbb{G}) \qquad (4)$$

Theorem 4.1 states that probabilistic L-compression indeed generalizes deterministic L-compression to the probabilistic setting.

```
1  def generate_read(𝔾: Graph):
2    v = 𝔾.start
3    s👁 = Segment('')
4
5    while len(s👁) < L:
6      # generate genome
7      edge = 𝔾.pick_edge(v)
8      # mutate genome
9      action, letter = mutate_letter(edge.letter)
10     s👁.mutations += [(edge, action, letter)]
11     if action != 'del':
12       # read letter
13       action, letter = read_letter(letter)
14       s👁.errors += [(edge, action, letter)]
15       # append letter to read
16       s👁 += letter
17     if action != 'ins':
18       v = edge.t
19   return s👁
```

Listing 3. Code for generating a read using the graph model.

Lemma 4.1. *Assume $\mathbb{G}$ does not contain edges that may never be used, i.e., assume $\forall e \in E.p(e) > 0$. If $\mathbb{G}$ probabilistically L-compresses a distribution over genomes G, then $\mathbb{G}$ non-deterministically L-compresses the underlying set of genomes $\mathcal{G}$.*

Proof. Let $L' < L$.

Showing Eq. (2): Let $\pi = e_1, \dots, e_n \in \text{Paths}_{L'}(\mathbb{G})$. Because $p(\pi) > 0$ (otherwise there would be an edge that may never be used), and due to Eq. (4), $\Pr\left[G[\![B{:}B + L']\!] \in \gamma(\pi)\right] > 0$. Therefore, there must be at least one g, b with $G[\![B{:}B + L']\!] \in \gamma(\pi)$.

Showing Eq. (3): Due to Eq. (4), and because the transition probabilities p induce a probability distribution, we have

$$\sum_{\pi \in \text{Paths}_{L'}(\mathbb{G})} \Pr\left[G[\![B{:}B + L']\!] \in \gamma(\pi)\right] = \sum_{\pi \in \text{Paths}_{L'}(\mathbb{G})} p(\pi) = 1.$$

Therefore, we know that all genome sequences must be covered by paths in $\mathbb{G}$.

Approximate Compression. In practice, we can only approximately satisfy Eq. (4). FiXme: Extend probabilistic L-compression to its approximate version

4.1 Generative Graph Model

In Lst. 3, we show an algorithmic description of our generative model. It starts from an empty read (Lin. 3) and extends it until it reaches L letters (Lin. 3). In each iteration, it samples a new edge in the graph (Lin. 7), possibly mutates the letter associated with this edge (Lin. 9) and records the resulting action (Lin. 10). Unless the mutation (Lin. 9) results in a deletion, the model reads the resulting letter (Lin. 13) possibly resulting in a read error, and appends the resulting letter to the read. In addition, unless the mutation (Lin. 9) results in an insertion (in which case the next letter to be read remains the same), Lin. 18 updates the current position in the graph.

4.2 Equivalence of Graph Model and Core Genomes Model

FiXme: Discuss and prove

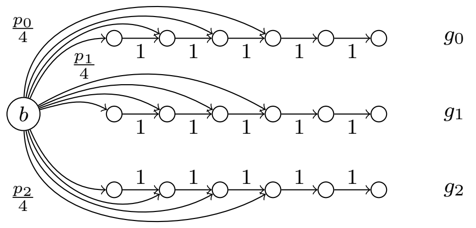


Fig. 5. General construction of a variance graph from core genomes. Each outgoing edge of b jumping into the i^{th} genome g_i has transition probability $\frac{p_i}{4}$. We do not allow jumping to the end of the genomes, as we must be able to read sequences of length L (here, we depict $L = 3$).

5 Old Graph Model

Instead of selecting an entire core genome $g^{\odot}$, we generate a core segment $s^{\odot}$ of L letters that follows the same distribution as (i) sampling a core genome $g^{\odot}$ and (ii) emitting L letters from a random starting point in $g^{\odot}$.

To select the core sequence $S^{\odot}$, we sample from a variation graph, which provides a succinct representation of genomes in $\mathcal{G}^{\odot}$. A variation graph $H = (V, E)$ consists of vertices V and directed edges $E \subseteq V \times V$ and is equipped with edge labels $l\colon E \to \{A, C, G, T\}$, transition probabilities $p\colon E \to [0, 1]$, and a distinguished starting node $b \in V$. We assume that the probabilities p induce a probability distribution over all outgoing edges, i.e., for all $v \in V$, $\sum_{(v,v') \in E} p(v, v') = 1$. Starting from b, we can sample a random walk $p = e_1, \dots, e_n$ in H (where $e_1 = (b, v)$ for some v), whose probability is given by $\prod_{i=1}^{n} p(e_i)$. Every path p spells part of a reference genome by $l(e_1) \cdots l(e_n)$, where $\cdot$ denotes string concatenation. Fig. 5 shows a general construction, which builds a variance graph H from a set of core genomes $\mathcal{G}^{\odot}$ by $|\mathcal{G}^{\odot}|$ separate paths, each spelling one core genome, and a starting node that can jump to any location of any genome in $\mathcal{G}^{\odot}$, as long as this location still allows spelling a sequence of length L. In practice, we will aim for a more compact representation of H. FiXme: reference graph building section

To mutate the core sequence $S^{\odot}$, we assume a process that copies $S^{\odot}$ letter by letter, generating a mutated sequence $S^{🔧}$. When the process is about to copy the i^{th} letter of $S^{\odot}$, it probabilistically decides to (i) insert a new letter (insertion; with probability p_{ins}), (ii) skip this letter (deletion; p_{del}), (iii) copy the letter incorrectly (edit; p_{ed}), or (iv) copy the letter correctly (copy; $p_{\text{copy}} = 1 - p_{\text{ins}} - p_{\text{del}} - p_{\text{ed}}$). We assume that the new letter for edits is selected uniformly at random from all letters different from the current one, e.g., copying letter A yields (edited) letter C, G, or T with a probability of $\frac{p_{\text{ed}}}{3}$, respectively.

Finally, to simulate the sequencing, we generate a read by copying $S^{🔧}$ letter by letter. During copying, we may misread each letter, editing the i^{th} letter of $S^{🔧}$ with probability phred_i, which depends on the letter's position i.

In Appendix 1, we discuss the equivalence of the Core Genomes Model with the Graph Model. We demonstrate that the distributions induced by both models are approximately the same, with different behavior only towards the end of the genome FiXme: what happens here exactly? .

Including Mutations in the Graph Model. The fourth column in Fig. 7 transforms the variant graph H to a new variant graph H' that directly generates mutated genome sequences. This is desirable, as it means that our inference algorithm does not need to handle mutations explicitly, because we may handle mutations by incorporating them into the variant graph directly.

Fig. 6 shows this transformation, which replaces every edge e in H by 9 edges (4 for insertions, 1 for deletions, 3 for edits, and 1 for copy).

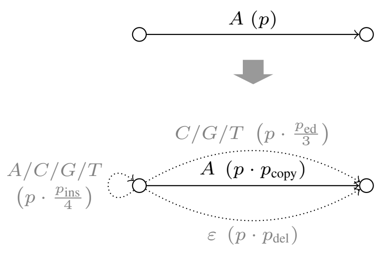

$A\ (p)$
$C/G/T\ (p \cdot \frac{p_{\text{ed}}}{3})$
$A/C/G/T\ (p \cdot \frac{p_{\text{ins}}}{4})$
$A\ (p \cdot p_{\text{copy}})$
$\varepsilon\ (p \cdot p_{\text{del}})$

Fig. 6. Enhancing H to account for mutations. For illustration purposes, H only consists of a single edge emitting letter A with probability p. We list multiple letters to indicate multiple edges (e.g., $A/C/G/T$).

