Introducing the Practical Haplotype Graph Version 2: A Streamlined and Simple Pangencode System

Zachary Miller
Institute for Genomic Diversity
Cornell University
Introduction

- Lower cost of sequencing - but still cost prohibitive to sequence entire plant breeding populations at high depths
- Reference Quality Assemblies are becoming widely available for a number of staple food crops with more on the way
- Can we use the diversity captured by a collection of assemblies (pangenome) to impute low cost short read genotype data better than traditional reference alignment techniques?
Introducing the Practical Haplotype Graph (PHG)

- Started development in 2017
- Initial success in building PHGs
  - Imputation
  - Genomic Selection
  - General data storage
- But had some issues
  - Utilized a custom Postgres DB
  - Certain components slow
  - Overly Parameterized
  - User Interface hard to use
  - Poor/Out of date documentation
PHGv2 - Works to address these issues

- Fast
- Easy to use
- Clear, Concise, and Up-to-date Documentation
- Integrate standard software development practices
  - Continuous Integration - Code is tested early, often and automatically. >80% of code is covered by unit tests
  - Continuous Delivery - Once code is reviewed and merged in a new build and release of the package happens automatically
- Utilize state of the art community tools as much as possible
  - Anchorwave - aligner
  - tileDB - Genotype(VCF) storage
  - Assembled Genomes Compressor(AGC) - Sequence Storage
What do we mean by Practical Haplotype Graph?

● **Practical**
  ○ Keep it simple!
  ○ Biology produces genomes with consistent patterns
  ○ Somewhat Conserved Genes + Intergenic Regions with tremendous variation
  ○ Slice the genome at these conserved boundaries -> Reference Ranges
    ■ Simplifies the pangenome representation

● **Haplotype**
  ○ A set of DNA variations, or polymorphisms, that tend to be inherited together
  ○ Store both sequence and Variants

● **Graph**
What is a graph?

Vertex

Edge
A directed acyclic graph (DAG)
PHG - Trellis Graph

Reference Range 1 → Reference Range 2 → Reference Range 3 → Reference Range 4

Individual 1

Individual 2

Individual 3
**Terms**

- **Reference Range**
  - A Segment of the Reference Genome
  - Typically recommended that range Start and End are conserved
  - Common way to define these are genic boundaries

- **Haplotype**
  - A set of DNA variations, or polymorphisms, that tend to be inherited together
  - The PHG holds the following information for each haplotype
    - Variants - in gVCF file
    - Nucleotide Sequence - aligned to the reference for a specific Reference Range
PHG - Trellis Graph

Reference Ranges

<table>
<thead>
<tr>
<th>Reference</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>GATCG</td>
<td></td>
<td>ATTGAG</td>
<td>CTAGA</td>
<td>CCAAGG</td>
</tr>
<tr>
<td>Assembly A</td>
<td>GATCG</td>
<td></td>
<td>CTAGA</td>
<td>CCAAGG</td>
</tr>
<tr>
<td>Assembly B</td>
<td>GAGCG</td>
<td>ATTGAG</td>
<td>CTACAGA</td>
<td>CCAAGG</td>
</tr>
<tr>
<td>Assembly C</td>
<td>GAGCG</td>
<td></td>
<td>CTAGA</td>
<td>CCAAGG</td>
</tr>
</tbody>
</table>
Create the Database

Create Ranges

Load AGC with ref and assemblies

Align Assemblies and Convert to gVCF/hVCF

Load Ref and Assemblies to TileDB

AGC compressed fastas

TileDB datasets

h.VCF

Find Paths

Read Mapping

Build Kmer Index

WGS/short reads

Impute from WGS short reads
Build a PHG - Create Ranges

GFF File
chr1 . gene 1 5
chr1 . mRNA 2 4
chr1 . CDS 2 4
chr1 . gene 12 16
chr1 . mRNA 13 15
chr1 . CDS 13 15

