Chapter 4

De novo genome assembly

4.1 Introduction

All existing DNA sequencing technologies are are limited by short read lengths, orders of magnitude shorter than whole chromosomes (see e.g. [147]). All the DNA sequence data generated during this PhD is in the form of 100 or 125bp long reads (section 3.2.1). The goal of *de novo* genome assembly is to reconstruct the underlying genome sequence from the reads by finding and following overlaps between sequences. It is often helpful to visualise and conceptualise this process by drawing an 'assembly graph', where each vertex corresponds to a sequence and edges between vertices correspond to overlaps between reads, as in Figure 4.1. If error-free reads could be obtained from a single underlying genome sequence with randomly generated bases, the assembly task would be a trivial problem given 100bp read length. A 50bp overlap between two reads would then effectively guarantee that the two DNA sequences indeed originated from overlapping loci in the underlying genome sequence (with a negligible error rate of 7.8×10^{-31}). In the real world, however, genome assembly is a difficult problem. This is due to a combination of three factors: 1) DNA sequencing errors; 2) heterozygosity; 3) the highly repetitive nature of vertebrate genome sequences (see section 1.1.2).

When sequencing DNA from a diploid organism (this includes humans and the majority of animals), most DNA sequence is found in two copies, one contributed by the mother and the other by the father. Heterozygous sites, the differences between the maternally and paternally contributed sequences (chromosomes), are usually represented within the assembly graph topology as 'bubbles' (Figure 4.1).

Genome assembly typically involves three major stages: error correction, assembly, and *scaffolding*. First, the correction process attempts to eliminate the majority of sequencing errors. Second, sequence overlaps are found and sequences merged into

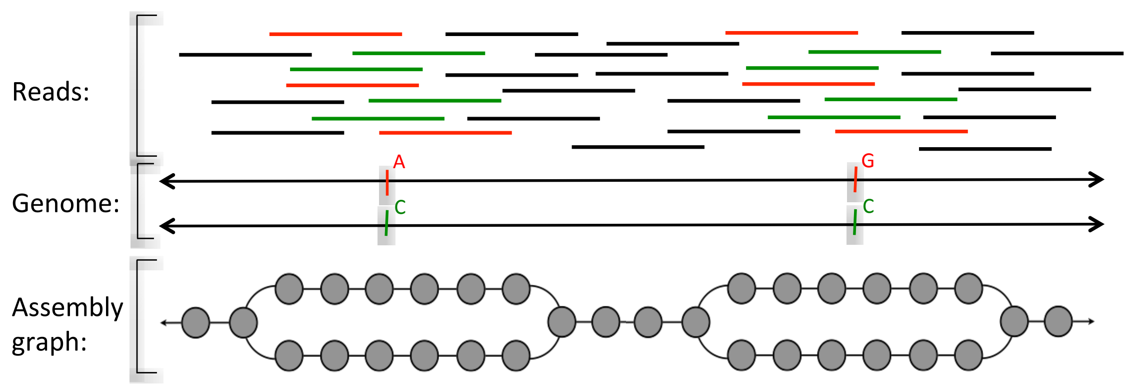

Fig. 4.1 **An example of genome assembly graph with two bubbles due to heterozygous sites**.

contiguous segments (*contigs*). During assembly, the assembly algorithms attempt to to resolve repeats and remove additional sequencing errors and heterozygous variants by examining the structure of the assembly graph. The contig ends if the underlying genome sequence cannot be resolved due to ambiguity (caused by repeats alone, or by any combination of repeats, errors, and heterozygous variants), or if there is not any read/sequence with sufficient overlap to add to the end of the contig. Third, genome assemblers use the long range information provided by paired-end or mate-pair reads whose inserts (see section 3.2.1) span unresolvable sequences. Thus, assemblers generate *scaffolds* - multiple contigs linked together and separated by gaps. Unresolvable sequence between the contigs is denoted by runs of the symbol `'N'` and insert sizes are used to determine approximate lengths of the unresolvable sequences.