Insertions produce a new inserted letter without reading a letter from the core genome. Therefore, H' adds 4 edges (each with probability $p \cdot \frac{p_{\text{ins}}}{4}$) that allow emitting a letter without changing the current state. Deletions allow reading a letter from the core genome without copying it. Therefore, H' adds an edge that does not emit a letter (it is labeled by ε) but changes the state. Edits allow reading a letter from the core genome but copying a different letter. Therefore, H' adds 3 edges (each with probability $p \cdot \frac{p_{\text{ed}}}{3}$) that emit a letter but change the state is if emitting the original letter.

6 Problem statement

6.1 Biological generative model

Here we suggest a model of the biological process such that given a set of reference genomes G produces a query read (with phred values). There are no assumptions on the genomes and their (dis-)similarities.

Let $\{G_i\}$ be a set of N given haploid genomes each with an associated occurrence probability $P_{occurence}(G_i)$. The large-scale variation between the genomes is assumed to be captured by G being representative for different genome classes depending on the task.

Let there exists an infinite universe of genomes $U \supset G$ such that each genome has a defined probability to be observed: $P^{g \in G}_{mutate}(g' \in U) \sim 1/d_{edit}(g, g')$. The edit distance $d_{edit}(\cdot, \cdot)$ function accounts for edit operations (insertions, deletion and substitutions) that are assumed to be a reasonable model for small-scale evolutional variation. The edit distance metric can account distinguish between different edit type or specific letter transformation.

Let P_{start} be the probability of a read starting at position $start$ in the reference genome g (note that the length may be slightly different than $\hat{g}$). Let there be a probabilistic process that introduces substitution changes to a read sequence. The substitution probabilities $phred_i$ are known for each read position i.

Generative model FiXme: figure: First, a genome g is picked from G according to $P_{occurence}$. Then g is mutated into $\hat{g}$ according to P_{mutate}. Then a set of reads is sampled from $\hat{g}$: each fixed-length read r is picked as a substring of $\hat{g}$ according to P^g_{start}. Finally, according to P_{phred} technical errors are introduced to produce the read with errors $\hat{r}$. The generated set of reads is $\hat{R} = \{(\hat{r}, phred)\}$.

6.2 Biological inference

We formalize the problem of *mapping* a set of reads (with phred values) to a set of genomes (or any structure M capturing the information of a set of genomes). The goal is to find such a mapping of the reads that maximizes the probability of the set of reads being generated by the model. In other words,

$$map_\theta(\hat{r}) = \underset{\hat{g} \in U \wedge pos}{\arg\max}\, P_{occ}(g) P^g_{mut}(\hat{g}) P^g_{start}(pos) P_{ph}(\hat{r}, r)$$

where

$$\theta = \langle P_{occ}, P_{mut}, P_{start}, P_{ph} \rangle$$
$$G - \text{is the finite set of the given reference genomes}$$
$$U - \text{is the infinite set of all possible genomes}$$
$$g = argmin_{g \in G} d_{edit}(\hat{g}, g),$$
$$r = g[pos : pos + len]$$

6.3 Boundary mapping conditions

We postulate a set of properties that a mapping framework should preserve when combining different sources of uncertainty. As a verification of our approach, we impose the following boundary conditions where our mapping optimization task has to be equivalent to existing well known optimization criteria that are recognized by the community:

- Assuming perfect phred values (all equal to 1, i.e. all bases are correctly read), no edit operations (no edit edges are added to the graph) and no discriminative variation probabilities (all equal to 1), the mapping task is equivalent to **exact mapping** (e.g. equivalen to vg[1] in the exact regime).
- Assuming perfect phred values and no edit operations: The graph accounts only for variation probabilities and the mapping is equivalent to **maximizing the path probability** generated by our model (by definition).
- Assuming perfect phred values and no discriminative variation probabilities: Mapping on such graph allows for edit operations and the mapping procedure **minimizes the edit distance**.
- Assuming no edit operations and no discriminative variation probabilities: Mapping is accounting only for the technical errors (i.e. phred values) and it is equivalent to **maximizing the probability that the query as a generative model will generate the path** (obviously, the path should be the same length as the query as indels are not permitted).

We show that our optimization task indeed satisfies all the stated boundary conditions and continuously and monotonically extrapolates between them. FiXme: justify, prove

After generating a sequence, technical noise is added according to a given phred profile.

7 Graph model (old)

We will use notations similar to [33]. Let a PNFA be 5-tuple $M = (Q, \Sigma, \delta, \pi, q_0)$ where Q is a finite set of states, Σ is a finite alphabet of symbols, $\delta \subseteq Q \times \Sigma \times Q$ is the transition function (note that as the automata is non-deterministic, the same state/symbol pair may repeat). For brevity, we will use $l(e) := label$ where $e \in Q \times \{label\} \times Q$, $to(e) := v$ where $e \in Q \times \Sigma \times \{v\}$. $\pi : e \to [0, 1]$ is the probability of transiting through the edge $e \in \delta$. q_0 is the initial superstate.

7.1 Sequence generation by a graph reference

Given the PNFA G and a path length L, we will generate a sequence with length L doing a random walk in G. If for every node, there is an outgoing non-ε edge with positive probability, the algorithm terminates

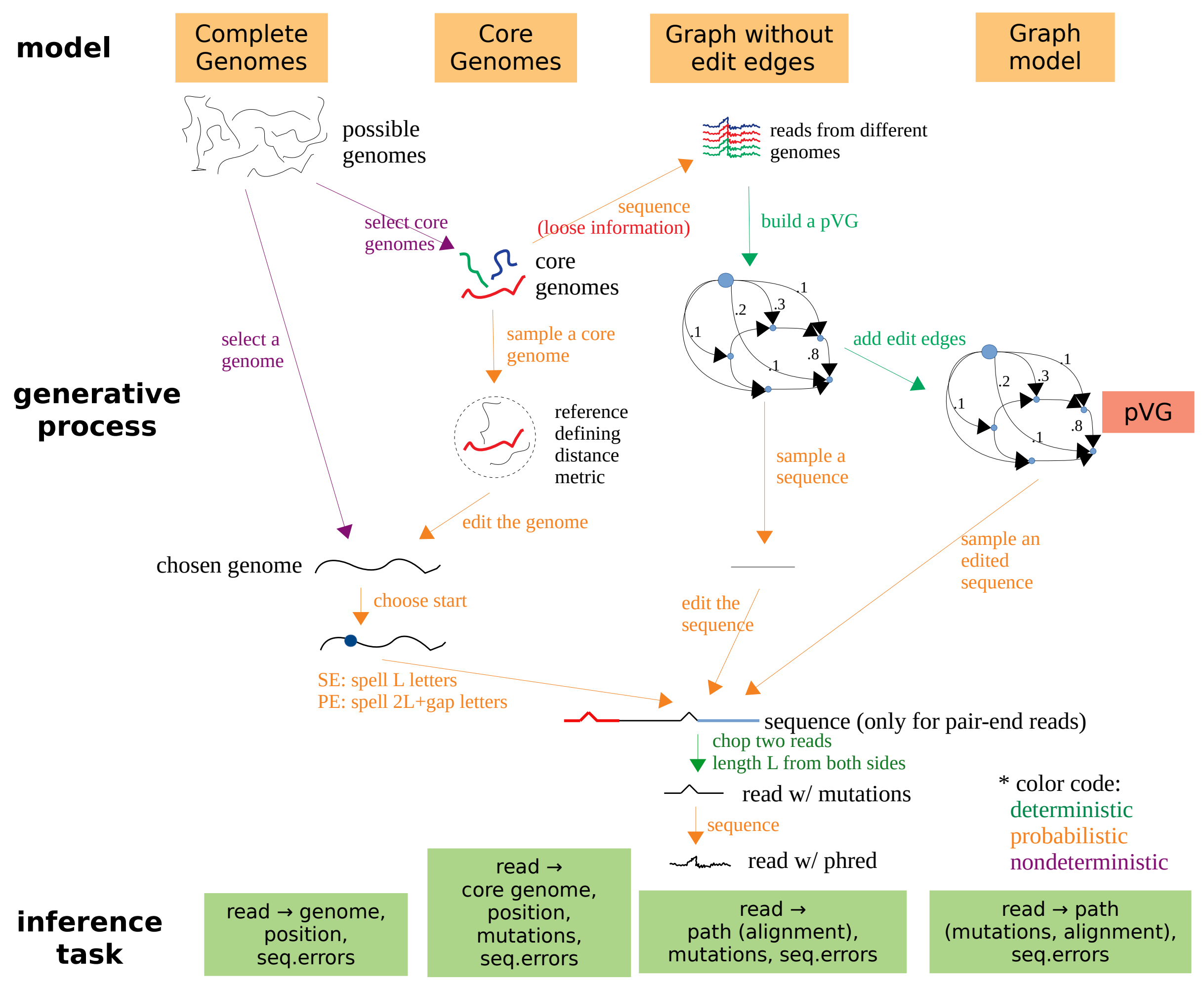


Fig. 7. Generative models and inference problems. Columns represent models. Colors capture the type of transitions: probabilistic, deterministic or nondeterministic