Output BED file
chr1 0 5
chr1 5 11
chr1 11 16
chr1 16 23

> phg create-ranges --gff reference.gff --reference-file Reference.fasta
   --output refRanges.bed
Create the Database

Create Ranges

Load AGC with ref and assemblies

Align Assemblies and Convert to gVCF/hVCF

Load Ref and Assemblies to TileDB

AGC compressed fastas

TileDB datasets

h.VCF

Find Paths

Read Mapping

Build Kmer Index

Impute from WGS short reads

Reference.gff

Reference.fa

Assemblies

WGS/short reads
## Build a PHG

- We have a Reference and 3 Assemblies

<table>
<thead>
<tr>
<th>Reference</th>
<th>GATCGATTGAGCTAGACCAAGG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly A</td>
<td>GATCGATGCTAGACCAAGG</td>
</tr>
<tr>
<td>Assembly B</td>
<td>GAGCGATTGAGCTACAGACCAAGG</td>
</tr>
<tr>
<td>Assembly C</td>
<td>GAGCGATGCTAGACCAAGG</td>
</tr>
</tbody>
</table>

```bash
>phg agc-compress --reference-file Ref.fa --fasta-list listOfFastas.txt
```
Create the Database

Create Ranges

Reference.gff
Reference.fa
Assemblies

Load AGC with ref and assemblies

AGC compressed fastas

Align Assemblies and Convert to gVCF/hVCF

Load Ref and Assemblies to TileDB

TileDB datasets

Find Paths

Read Mapping

Build Kmer Index

h.VCF

Impute from WGS short reads

WGS/short reads
Build a PHG: Build Reference Haplotypes

- Use Conserved Base Pairs to slice the genome
- Can use genic boundaries from the GFF annotation

```
>phg create-ref-vcf --bed /my/bed/file.bed --reference-file Ref.fa
   --reference-name Reference --output-dir /path/to/vcfs
```
hVCF - Simple Haplotype Storage Format

- We can store a set of haplotypes in a VCF based file format.
- Key idea: hash the sequence to have a unique identifier.
  - These identifiers can be stored as Symbolic alleles in a VCF file.

**Diagram:**

```
GATCG → MD5 Hash → 10f47f
```
## hVCF - Simple Haplotype Storage Format

### Reference Haplotypes

<table>
<thead>
<tr>
<th>Haplotype</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>GATCG</td>
<td>ATTGAG</td>
</tr>
<tr>
<td>CTAGA</td>
<td>CCAAGG</td>
</tr>
</tbody>
</table>

### Example haplotype entries

```plaintext
##ALT=<ID=10f47f,SampleName="Reference",Regions=chr1:1-5>,GT=1/1
##ALT=<ID=7d046e,SampleName="Reference",Regions=chr1:6-11>,GT=1/1
##ALT=<ID=9fb476,SampleName="Reference",Regions=chr1:12-16>,GT=1/1
##ALT=<ID=1d471f,SampleName="Reference",Regions=chr1:17-22>,GT=1/1
```

### Sample entries

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Position</th>
<th>Base</th>
<th>Haplotype</th>
<th>END</th>
<th>Genotype</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr1</td>
<td>1</td>
<td>G</td>
<td>&lt;10f47f&gt;</td>
<td>5</td>
<td>1/1</td>
</tr>
<tr>
<td>chr1</td>
<td>6</td>
<td>A</td>
<td>&lt;7d046e&gt;</td>
<td>11</td>
<td>1/1</td>
</tr>
<tr>
<td>chr1</td>
<td>12</td>
<td>C</td>
<td>&lt;9fb476&gt;</td>
<td>16</td>
<td>1/1</td>
</tr>
<tr>
<td>chr1</td>
<td>17</td>
<td>C</td>
<td>&lt;1d471f&gt;</td>
<td>22</td>
<td>1/1</td>
</tr>
</tbody>
</table>
```
hVCF Benefits

● Small # of ‘variants’
  ○ Only number of Reference ranges ~100k

● VCF based
  ○ Community Standard
  ○ Easy to understand
  ○ Lots of tools out there to process and analyze the data

● Works with small and large genomes
  ○ Supports .csi indexing so big genomes like wheat(15-17 Gbp) work just fine

● Sequences can be reconstituted based on haplotype metadata
  ○ Verify by checking the ID against the hash

● Can load into TileDB
Build a PHG: Align Assemblies to Reference

- We wrap the Anchorwave aligner to do this accurately and efficiently