The quality of a genome assembly is often evaluated based the distribution of lengths of assembled sequences (contigs and scaffolds). A commonly used summary statistic for assessment of contiguity of a genome assembly is N50. N50 is the length of the longest sequence s where half the length of the genome is assembled in sequences greater than or equal in length to s [148] (Figure 4.2).

When the true length of the genome is known (e.g. has been estimated by flow cytometry), it is also possible to assess the assembly by comparing the total length of the assembled fragments with the expected length of the genome. Finally, the options for assessing the accuracy for a *de novo* assembly are limited by the fact that in most cases the true answer is not known. Internal consistency of the assembly can be evaluated for example by assessing the consistency of alignment of paired-end reads to the completed assembly [148].

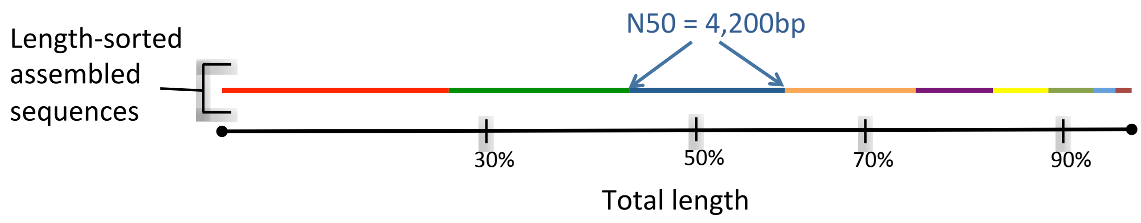

Fig. 4.2 **An illustration of the N50 measure of assembly contiguity**. Analogous measures can be defined to assess the length of assembled sequences covering a given proportion of the genome, e.g. N30 at 30%, or N70 at 70%.

I have completed all assemblies presented in this thesis using the `sga v0.10.13` genome assembler [149], and using my own extension `'trio-sga'` which takes advantage of mother-father-offspring trio sequence data to reduce problems associated with heterozygosity. The majority of sga's algorithms are based on efficient queries over an *FM-index*, a compressed index of a set of reads [150, 151]. Specifically, the FM-index data structure facilitates finding all occurrences of a sequence of length k (a k-mer; k must be shorter than the read length) in a set of reads in time that is independent of how many reads are searched.

Formally, let $\Re$ be a set of $|\Re|$ DNA reads, $\Re = \{R_0, R_1, ..., R_{|\Re|-1}\}$. Reads are indexed by $i = \{0, 1, 2, ..., |\Re| - 1\}$, and the length of each read is denoted by $|R_i|$. Then let Q be a query sequence of length $|Q|$, where $\exists i$ such that $|Q| \leq |R_i|$. Finding all occurrences of Q within $\Re$ by exhaustive search requires

$$\sum_{i=0}^{|\Re|-1} (|R_i| - |Q| + 1)$$

string comparisons. For example, searching for a 30-mer (k-mer of size 30bp) in the *Andinoacara coeruleopunctatus* dataset would require approximately 38billion comparisons. The FM-index based 'Backward Search' algorithm [150] counts all occurrences of a 30-mer within $\Re$ in 30 steps, regardless of the size of $\Re$. The number of CPU cycles in each step is comparable to the number of CPU cycles required for a single string comparison.

4.2 `trio-sga` - Trio-aware genome assembly

4.2.1 Overview

The majority of available genome assembly algorithms have been developed for organisms that are inbred, homozygous, or have low levels of heterozygosity and many genome projects reduced or even completely eliminated heterozygosity, by using inbred laboratory strains (e.g. the *C. elegans* genome project [152] and *D. melanogaster* genome project [153]), or by obtaining a homozygous form of the organism by other laboratory techniques (e.g. the potato genome [154]). However, in some cases such approaches are not feasible, for example if the organism in question is resistant to inbreeding due to strong inbreeding depression [155], or if the generation time is too long or the organism unsuitable for inbreeding in the laboratory. High heterozygosity can make genome assembly very challenging [156]. Algorithms specifically designed for organisms with moderate-to-high levels of heterozygosity have been developed [157, 158], but the methods still the lag in performance compared with assemblies of homozygous strains - simply due to the fundamental difficulty of assembling highly heterozygous genomes.