with probability 1. In order to generate a read, the generated sequence can be modified using additional phred values.

7.2 Inference on graphs

Definitions:

$\hat{r}$ — query read with base-calling error probabilities $P(bp_i \in \hat{r})$ G — PNFA (with ε edges)

$spell(path) = path$ with removed ε-letters

$score_G(e) = -10\log_{10} P_G(e)$ — phred-like score for an edge

$phred(r_i) = -10\log_{10} P(r_i)$ — phred value for a base pair in the read r

$P(r_i) = 10^{-phred(r_i)/10}$ — base-calling error probability

Optimization task in terms of probabilities:

$$P(path \mid G) = \prod_{e \in edges(path)} P_G(e)$$

$$P(spell \mid \hat{r}) = \prod_{spell_i = \hat{r_i}} (1 - P(\hat{r_i})) \prod_{spell_i != \hat{r_i}} \frac{1}{3} P(\hat{r_i})$$

FiXme: MAQ uses$P(spell \mid \hat{r}) = \prod_{spell_i != \hat{r_i}} P(\hat{r_i})$

$$P(path \mid \hat{r}) = P(path \mid G) P(spell(path) \mid \hat{r})$$

$$map_G(\hat{r}) = \arg\max_{path:|spell(path)|=|\hat{r}|} P(path|\hat{r})$$

$$= \prod_{e \in edges(path)} \mathbf{P_G(e)} \prod_{spell_i = \hat{r_i}} (\mathbf{1} - \mathbf{P}(\hat{\mathbf{r}}_\mathbf{i})) \prod_{spell_i != \hat{r_i}} \frac{\mathbf{1}}{\mathbf{3}} \mathbf{P}(\hat{\mathbf{r}}_\mathbf{i})$$

Any mapping probability $P(path \mid \hat{r})$ can be deconstructed into three components corresponding to read sequencing errors, variant edges and edit edges. We transform all probabilities to phred-like values to formulate an equivalent task (details in the Supplementary).

$$score(path \mid \hat{r}) = \sum_{e \in edges(path)} \mathbf{score_G(e)} + \sum_{spell_i = \hat{r_i}} \mathbf{-10 \log_{10}(1 - P(\hat{r_i}))} + \sum_{spell_i != \hat{r_i}} \mathbf{\frac{1}{3} phred(\hat{r_i})}$$

$$map_G(\hat{r}) \equiv \arg\min_{path:|spell(path)|=|\hat{r}|} score(path|\hat{r})$$

FiXme: compare with the “correct alignment probability” by FiXme: add citation: li2008mapping FiXme: prior on starting vertices “probability p(z|x,u) of z coming from the position u equals the product of the error probabilities of the mismatched bases at the aligned position.” FiXme: add citation: li2008mapping

7.3 Length-normalized alignment probability

FiXme: Is there a parallel with the generative model? In order to compensate for the monotonically decreasing probability with read length, we propose a length-normalized probability. It can be used as an intuitive alignment quality score, namely “the average alignment probability per read letter”, as a threshold parameter (substituting for the usual edit distance) together with a minimal alignment length parameter (above), or as an optimization criteria (algorithm presented next).

If a read with length L aligns with an (additive) score s, the normalized alignment score is $\bar{s} = s/L$. Thus (using definition), the normalized (multiplicative) probability becomes $\bar{p} = p^{1/L}$ where $p = 10^{-10s}$ (obviously, $\bar{p} \in [0, 1]$ as well). Theorem: After the Dijkstra algorithm finds an optimal alignment, the optimal alignments for all prefixes of the read have been inserted to the queue. Proof: From the contrary. FiXme: Formal proof. Thus, during the Dijkstra algorithm, we can keep the optimal subsolutions (for each read prefix) and then normalize each of them by the corresponding length.

In order to prioritize longer alignments, the alignment score should also include a length-dependent member FiXme: Can this be explained by the generative model?. Naturally, a probability factor of $0.25^{L-aligned}$ that punishes short alignments follows a generative process that uniformly at random assignes a nucleotide to each read position that has been read but happens to be after the known reference region. This factor can be extended to depend on the read letters that remained unaligned: for example, it can fit more closely to a known distribution of nucleotides in any genome.

7.4 Partial read alignment

The work [21] uses a sequence of dummy vertics in order to accomodate for a query prefix that is not included in the graph.

It may be necessary to align only part of the read to the reference (e.g., because of adapters on the read or because of the read covering genome parts that are not covered by the reference). We call an alignment *partial* if it is allowed to skip the alignmening of variable-length prefix and/or suffix of the read. We call *best partial alignment* a partial alignment that maximizes the length-normalized probability of the alignment. The best partial alignment can be seen as a generalization of MEMs that is widely used (Table 1). We call *a best partial alignment through a pivot* we call a best partial alignment that includes a certain read position called *pivot*.

To find a best partial alignment through a pivot position, we align the read part to the left (r_l) and to the right (r_r) of it. The right part we can align using the same procedure for mapping the whole read and the left part can be aligned starting from the pivot and going backwards through the reverse edges.

Theorem: After optimally solving the global task using Dijkstra, all local solutions (for all prefixes) will also be found as subsolutions. Proof: Obvious. FiXme: Formal proof.

Theorem: Solving the reverse tasks for r_l (iterating the read to the left of the pivot and using the reverse graph edges) finds the same shortest paths as solving the forward tasks for all suffixes of r_l. Proof: Obvious. FiXme: Formal proof.

Theorem: If the optimal solutions of the two parts align the pivot of the read to the same vertex, then the concatenation of the subtasks solve the whole task optimally.

A possible strategy for choosing the pivot could be the middle of the read, or trying several pivots at different parts of the read. Obviously, a minimal aligned read length can be enforced by choosing the left and the right alignement ends some distance apart.

7.5 Gap between the biological generative model and the graph model

The biological generative model represents the distribution $p_b(y = (genome, position), x = read)$ which is the most detailed model we consider. The graph generative models represent the distribution $p_g(y = node, x = read)$ which is included in and deterministically computable by p_b ($node$ is the starting node where a read is mapped). So there exists a surjective function $f : (genome, position) \to node$ that represents the transition from genome coordinates to graph nodes. The gap to be optimized is between the distribution p_g and the the original distribution transferred to the starting nodes domain:

$$p^*_g(node, read) = \sum_{\substack{genome \wedge position: \\ f(genome, position) = node}} p_b((genome, position), read)$$

$$gap(p^*_g, p_g \mid \theta) = D_{KL}(p^*_g \mid\mid p_g(\theta))$$

where

S — the set of observed sequences

p_G — the sequence distribution of the generative graph model G

θ — the edge probabilities (incl. edit probabilities)

8 Graph model construction

The PNFA captures the probabilistic nature of large-scale variation (by the reference genomes used for construction) as well as of small-scale variation (by the edit edges). It can be used as both a generative model and a reference for mapping new reads. The PNFA accommodates for edit operations and can be uniquely constructed from a pDFA given the edit probabilities 7. Each edge in the pDFA corresponds to exactly one letter. The pDFA can be uniquely constructed from a pDBG or another string graph with edge probabilities. Interestingly, the graph structure is computed independently of the edge probabilities so all existing string graph approaches can be reused and extended. The probability of an outgoing edge is in $[0; 1]$ and all outgoing edges from a vertex may or may not form distribution depending on the desired properties.