```
>phg align-assemblies --gff anchors.gff --reference-file Ref.fa
--a assembliesList.txt -o /path/for/alignment/files
```
Build a PHG: Build Assembly Haplotypes

Reference
GATCGATTGAGCTAGACCAAGG

Assembly A
GATCGAT - - - GCTAGACCAAGG

Reference Haplotypes
GATCG ATTGAG CTAGA CCAAGG

Assembly A Haplotypes
GATCG AT - - G CTAGA CCAAGG

>phg create-maf-vcf --bed anchors.bed --reference-file Ref.fa
  --maf-dir /path/for/alignment/files -o path/to/vcfs
Create the Database

Create Ranges → Load AGC with ref and assemblies → Align Assemblies and Convert to gVCF/hVCF → Load Ref and Assemblies to TileDB

AGC compressed fastas → TileDB datasets

h.VCF → Find Paths

Read Mapping → Build Kmer Index

Impute from WGS short reads
Build a PHG: Do all Assemblies + Load

- Reference Haplotypes: GATCG ATTGAG CTAGA CCAAGG
- Assembly A Haplotypes: GATCG AT---G CTAGA CCAAGG
- Assembly B Haplotypes: GAGCG ATTGAG CTACAGA CCAAGG
- Assembly C Haplotypes: GAGCG AT---G CTAGA CCAAGG

>phg load-vcf --vcf-dir /path/to/vcfs
Directed edges connect each haplotype with all haplotypes in next reference range.

Stronger weights are set for consecutive haplotypes of a given assembly.

Now that we have a PHG, what can we do with it?
Imputation

Reference Ranges

Reference
1. GATCG
2. ATG
3. CTAGA
4. CCAAGG

Assembly A
1. GATCG
2. AT---G
3. CTAGA
4. CCAAGG

Assembly B
1. GAGCG
2. ATG
3. CTACAGA
4. CCAAGG

Assembly C
1. GAGCG
2. AT--G
3. CTAGA
4. CCAAGG
Imputation

- We can have 1000s of samples sequenced with low depth short reads
  - GBS, DaRTSeq, Skim Sequence, anything in a fastq
- 2 Step Process - per sample
  - Read Mapping
  - Path Finding
- Goal is to have this process take minutes
Imputation: Read Mapping

- Traditional Alignment tools like Minimap2 do work, but can we be more efficient?
- Aligning against the pangenome substantially increases resource requirements
- For this reason, we developed a Kmer based read mapping approach
Imputation: Index the Graph
phg build-kmer-index --db-path /path/to/tiledb
--index-file /path/to/write/index.txt
--hvcf-dir /path/to/hvcfs
Create the Database

Create Ranges

Load AGC with ref and assemblies

Align Assemblies and Convert to gVCF/hVCF

Load Ref and Assemblies to TileDB

AGC compressed fastas

TileDB datasets

h.VCF

Find Paths

Read Mapping

Build Kmer Index

WGS/short reads

Impute from WGS short reads
Map Reads

>phg map-kmers
  --hvcf-dir /path/to/hvcfs/
  --kmer-index /path/to/index
  --read-files file1.fq, file2.fq
  --output-dir /path/to/output

<table>
<thead>
<tr>
<th>HaplId Set</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>10f47f</td>
<td>1</td>
</tr>
<tr>
<td>0a16aa</td>
<td>1</td>
</tr>
<tr>
<td>9fb476, 361174</td>
<td>1</td>
</tr>
<tr>
<td>361174</td>
<td>1</td>
</tr>
<tr>
<td>1d471f</td>
<td>1</td>
</tr>
</tbody>
</table>
Kmer Read Mapper Performance

- Tested with a Maize PHG made with ~80 assemblies
  - Paired end 150 bp WGS reads
- Because of the low footprint for mapping, once an index is built, read mappings can be generated on just about any Linux machine