Communities of scientists have recently come together with the aims to *de novo* assemble the genomes of 10,000 vertebrate species [159], 5,000 arthropod species [160], and 7,000 (mainly marine) invertebrates [161]. Many species, especially in the latter two groups, will have high levels of heterozygosity.

Therefore, I have developed `trio-sga` - a set of three algorithms designed to facilitate better quality genome assembly for organisms with moderate-to-high levels of heterozygosity. `trio-sga` algorithms extend the `sga` genome assembler [149]. Two of the algorithms use haplotype phase information in mother-father-offspring trios to eliminate the majority of heterozygous sites even before the assembly itself (i.e. search for sequence overlaps) commences. The third algorithm is designed to reduce sequencing costs by enabling the use of parents' reads in the assembly of the genome of the offspring. In this section, I briefly describe `trio-sga`. In Section 4.3, I illustrate its performance by assembling highly heterozygous *Heliconius* butterfly genomes. Then, in Section 4.4, I demonstrate that the algorithms can improve assembly contiguity even at the lower levels of heterozygosity found in cichlids.

`trio-sga` software is available at `https://github.com/millanek/trio-sga`. It is written in C++, and can run multithreaded on UNIX-like systems. The core code implementing the logic of Algorithms 1 and 2 is in the file `TrioCorrectProcess.cpp` and the code for Algorithm 3 is in `FilterParentProcess.cpp`.

4.2.2 Algorithms

The input into `trio-sga` are three separate sets of DNA reads: reads from the mother, reads from the father, and reads from their offspring. `trio-sga` assembles the reads from the offspring, taking advantage of information present in the parents' reads with the aim to generate two separate assemblies of the offspring's genome: maternal (i.e. the haplotype inherited from the mother) and paternal (i.e. the haplotype inherited from the father).

The basic building block of all three `trio-sga` algorithms is a query over an FM-index about the number of occurrences of a given k-mer and of its reverse complement in a read set. For each trio we build three FM-indices (separately for the reads from the mother, father, and the offspring). K-mer occurrences in reads from the mother's DNA are denoted $\mathrm{C}_M(\mathrm{k})$, occurrences in reads from the father are denoted $\mathrm{C}_F(\mathrm{k})$, and from the offspring $\mathrm{C}_O(\mathrm{k})$. The three `trio-sga` algorithms are described below.

1. Pre-filtering the set of reads sequenced from the offspring in order to reduce heterozygosity. Two, usually overlapping, sets of reads are generated with the goal of eventually assembling the paternally and maternally contributed chromosomes separately. A conceptual overview of this algorithm, assuming error-free reads is in Algorithm 1 and Figure 4.3A.

Algorithm 1: Filtering the set of reads sequenced from the offspring in order to reduce heterozygosity (assuming error-free reads)

```
Data: FM-indices of mother and father reads; reads from the offspring
Result: Two partially overlapping sets of offspring reads for paternal and maternal haplotype
        assembly
foreach (read R from the offspring) do
    foreach (k-mer k in R) do
        if ($C_F(\mathbf{k}) > 0$ and $C_M(\mathbf{k}) == 0$) then
            assign R to paternal assembly read set; assigned = TRUE;
        end
        if ($C_F(\mathbf{k}) == 0$ and $C_M(\mathbf{k}) > 0$) then
            assign R to maternal assembly read set; assigned = TRUE;
        end
    end
end
if assigned = FALSE then
    assign R to both read sets
end
```