8.1 Estimating edge probabilities

FiXme: Compare with the probabilities estimation of Partis’16[35]. FiXme: Compare with FORGe’18[34](preprint).

The learning process optimizes the edge probabilities in order to minimize the gap between the distribution graph generative model and the transformed distribution of the biological generative model.

$$\arg\min_{\theta} gap(p_g^*, p_g \mid \theta)$$

From a set of reference sequences In case the graph structure is defined by the same set of sequences which is to be used for edge probabilities optimization, then no edits are expected. In this case it is reasonable to perform the optimization on a graph model without edit edges (i.e. pDBG or pDFA). Errors in the reads are expected to be taken care of by assigning low probabilities.

The optimization task can then be solved exactly by counting. FiXme: To prove

From independent graph and a set of sequences In the most general case when the graph structure is not assumed to be build by the given set of sequences, the optimization can be performed on the PNFA model which accounts for both edits and technical errors (in case the observed sequences include phred values).

Construction from a set of reads: every kmer from the hidden reference is seen in the reads. (+ the noise is improbable to give a better alignment) FiXme: how to compute the transition probabilities (construction) and what is their benefit; how do the HMM paper solves the problem of comparing short-vs-long paths and low complexity regions vs high complexity regions [35]

From a reference sequence and a set of known variants

From a sequence graph or PRG

8.2 pDBG to pDFA

We define a probabilistic DBG (pDBG) as a generalization of a DBG associating an additional transition probability to each edges. If not stated explicitly, we assume that the outgoing probabilities from each non-terminal vertex form a distribution (sum up to 1). Naturally defined by the DBG, every transition is associated with a the rightest letter of the kmer of the receiving vertex. In order to support paths with edits, we generalize the letters by using not only nucleotides but also gap letters (ε). We can the look at the pDBG also as a PNFA which is a generalization of the discrete-time discrete-state Markov process with transition labels. Note that the PNFA is a generalization of NFA so there may be repeated outgoing letters and epsilon transition.

For every node in the pDBG, k nodes (a tower up) are added to the PNFA: for each non-empty prefix. A node for prefix length i is connected to the node for prefix length $i+1$ with the $(i+1)$st letter. The new nodes allow for edits in the first kmer of a match. Added is a supersource having epsilon transitions to all nodes corresponding to prefixes of size 1 (transitions to longer prefixes would be equivalent to transitions to towers of another kmer nodes).

Construction is trivial when probabilities per edge are given along with the graph. In case no probabilities are supplied, they are assumed to be all 1 (with equivalent scores $-log(1) = 0$). This may be tricky when mapping using DP as it induces cycles (because of deletion edit operations) that cannot be ordered correctly because of the 0 score penalties.

DBG->PNFA: Matching the first k letters as well Before aligning the query to a path, we have to align the prefix of the query to the kmers of each vertex. It is not trivial to align the query to a kmer as the best aligning prefix does not have to be the one leading to optimal alignment on the graph. Thus we are converting the dDBG to a PNFA by adding a new vertex for each of the k prefixes of each DBG vertex resulting in total of kN new vertices. Each group of k vertices are sequentially connected and finally lead to the initial kmer vertex. In order to save space, the repetitive prefixes of equal size can be further unified to form a DAG. The size of the new graph becomes $O(E + Vk)$.

8.3 Edit scores to probabilities

Additional edges can be added to the pDBG to account for edit operations without worsening the algorithms complexity. Note that under the assumption of a correct alignment (FiXme: what does this mean? The situation here is strange, as both positions tell us similar pieces of information (about the prefix). It seems more reasonable to only allow insertions/deletions/substitutions when walking around in the graph, and ignore the query for this. The situation here is strange, as both positions tell us similar pieces of information (about the prefix). It seems more reasonable to only allow insertions/deletions/substitutions when walking around in the graph, and ignore the query for this.), it is enough to condition the edit penalties (probabilities) only on the graph position and not on the query position.

Let p_i, p_d and p_s stand for insertion, deletion and substitution probabilities (for notational simplicity, we assume these probabilities are global and not dependent on the vertex; this can be trivially generalized). An edit-pDBG is defined by a pDBG and edit probabilities by adding 4(V+E) more edges and no additional vertices. The additional edges per edit operations we define as:

- insertion (append to query): +4V edges: 4 edges are added for each vertex v: (v, v, letter, $p_i/4$) for letter in nucleotides
- deletion (erase from query): +E edges: one edge per existing edge $(u, v, letter, p) \to (u, v, \varepsilon, p_d)$
- substitution: +3E parallel edges: $(u, v, oldletter, p) \to (u, v, newletter, ?/(3* outNum))$ for newletter != oldletter, FiXme: should be $p * p_s/3$: we proceed with oldletter, but it was substituted
- FiXme: transposition?

After adding all edit transitions, we normalize the outgoing probabilities for each vertex to 1. More complex operations like inversions, duplications, or highly variable copy number remain challenging.

8.4 Quantifying the mapping quality

Blast tutorial: https://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html

The alignment accuracy can be measured in global (we refer to it as mapping) and local sense (we refer to it as alignment): is the region where a read is mapped correct; and how accurate is the local alignment along the reference[10]. pVG has an accent on improving local alignment while preserving the global mapping accuracy.

“One of the computational (and modeling) challenges facing the field of pan-genomics is how to deal with data uncertainty propagation through the individual steps of analysis pipelines. To do so, the individual processing steps need to be able to take uncertain data as input and to provide a ‘level of confidence’ for the output made. This can, for instance, be done in the form of posterior probabilities. Examples where this is already common practice include read mapping qualities [155] and genotype likelihoods [156].Computing a reasonable confidence level commonly relies on weighing alternative explanations for the observed data. In the case of read mapping, for example, having an extensive list of alternative mapping locations aids in estimating the probability of the alignment being correct. A pan-genome expands the space of possible explanations and can, therefore, facilitate the construction of fairer and more informative confidence levels”[2].

Different alignment quality scores include γ-centroid (probabilistic) alignment, E-values[10] (the expected number of mapping occuring by chance: dependent only on the size of the reference and the length of the alignment). FiXme: Computing the marginalized probabilities for each position of an alignment may be needed.

FiXme: Handle multiple good mapping positions. MAQ outputs a score of 0.

The generative process can generate the same sequence by going through different paths. The mapping solution finds only the path with maximum. Any mapping in our approach is characterized by a total mapping probabiliy that represents the product of three probabilities: acounting for variant edges, edit edges and phred scores. As such a probabilitiy is heavily dependent on read length, overall read quality, reference size, etc., we suggest using another, comparatative, score as a signal for the certainty of mapping (i.e. the confidence of the mapping position while fixing the reference and the read). As a comparative score we can use the ratio between the probability of the best mapping and the probability of the second best mapping. We extend the MAQ[24] technique by applying it to graphs, allowing for edits, and apploximating more precisely. In our framework, the mapping probabilities are trivilly extracted when aligning. To make the mapping quality approximation more accurate, we find the K best mappings use sampling to evaluate the precision of our approximation. We call a mapping *unique* if the probability of alignment is above a certain threshold (in MAQ a mapping is called *unique* if the second best mapping has more mismatches than the best one).