<table>
<thead>
<tr>
<th></th>
<th>Minimap2(PHGv1)</th>
<th>Kmer (PHGv2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Graph - 1 time</td>
<td>30 min(100GB RAM)</td>
<td>60 min(80GB RAM)</td>
</tr>
<tr>
<td>Map 5x WGS Paired End</td>
<td>90 min(100GB RAM)</td>
<td>50 sec(&lt;10GB RAM)</td>
</tr>
</tbody>
</table>
Create the Database

Create Ranges → Load AGC with ref and assemblies → Align Assemblies and Convert to gVCF/hVCF → Load Ref and Assemblies to TileDB

AGC compressed fastas → TileDB datasets

h.VCF
Find Paths → Read Mapping → Build Kmer Index

Impute from WGS short reads

Reference.gff → Reference.fa → Assemblies
<table>
<thead>
<tr>
<th>Haplod Set</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>10f47f</td>
<td>1</td>
</tr>
<tr>
<td>0a16aa</td>
<td>1</td>
</tr>
<tr>
<td>9fb476, 361174</td>
<td>1</td>
</tr>
<tr>
<td>361174</td>
<td>1</td>
</tr>
<tr>
<td>1d471f</td>
<td>1</td>
</tr>
</tbody>
</table>
Apply Hidden Markov Model

> phg find-paths --path-key-file key-file.txt --hvcf-dir /path/to/hvcfs/
   --output-dir /path/to/hvcf/output --path-type haploid
Imputation: Path Finding

- The find-paths command will output a hVCF for each sample
- Allows the PHG to associate SNPs from assemblies with a new sample
- Can do both haploid and diploid path finding
- Option to use likely-parents to improve imputation
How Can You Actually Use the Graph?

- `phg_v2` comes bundled with a simple BrAPI compliant ktor server

> `phg start-server`
Types of Analysis You Can Do!

- Genomic Selection / Genome Wide Association Studies
  - Can use imputed variants or imputed haplotypes
- Link and Visualize Haplotype information with metadata
- Generate Kinship/Distance Matrices
- Subset regions of the paths that you would like to focus on
  - This gene is interesting, what is surrounding it?
- Any type of analysis you can do with a VCF you can try with an hVCF
Whats next? - Summer Plans

● QC Reports
  ○ Each step will report back useful QC metrics for easy pipeline debugging

● Rare Allele pipeline
  ○ Using imputed paths can we find rare alleles within samples?
  ○ Can we use this to add additional diversity to the graph?

● AI Driven imputation
  ○ Can we train a model to give better imputation results?

● Support more Species
  ○ Right now ~5 species PHGv2s are being built
    ■ Maize, Sorghum, Cassava, Wheat, Cotton and more on the way
    ■ What are stress points?
    ■ What is confusing?
Check out our Github!

- We have great documentation
- Daily builds and releases
- We welcome code contributions!
- Issues can be submitted through Github or the ‘phg’ Biostars tag
Acknowledgements

Core Development Team:

- Ana Berthel
- Brandon Monier
- Lynn Johnson
- Peter Bradbury
- Terry Casstevens

Biology QC/Alpha Test:

- Bethany Econopouly
- Cinta Romay
- Qi Sun

IGD/Buckler Lab Leadership

- Ed Buckler
- Sara Miller

This research is funded by NSF Plant Genome Project (NSF #1822330), Bill and Melinda Gates Foundation, and USDA-ARS.
Questions?