2. Improving error correction. Error correction used by `sga` and most other genome assemblers (e.g. ref [162]) is based on k-mer frequencies. It relies on the fact that the number of occurrences in the read set of error-containing k-mers is in general lower than the number of occurrences of k-mers that do not contain errors, i.e. the frequency distributions of correct and error-containing k-mers differ (Figure 4.3B and Figure 4.3C). In practice, an occurrence threshold is set to distinguish between correct and error-containing k-mers. However, in most data sets (depending on the error rate) there is a 'grey zone' where the two distributions overlap (Figure 4.3C), with low k-mer occurrences of correct sequence due to low coverage and/or non-random sampling and high k-mer occurrences of error-containing k-mers for example due to repeated (or systematic) errors. Using data from the parents helps to distinguish between error-containing and correct sequences within the grey zone and to prevent under-correcting (accepting as correct reads that contain errors) and over-correcting ('fixing' reads that are in fact correct). For example, if a k-mer fails the threshold in the offspring, but is present above threshold in one of the parents, it is unlikely to be an error and

correction is not attempted. Algorithm 2 outlines how parents' data are used in the decision on whether or not to attempt correction on a k-mer in the offspring.

Algorithm 2: Trio-aware error correction: deciding whether to attempt correcting a k-mer

```
Data: FM-indices of mother, father, offspring reads; reads from the offspring
Result: Corrected offspring reads
// Initialise occurrence thresholds for k-mers in offspring, mother, and
   father FM-Indices
Init: set thresholds $t_O$, $t_M$, $t_F$;
// Initialise indicator variables for ensuring haplotype phase consistency of
   corrected reads
Init: set mc=FALSE; set fc=FALSE;
foreach (read R from the offspring) do
    foreach (k-mer k in R) do
        if ($C_F(k) < t_F$ and $C_M(k) < t_M$) then increase $t_O$;
        // Test the offspring threshold
        if ($C_O(k) > t_O$) then next;
        else
            // This k-mer failed offspring threshold - test it in the parents
            if ((mc == fc) or (mc==TRUE and fc==FALSE)) then
                if ($C_M(k) > t_M$) then set mc.temp=TRUE;
            end
            if ((mc== fc) or (mc==FALSE and fc==TRUE)) then
                if ($C_F(k) > t_F$) then set fc.temp=TRUE;
            end
            if (mc.temp==TRUE or fc.temp==TRUE) then
                // Passed k-mer count threshold in the parental reads
                set mc = mc.temp; set fc = fc.temp; next;
            end
        end
        // Call the correction algorithm (not shown)
        correction(R,k);
    end
end
```

Line 5: if a k-mer found in the offspring does not occur (above threshold) in either parent, it is likely to be an error (or a de-novo mutation, but these are exceedingly rare compared with errors). Therefore, I increase the offspring k-mer occurrence threshold for this k-mer.

Lines 16-19: It is necessary to ensure haplotype phase consistency in error-correction; for example, if a k-mer is not corrected thanks to passing the threshold in the mother (but not the father), I assume that the read comes from the maternal haplotype. I keep track of this information (i.e. set **mc**=TRUE) and only take the mother's reads into account when assessing k-mers from the remainder of the read.