Statistical issues with the huge alignment space.

$$\sum_{path(v_1 \xrightarrow{e_1} v_2 \xrightarrow{e_2} \ldots \xrightarrow{v_{k-1}} v_k):\ v_k = u} P(path)$$

Algorithm 1 Sum of path probabilities given a read ($\pm\varepsilon$ guarantees)

```
1: function SumOfPathProbsIntervalNoPhred($G$ : $Graph, r$ : $Read, \varepsilon > 0 : Probability$)
2:     $s \leftarrow State(i = 0, v = supersource(G), P_{pref} = 1.0)$
                                         ▷ Sorting criteria determines number of steps
3:     $Q \leftarrow PriorityQueue(\{s\}, sortby = P_{pref})$
4:     while $sum_{q \in Q}(P_{pref}(q) P_{best}(q)) - sum(P_{suff}(q \in Q)) > \varepsilon$ do      ▷ $|Q| > 0$
5:         $s \equiv State(i, u, p) \leftarrow Q.pop()$
6:         for all $v, label, prev\_prob \equiv e \in G.outgoing\_edges(u)$ do
7:             if $label = r[i]$ or $label = \varepsilon$ then
8:                 $Q.add(State(i, v, prev\_prob * prob(e)))$
9:     return $(lo = sum(P_{suff}(q \in Q), up = sum(P_{pref}(q \in Q)))$

10: function PathCertainty($best\_path: Path, G: Graph, r: Read, \delta = 0.05: Accuracy$)
11:     $best\_prob \leftarrow PathProb(best\_path, r)$
12:     $\varepsilon \leftarrow \delta * best\_prob$
13:     $sum\_prob \leftarrow SumOfPathProbsInterval(G, r, \varepsilon)$
14:     return $(lo = best\_prob/up(sum\_prob), up = best\_prob/lo(sum\_prob))$
```

9 Inference

9.1 Sampling

Algorithm for sampling a read from a reference graph.

10 Results

TODO table: region (small and complex), real use case, realistic assumpitions, alignment specific measurements, feasible, strengths (graph, edits, phreds)

we will evaluate our tool in two phases, where the first phase is directly evaluating the alignment accuracy of our approach, and the second phase leverages our alignment to solve the high-level task of VDJ annotation (from a read, identifying which regions are from which variant of V, D, and J)

1. the first phase will be a validation of our approach with simulated data. for the graph, we will take a VDJ graph (possibly with transition probabilities taken from the second phase), and manually picked mutation and phred probabilities. we will then evaluate the precision of the alignment, against the ground truth (which we have because this is a simulation). as a baseline, we will also try edit distance. we may also turn mutations and phred values on&off to see the effect of this 2. the second phase will be on the high-level application of VDJ annotation (from a read, identifying which regions are from which variant of V, D, and J). this has been done before (in iGOR and partis), and partis has a simulator we can leverage. based on this simulator, we construct a graph by (i) manually building a graph skeleton and (ii) inferring transition probabilities by relative frequencies of simulated reads. then, we evaluate by (i) simulating a read (from the existing simulator), (ii) adding noise (edits + phreds), (iii) aligning the resulting read (using the graph from before) and (iv) annotating the aligned read.

We follow the testing protocol of [13]. They have asked different groups to generate variant graphs. Then they simulated reads from these graphs, mapped them onto the graphs and measured the mapping accuracy. We reuse the structure of these graphs and we additionally generate transition and edit probabilities for the graphs as well as phred probabilities for the reads. For simplicity and generality, we use uniform distributions in chosen intervals instead of fine-tuned read simulators for specific technologies.

10.1 Experiments

To analyse the importance and behaviour of different mapping modes in our framework, we independently test all combinations of uncertainty sources. According to 1 there are 7 modes (defined by the activated sources of undertainty):

1. *Variants* – Mapping non-probabilistic reads to a graph reference without making edits (equivalent to the exact mapping in vg).
2. *Edits* – Mapping non-probabilistic reads to a linear reference with edit allowed (equivalent to [36]).
3. *Phred* – Mapping probabilistic reads to a linear reference without making edits (MAQ [24], FiXme: Check MAQ).
4. *Variants + Edits* – Mapping non-probabilistic reads to a graph reference with edits allowed (FiXme: Is this possible in vg?).
5. *Variants + Phred* –
6. *Edits + Phred* –
7. *Variants + Edits + Phred* –

First try: measure only the percentage of correct alignments partis VDJ: goals – 1) increase the D gene prediction accuracy (because of another construction; because of phreds; because of indels), 2) report sub-allele-level of annotation, 4), 3) ideally, report a mapping score

Evals scheme: generate TCR/BCER sequences using the simulator from IGoR'18[28], (2) align using SW and pVG

Cross-align: reads from one specie on graph build from another specie(es).

Plots correctly mapped reads vs incorrectly mapped reads[10].

sensitivity, p-value (FiXme: Mentioned in [10]. What would be the null hypothesis?), E-value

Variants, Mendelian accuracy[8]

For both graph construction and read sampling we assume that the genomes follow to a geometric distribution (similarly to [6]).

Compare paths

10.2 Datasets

opensnp.org ? Human genome?, MHC, HLA reference alleles were fetched from the IPD-IMGT/HLA database[37] (analoguous experimet to Graphtyper[8]), TCR/BCR

10.3 Criteria

"in the evaluation, a mapped read is considered to be correct if it overlaps the true mapping (we call it 'Mapping accuracy'). This means that the detailed alignment between the mapped read and the reference genome, for example, 'Aligned column accuracy' or 'Gap accuracy', is not always correct even if the mapping is correct. It is possible that probabilistic alignment improves those accuracy measures."[16]

We measure mapping correctness, alignment accuracy and performance. We prove the output from all solutions approaches to be the same so we compare the DP, Dijkstra and A* on performance.

10.4 Results

In terms of performance, we note that the Dijkstra approach is faster than the DP but still comparable whereas the A* rulzz FiXme: verify :).

11 Discussion

11.1 Applications

PNFA generalizes vg with transition uncertainties and edit uncertainties. PNFA is also expressive enough to fully capture tools igGraph[3], partis [35] and PRG [7].

The PNFA can express any Markov model over any HMM with finite observed states (e.g. nucleotides).

High noise scenario with long reads

The HMM from the partis tool [35] can be rewritten as a PNFA:

1. The hidden states per nucleotide position that generates four output symbols translate to four vertices per symbol at that position
2. The HMM topology in the YAML DSL (VDJ hardcoded sequences with probabilities, filters, etc.) translate into PNFA: each allele translates to a different path; the VDJ hidden state annotations translates to vertex attributes (subsets of V/D/J genes as in [3])
3. probabilities for allele usage, transitions and emissions translate to edge probabilities: substitution/deletion/insertion probabilities
4. time complexity drops

11.2 Future work

- Intro/Abstract/Discussion: Fronts: 1) unifying different sources of uncertainty, 2) existing preknown variant probabilities, 3) dealing with errors (both technical and biological) efficiency (how long does it take to compute the answer, how much memory does it need?) power (does it make good use of the data, or is information being wasted?) consistency (will it converge on the same answer repeatedly, if each time given different data for the same model problem?) robustness (does it cope well with violations of the assumptions of the underlying model?) falsifiability (does it alert us when it is not good to use, i.e. when assumptions are violated?)
- Evals: Multiple alignment and variant calling
- Algorithm: time and memory efficient, Seen-and-extend approach [26], minimizers, seeds
- Construction: cycles; be able to represent different genome graphs (e.g. DBGs, string graphs); independency (or at least weaker dependency) of the k parameter in DBG (because of edit distance guarantees) glocal alignment in probabilistic terms?
- reverse complementary: bidirected edges
- Technical: I/O compatible with other tools (e.g. vg), translation tool from HMM for VDJ to pVG
- Future: Stability (e.g. under edit operation costs)[15], support long indels – teleport supernodes; affine gaps (implement by parallel layers in the graph) and more specific distributions of lengths of deletions or insertions (e.g. for VDJ); capturing longer relationships in the graph (longer-range information about haplotype structure); diploid paths (chromotype), discover novel variation (not present in the graph); pair-end reads; support different variants as listed by svaha: Deletions, Inversions, Insertions, SNPs, Duplications, Transversions, Breakpoints; RNA mapping like TopHat2 Chimera-resolving Compressing the graph into bigger nodes Centroid estimation instead of solving a MAP problem[4, 16] Mapping quality invariance: mapping score, second best mapping – invariances of the prob. metric to query distribution, to query length, to additional entries in the ref. graph, visualize it with shades according to prob.