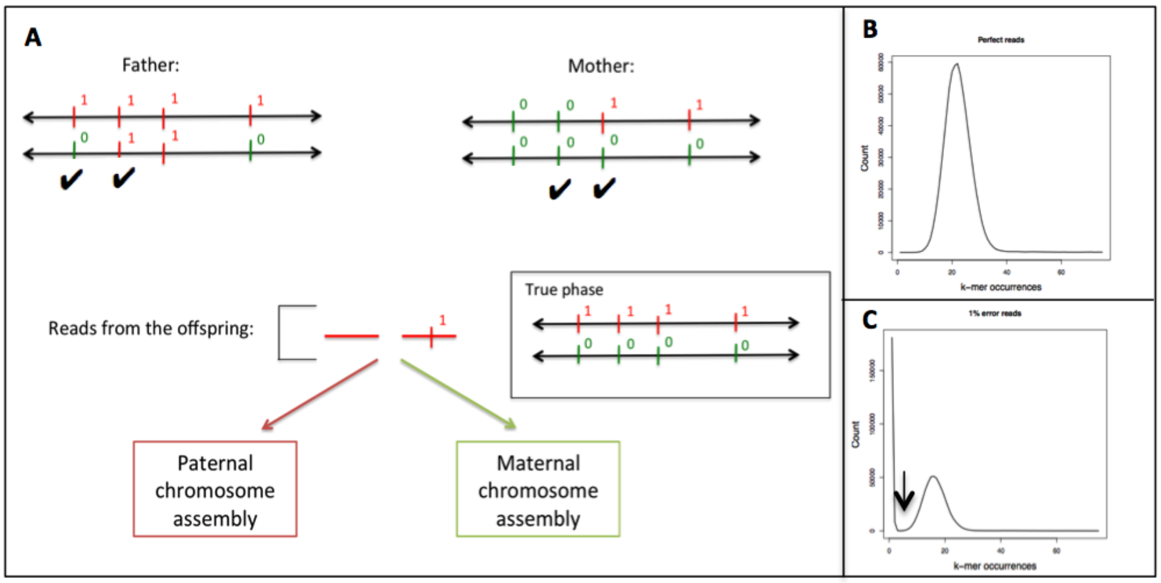

Fig. 4.3 **Trio aware read filtering and error correction.** **(A)** An example region of the genome with four segregating sites. The offspring inherited a haplotype with four derived alleles (denoted as 1) from the father and ancestral alleles from the mother. Read (or read pairs) from DNA containing the first or the second segregating site and the derived allele (as shown) can be phased and confidently assigned for paternal haplotype assembly. Similarly, reads from DNA containing the second or the third segregating site and the ancestral allele can be phased and confidently assigned for maternal haplotype assembly. **(B)** The distribution of 31-mer counts in simulated 100bp error-free reads with 30X genome coverage. **(C)** The distribution of 31-mer counts in simulated 100bp reads with uniform 1% error rate. There are now many more k-mers with low k-mer occurrences (<3); these are mainly errors, but there is a 'grey zone' (arrow), with kmers occurring 2-4 times being a mixture of correct and error-containing sequences. Figures **(B)** and **(C)** by Jared Simpson.

3. An algorithm to ‘fill’ regions of low coverage in the offspring by bringing in reads sequenced from the parents’ DNA, thus using the parents’ datasets to ‘assemble through’ these regions. Sequencing costs grow almost linearly with sequencing depth and in the trio assembly setting, three genomes need to be sequenced. This algorithm helps to keep the costs down. Reads from the father’s DNA are checked for consistency with the offspring’s paternal chromosome assembly and used to fill coverage gaps, and reads from the mother are used in the same way for the maternal chromosome assembly.

Algorithm 3: Check for consistency between error-corrected reads from the mother and error-corrected reads from the offspring’s maternal haplotype or reads from the father and the paternal haplotype.

Data: FM-indices of error-corrected reads from one parent and the corresponding haplotype in the offspring

Result: Reads from the parent that are consistent with the offspring read set and can be used to bridge coverage gaps

```
foreach (read R from the parent) do
    consistent=TRUE;
    foreach (k-mer k in R) do
        if ($C_O$(k) == 0) then
            consistent=FALSE; break;
        end
    end
    if (consistent) then mark R as consistent with the offspring;
end
```

4.3 *Heliconius* butterfly genome assemblies

I applied `trio-sga` to the deep coverage cichlid samples (Table 3.2 - Panel B), and will describe this work in section 4.4 below. However, the heterozygosity in these cichlid samples was not high enough for the trio approach to have a very large effect. To illustrate more dramatically the benefits of `trio-sga`, I will detour from cichlids to describe its application to *Heliconius* butterflies from South America, in collaboration with John Davey and Chris Jiggins from the Department of Zoology at Cambridge University.