References

[1]Variant Graph tool, `https://github.com/vgteam/vg`, 2013.

[2]Computational pan-genomics: Status, promises and challenges. *Brief Bioinform*, 19(1):bbw089, Oct. 2016.

[3]S. R. Bonissone and P. A. Pevzner. Immunoglobulin classification using the colored antibody graph. In *International Conference on Research in Computational Molecular Biology*, pages 44–59. Springer, 2015.

[4]L. E. Carvalho and C. E. Lawrence. Centroid estimation in discrete high-dimensional spaces with applications in biology. *Proceedings of the National Academy of Sciences*, 105(9):3209–3214, Feb. 2008.

[5]A. Danek, S. Deorowicz, and S. Grabowski. Indexes of large genome collections on a PC. *PLoS One*, 9(10):e109384, Oct. 2014.

[6]D. C. Danko, D. Meleshko, D. Bezdan, C. Mason, and I. Hajirasouliha. Minerva: An alignment- and reference-free approach to deconvolve linked-reads for metagenomics. *Genome Res.*, 29(1):116–124, Dec. 2018.

[7]A. Dilthey, C. Cox, Z. Iqbal, M. R. Nelson, and G. McVean. Improved genome inference in the MHC using a population reference graph. *Nat Genet*, 47(6):682–688, Apr. 2015.

[8]H. P. Eggertsson, H. Jonsson, S. Kristmundsdottir, E. Hjartarson, B. Kehr, G. Masson, F. Zink, K. E. Hjorleifsson, A. Jonasdottir, A. Jonasdottir, I. Jonsdottir, D. F. Gudbjartsson, P. Melsted, K. Stefansson, and B. V. Halldorsson. Graphtyper enables population-scale genotyping using pangenome graphs. *Nat Genet*, 49(11):1654–1660, Sept. 2017.

[9]B. Ewing and P. Green. Base-calling of automated sequencer traces UsingPhred.II. error probabilities. *Genome Res.*, 8(3):186–194, Mar. 1998.

[10]M. C. Frith, M. Hamada, and P. Horton. Parameters for accurate genome alignment. *BMC Bioinf.*, 11(1):80, Feb. 2010.

[11]M. C. Frith, R. Wan, and P. Horton. Incorporating sequence quality data into alignment improves DNA read mapping. *Nucleic Acids Res.*, 38(7):e100–e100, Jan. 2010.

[12]E. Garrison, J. Sirén, A. M. Novak, G. Hickey, J. M. Eizenga, E. T. Dawson, W. Jones, S. Garg, C. Markello, M. F. Lin, B. Paten, and R. Durbin. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nat Biotechnol*, 36(9):875–879, Aug. 2018.

[13]E. Garrison, J. Sirén, A. M. Novak, G. Hickey, J. M. Eizenga, E. T. Dawson, W. Jones, S. Garg, C. Markello, M. F. Lin, B. Paten, and R. Durbin. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nat Biotechnol*, 36(9):875–879, Aug. 2018.

[14]A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

[15]D. Gusfield, K. Balasubramanian, and D. Naor. Parametric optimization of sequence alignment. *Algorithmica*, 12(4-5):312–326, Nov. 1994.

[16]M. Hamada, E. Wijaya, M. C. Frith, and K. Asai. Probabilistic alignments with quality scores: An application to short-read mapping toward accurate SNP/indel detection. *Bioinformatics*, 27(22):3085–3092, Oct. 2011.

[17]M. Heydari, G. Miclotte, Y. Van de Peer, and J. Fostier. Browniealigner: accurate alignment of illumina sequencing data to de bruijn graphs. *BMC Bioinformatics*, 19(1):311, Sep 2018.

[18]G. Holley and P. Peterlongo. BlastGraph: Intensive approximate pattern matching in string graphs and de-Bruijn graphs. In *PSC 2012*, 2012.

[19]L. Huang, V. Popic, and S. Batzoglou. Short read alignment with populations of genomes. *Bioinformatics*, 29(13):i361–i370, June 2013.

[20]W. Huang, L. Li, J. R. Myers, and G. T. Marth. Art: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2011.

[21]C. Jain, H. Zhang, Y. Gao, and S. Aluru. On the complexity of sequence to graph alignment. *bioRxiv*, page 522912, 2019.

[22]D. Kim, B. Langmead, and S. L. Salzberg. HISAT: A fast spliced aligner with low memory requirements. *Nat Methods*, 12(4):357–360, Mar. 2015.

[23]B. Langmead and S. L. Salzberg. Fast gapped-read alignment with bowtie 2. *Nat Methods*, 9(4):357–359, Mar. 2012.

[24]H. Li, J. Ruan, and R. Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res.*, 18(11):1851–1858, Nov. 2008.

[25]A. Limasset, B. Cazaux, E. Rivals, and P. Peterlongo. Read mapping on de bruijn graphs. *BMC Bioinf.*, 17(1):237, June 2016.

[26]B. Liu, H. Guo, M. Brudno, and Y. Wang. deBGA: Read alignment with de bruijn graph-based seed and extension. *Bioinformatics*, 32(21):3224–3232, July 2016.

[27]S. Maciuca, C. del Ojo Elias, G. McVean, and Z. Iqbal. A natural encoding of genetic variation in a Burrows-Wheeler Transform to enable mapping and genome inference. In *International Workshop on Algorithms in Bioinformatics*, pages 222–233. Springer, 2016.

[28]Q. Marcou, T. Mora, and A. M. Walczak. High-throughput immune repertoire analysis with IGoR. *Nat Commun*, 9(1):561, Feb. 2018.

[29]E. W. Myers Jr. A history of DNA sequence assembly. *it - Information Technology*, 58(3):126–132, Jan. 2016.

[30]J. C. Na, K. Roh, A. Apostolico, and K. Park. Alignment of biological sequences with quality scores. *IJBRA*, 5(1):97, 2009.

[31]R. Nielsen, J. S. Paul, A. Albrechtsen, and Y. S. Song. Genotype and SNP calling from next-generation sequencing data. *Nat Rev Genet*, 12(6):443–451, June 2011.

[32]B. Paten, A. M. Novak, J. M. Eizenga, and E. Garrison. Genome graphs and the evolution of genome inference. *Genome Res.*, 27(5):665–676, Mar. 2017.

[33]D. Pfau, N. Bartlett, and F. Wood. Probabilistic deterministic infinite automata. In *Advances in neural information processing systems*, pages 1930–1938, 2010.

[34]J. Pritt, N.-C. Chen, and B. Langmead. FORGe: Prioritizing variants for graph genomes. *Genome Biol*, 19(1):311720, Dec. 2018.

[35]D. K. Ralph and F. A. Matsen. Consistency of VDJ rearrangement and substitution parameters enables accurate b cell receptor sequence annotation. *PLoS Comput Biol*, 12(1):e1004409, Jan. 2016.

[36]M. Rautiainen and T. Marschall. Aligning sequences to general graphs in $O(V + mE)$ time. *bioRxiv*, page 216127, 2017.

[37]J. Robinson, J. A. Halliwell, J. D. Hayhurst, P. Flicek, P. Parham, and S. G. E. Marsh. The IPD and IMGT/HLA database: Allele variant databases. *Nucleic Acids Res.*, 43(D1):D423–D431, Nov. 2014.

[38]J. Siren, N. Valimaki, and V. Makinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Trans. Comput. Biol. and Bioinf.*, 11(2):375–388, Mar. 2014.

[39]T. F. Smith and M. S. Waterman. Comparison of biosequences. *Adv. Appl. Math.*, 2(4):482–489, Dec. 1981.

[40]R. Vijaya Satya, N. Zavaljevski, and J. Reifman. A new strategy to reduce allelic bias in RNA-seq readmapping. *Nucleic Acids Res.*, 40(16):e127–e127, May 2012.

12 Definitions

- base pairs (bp) = nucleotides = characters = $\{A, C, G, T, N\}$
- symbol = nucleotides $\cup\{\varepsilon\}$
- haploid genome — a sequence of nucleotides
- kmer — a string with length k that occurs somewhere as a substring (usually around 20-70bp)
- (single-end) read — a sequence of pairs (nucleotide, phred) produced by a sequencer (usually around 50-200bp but may be much longer); a phred value is the certainty of a nucleotide (we distribute the remaining 1-phred probability equally to the other 3 nucleotides). a sequencer produces a set of reads
- DBG (de Bruijn Graph) $GK < V, E >$ is a directed graph with a set of kmers as vertices, (kmer1, kmer2) is in E iff suffix(kmer1, k-1) == prefix(kmer2, k-1).
- edge (in a pDBG) — a tuples $(kmer1, kmer2, letter, p)$ for letter $\in$ nucleotides U $\{\varepsilon\}$, and for $p \in [0, 1]$; from(edge), to(edge), l(edge) and w(edge) stand for kmer1, kmer2, letter and p
- pDBG (probabilistic DBG) $GK < V, E >$ is a DBG with sum of the outgoing probabilities for every vertex equal to 1 or 0 (the outgoing edges from each non-terminal vertex form a probability distribution)
- edit-pDBG — a pDBG produced by adding edges accounting for edit operations. the edit-pDBG may contain duplicated outgoing letters
- path (in pDBG) — a sequence of edges, s.t. the end of the previous edges is the beginning of the next (repeated edges are allowed); edges(path) and vertices(path) denote the sequence of visited edges and vertices by the path
- path probability (in pDBG) — the probability of following this path as in an automata given that the start is fixed at the beginning of the path
- a read is a sequence of the same structure of a read but not necessarily produced by a sequencer (it can be a result of an assembly algorithm)
- alignment (of a read to a path) — a correspondence of the consecutive characters (with their phred values) of the read to the edges of a path; it is an equivalent of semi-global alignment between strings
- edit distance operations' probabilities (penalties) — the probability of an insertion, deletion and substitutions (dependent on the positions in the graph)
- path probability (of a path to a PNFA) — probability of sampling a path accounting for edit distance operations and transition probabilities
- matching probability (of a read to a path) — the probability that the path spells the read (accounting for phred values but not for edit distance operations and transition probabilities)
- mapping probability (of a read to a path) — the matching probability multiplied by the path probability
- global optimal mapping (of a read to a PNFA) — a mapping with maximal mapping probability
- optimal average mapping (of a read to a PNFA) — optimal mapping with maximal mapping probability divided by the alignment length
- edit distance — FiXme: explain
- PNFA (Probabilistic Nondeterministic Finate Automata) — Finate Automata with probabilities associated to each edge; every edge is labeled with a nucleotide, $\star$, or ε

$$
\begin{aligned}
score(path \mid G) &= -10\log_{10} P(path \mid G)\\
&= -10\log_{10} \prod_{e\in edges(path)} P_G(e)\\
&= \sum_{e\in edges(path)} -10\log_{10} P_G(e)\\
&= \sum_{e\in edges(path)} score_G(e)
\end{aligned}
$$

$$
\begin{aligned}
score(spell \mid \hat{r}) &= -10\log_{10} P(spell \mid \hat{r})\\
&= -10\log_{10}\left[\prod_{spell_i=\hat{r_i}}(1-P(\hat{r_i}))\prod_{spell_i!=\hat{r_i}}\frac{1}{3}P(\hat{r_i})\right]\\
&= \sum_{spell_i=\hat{r_i}} -10\log_{10}(1-P(\hat{r_i})) + \sum_{spell_i!=\hat{r_i}} -10\log_{10}\left(\frac{1}{3}P(\hat{r_i})\right)\\
&= \sum_{spell_i=\hat{r_i}} -10\log_{10}(1-P(\hat{r_i})) + \sum_{spell_i!=\hat{r_i}} \frac{1}{3}phred(\hat{r_i})
\end{aligned}
$$

$$
\begin{aligned}
score(path \mid \hat{r}) &= -10\log_{10} P(path \mid \hat{r})\\
&= score(path \mid G) + score(spell(path) \mid \hat{r})\\
&= \sum_{e\in edges(path)} \mathbf{score_G(e)} + \sum_{spell_i=\hat{r_i}} \mathbf{-10\log_{10}(1-P(\hat{r_i}))} + \sum_{spell_i!=\hat{r_i}} \mathbf{\frac{1}{3}phred(\hat{r_i})}
\end{aligned}
$$

$$
\mathbf{map_G(\hat{r})} \equiv \operatorname*{arg\,min}_{path:|spell(path)|=|\hat{r}|} score(path|\hat{r})
$$

Algorithm 2 Complete Genomes Model: Sequence generation from a set of genomes

```
1: function ChooseGenome($P_G$ : $Genome \to [0,1]$, $L$ : $Length$)
2:     $G \leftarrow sample(P_G)$                                ▷ Pick a genome
3:     return $Chop(G, L)$

4: function ChopSeq($G, L$)
5:     $start \leftarrow rand(1, |G| - L + 1)$
6:     return $G[start : start + L - 1]$                  ▷ Return a sequence

7: function phred2prob($phred$)
8:     return $pow(10, -phred/10)$

9: function MaybeSame($letter, error\_prob$)
10:    if $sample(uniform(0,1)) < error\_prob$ then
11:        return $sample(uniform(\{A,C,G,T\} \setminus letter))$
12:    else
13:        return $letter$
14: function Seq2Read($seq$ : $Sequence$, $phreds$ : $float[]$)
15:    $read \leftarrow [\,]$              ▷ Read as an array of (letter, prob) elements
16:    for all $letter, phred \in zip(seq, phreads)$ do
17:        $error\_prob \leftarrow phred2prob(phred)$
18:        $new\_letter \leftarrow MaybeSame(letter, error\_prob)$
19:        $read.append((new\_letter, phred))$
20:    return $read$
```

Algorithm 3 Core Genomes Model: Sequence generation from a set of core genomes FiXme: Add an automata illustrating the editing

```
1:  function Edit($G : Genome, P_{edit}, P_{ins}, P_{del}$)
2:      $res \leftarrow []$
3:      $i \leftarrow 1$
4:      while $i \leq |G| + 1$ do
5:          $distr \leftarrow \{(edit, l) : P_{edit}/3\,\forall l \in \{A, C, G, T\} \setminus G[i]\} \cup$
6:                  $\{(ins, l) : P_{ins}/4\,\forall l \in \{A, C, G, T\} \cup$
7:                  $\{(del, \varepsilon) : P_{del}\} \cup$
                                        ▷ Assume $\varepsilon$ if $G[i]$ is not defined
8:                  $\{(none, G[i]) : 1 - P_{edit} - P_{ins} - P_{del}\}$
9:          $op, letter \sim distr$
10:         if $letter \neq \varepsilon$ then
11:             $res.append(letter)$
12:         else
13:             $i \leftarrow i + 1$
14:     return $res$

15: function ReadGen($P_G : Genome \rightarrow [0, 1], phreds : float[], P_{edit}, P_{ins}, P_{del}, L : Length$)
16:     $G_{core} \leftarrow sample(P_G)$                        ▷ Pick a core genome
17:     $G \leftarrow Edit(G_{core}, P_{edit}, P_{ins}, P_{del})$      ▷ Introduce edits
18:     $seq \leftarrow Chop(G, L)$
19:     $read \leftarrow Seq2Read(seq, phreads)$
20:     return $read$
```

Algorithm 4 Graph Model: Sequence generation given a graph with edit edges

```
1:  function SeqGen($G : Graph, L : Length$)
2:      $curr \leftarrow q_{start}$                      ▷ Current node
3:      $seq \leftarrow []$                 ▷ Current sequence of letters
4:      while $|seq| < L$ do
5:          $e \leftarrow sample(outgoing\_edges(curr))$
6:          if $l(e) \neq \varepsilon$ then
7:              $seq.append(l(e))$
8:          $curr \leftarrow to(e)$             ▷ Move through the edge
9:      return $seq$

10: function ReadGen($G : Genome, L : Length$)
11:     $seq \leftarrow SeqGen(G, L)$
12:     $read \leftarrow Seq2Read(seq \rightarrow uniform)$
13:     return $read$
```

Algorithm 5 Best path quality estimation

```
1:  function SampleOutgoingEdge($E : Edges$)
2:      $total\_score \leftarrow sum([score(e) fore \in E])$
3:      $total\_prob \leftarrow phred2prob(total\_score)$
4:      $e2p \leftarrow set([e : phred2prob(score(e))/total\_probfore \in E])$
5:      return $sample(keys(e2p) \rightarrow values(e2p))$

6:  function SampleOutgoingEdge($G : Graph, u : Vertex, letter$)
7:      $candidates \leftarrow set()$
8:      for all $e \in G.outgoing\_edges(u)$ do
9:          $u, v, label, score \leftarrow e$
10:         if $label = letter$ or $label = \varepsilon$ then
11:             $candidates.add(e)$
12:     return $SampleEdge(candidates)$         ▷ $|candidates| > 0$

13: function SamplePath($G : Graph, r : Read$)
14:     $total\_score \leftarrow 0.0$                       ▷ Path score
15:     $u \leftarrow supersource(G)$
16:     $i \leftarrow 1$
17:     while $i < |r|$ do
18:         $letter \leftarrow MaybeSame(r_i, phred(r_i))$
19:         $u, v, label, score \leftarrow SampleOutgoingEdge(u, letter)$
20:         $u \leftarrow v$
21:         $total\_score \leftarrow total\_score + score$
22:         if $label \neq \varepsilon$ then
23:             $i \leftarrow i + 1$
24:     return $total\_score$

25: function EstimateQuality($G : Graph, r : Read, N$)
26:     for all $i \in range(N)$ do
27:         $score \leftarrow SamplePath(G, r)$
28:         FiXme: Statistics

29: function PhredScore($read\_letter, path\_letter$)
Require: $read\_letter \neq \varepsilon$ & $path\_letter \neq \varepsilon$
30:     if $read\_letter = path\_letter$ then
31:         return $phred(read\_letter)$
32:     else
33:         return $log(1 - phred2prob(phred(read\_letter)))/3$
```

1 Equivalence of Core Genomes Model and Graph Model

In this section, we discuss why the distributions induced by the Core Genome Model and the Graph Model are approximately equal. To this end, we investigate why picking a random starting point in the core genome and only consider mutations from this starting point is approximately equivalent to picking a random starting point in the mutated genome (Theorem 1.1). The approximation is quite close; the behavior of the two processes is different in less than $1/|g^{\odot}|$ of all cases.

Theorem 1.1 does not take into account that we must read L letters from $g^{\faWrench}$, starting from $b^{\faWrench}$. Instead, it considers a process reading all letters starting from $b^{\faWrench}$. However, it is straightforward to extend Theorem 1.1 (and its proof) to take this into account. In this case, the two models (the Core Genome Model and the Graph Model) differ in less than $(L+1)/|g^{\odot}|$ of all cases.

Ignoring Mutations Before the Start of a Read. The following lemma states that instead of mutating a whole core genome and picking a starting point in the mutated genome, it suffices to mutate only the part after a randomly chosen starting point in the core genome.

Lemma 1.1. *When starting from a fixed core genome $g^{\odot}$, the following processes approximately induce the same output distributions:*

- *Sample $g^{\faWrench}$ according to mutations M*
- *Uniformly sample starting point $b^{\faWrench}$*
- *Return $g^{\faWrench}[b^{\faWrench}:]$ and $M[b^{\faWrench}:]$*

and

- *Uniformly sample starting point $b'^{\odot}$ in $g^{\odot}$*
- *Sample $S'^{\faWrench}$ according to mutations M' on $g^{\odot}[b'^{\odot}:]$, assuming M' does not start with a deletion*
- *Return $S'^{\faWrench}$ and M'*

Here, $g^{\faWrench}[b^{\faWrench}:]$ ignores all letters before the $b^{\faWrench}$-th letter and $M[b^{\faWrench}:]$ ignores all mutations that occurred before producing the $b^{\faWrench}$-th letter from $g^{\faWrench}$.

Proof. As stated in Theorem 1.2, picking $b^{\faWrench}$ in $g^{\faWrench}$ is equivalent to picking $b'^{\odot}$ in $g^{\odot}$, unless $b'^{\odot} = |g^{\odot}| + 1$. As the latter happens very rarely (with a probability of less than $1/|g^{\odot}|$), we can safely ignore this case.

Therefore, there is a one-to-one correspondence between the mutations $M[b^{\faWrench}:]$ and M' (here, it is crucial that M' does not start with a deletion). Hence, their respective distributions are the same. As a consequence, the distribution of $g^{\faWrench}[b^{\faWrench}:]$ and $S'^{\faWrench}$ is also the same, as they are deterministically derived from their respective mutations.

Uniform Position in $g^{\odot}$. The following lemma states that picking a uniform point in the mutated genome $g^{\faWrench}$ is equivalent to picking a uniform point in the core genome $g^{\odot}$.

It assumes one-based indexing for $b^{\odot}$ and $b^{\faWrench}$.

Lemma 1.2. *Picking a uniform point $b^{\faWrench} \in_{u.a.r} \{1, \ldots, |g^{\faWrench}|\}$ and determining its origin $b^{\odot}$ yields a uniformly picked $b^{\odot} \in_{u.a.r} \{1, \ldots, |g^{\odot}|\}$, given that $b^{\odot} \neq |g^{\odot}| + 1$.*

Proof. We prove this lemma by induction over the number of insertions (n) and deletions (m), always assuming that $b^{\odot} < |g^{\odot}| + 1$ Then, as the statement is true for any fixed number of insertions and deletions, it is also true for arbitrary numbers. We do not consider edits, as they are irrelevant for the selection of $b^{\odot}$ and $b^{\faWrench}$.

Base Case. The base case holds because for $n = 0$ and $m = 0$, picking $b^{\faWrench}$ is analogous to picking $b^{\odot}$.
Induction Step (insertions). For the induction step where we add one insertion, recall that the mutations M describe all operations when copying $g^{\odot}$.

We observe that the distribution of M for $n+1$ insertions and m deletions equals the distribution resulting from adding an insertion to the distribution of M' for n insertions and m deletions, where we sample the position of the new insertion uniformly over all possible positions in M'. This is because due to symmetry, all locations for the new insertion are equally likely.

Given M, assuming that we do not select the newly inserted position, $b^{\odot}$ is distributed uniformly, according to the induction hypothesis. Otherwise, as every position is equally likely to be selected for the insertion,$b^{\odot}$ is also uniformly distributed. Here, it is important that $b^{\odot} \neq |g^{\odot}| + 1$, as adding a new insertion to the end increases the probability that $b^{\odot}$ lies at the end (i.e., that $b^{\odot} = |g^{\odot}| + 1$).

Induction Step (deletions). For the induction step where we add one deletion, the distribution of M for n insertions and $m+1$ deletions equals the distribution resulting from replacing one copy transition by a deletion transition in M' (M' again consists of n insertions and m deletions). Since every copy transition has the same probability of being replaced by a deletion, the distribution of $b^{\odot}$ remains uniform.