With 43 species and a multitude of colour races that have radiated across the tropics of the South and Central America, *Heliconius* butterflies, like cichlids, provide outstanding opportunities to study a wide variety of evolutionary phenomena, including adaptation and speciation [163]. In common with other insects, such as *D. melanogaster*, *Heliconius* butterflies tend to have very large N_e and, therefore, very heterozygous genomes [164, 165]. Sequencing libraries for three *Heliconius* mother-father-offspring trios were prepared and sequenced by the Sanger Institute sequencing core, obtaining 125bp paired-end reads with 300-500bp insert sizes.

The samples include:

1. A *Heliconius melpomene* trio
2. A *Heliconius cydno* trio
3. A cross between two *Heliconius* species: *H. cydno* mother, *H. melpomene* father, hybrid offspring

I obtained estimates of genome size and of heterozygosity using `sga preqc`, a recent extension of `sga` for estimating characteristics of a genome based on metrics derived from a random subset of reads [166]. The results (Figure 4.4) revealed that both *Heliconius* species species have genome sizes of ~280Mb, consistent with 292Mb flow cytometry estimate for *H. melpomene* [167]. The `sga preqc` genome size estimate for the hybrid offspring of the cross is ~560Mb, twice the true genome size. The erroneous estimate stems from the very high level of heterozygosity in the hybrid. On average, one in every 33bp is heterozygous in the hybrid. Heterozygosity in *H. cydno* is estimated to be $\sim\frac{1}{50}$bp and in *H. melpomene* $\sim\frac{1}{70}$bp. For comparison, heterozygosity in the cichlid *L. lethrinus* is estimated to be approximately an order of magnitude lower at $\sim\frac{1}{450}$bp.

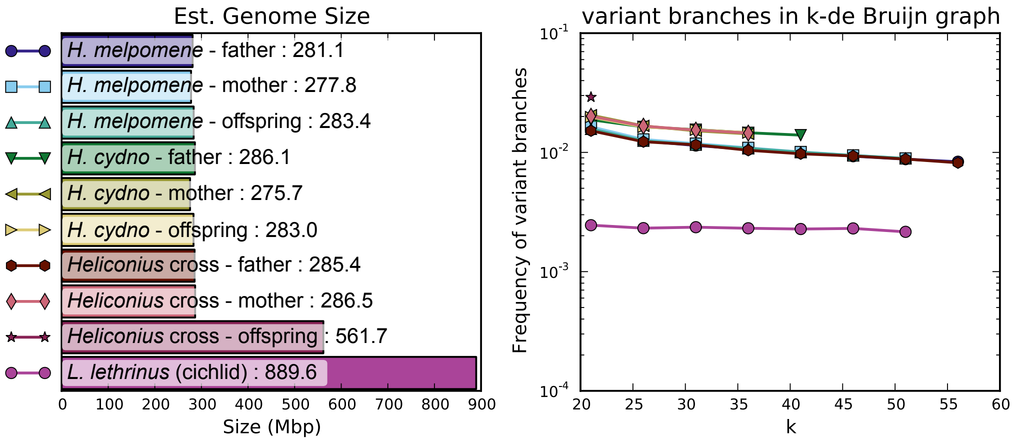

Fig. 4.4 **Estimates of genome sizes and heterozygosity for *Heliconius* genomes.** The cichlid species *L. lethrinus* has been included for comparison.

All read pairs with undetermined bases (`N` characters) were removed with the `sga preprocess` command. Then the FM-Indices were built and the `trio-sga` read filtering (phasing) and error-correction algorithms were called as follows:

```
sga correct-trio --paired --phase -x 3 -k 41 --mother-kmer-threshold=3
```

```
--father-kmer-threshold=3 offspring.fastq.gz mother.fastq.gz father.fastq.gz
```

The `trio-sga` read filtering (phasing) algorithm reduces heterozygosity in *Heliconius* data by approximately two orders of magnitude. Estimates by `sga preqc` show that the filtered datasets for *H. melpomene* and *H. cydno* have one heterozygous site approximately every 1700bp. Average heterozygosity in the hybrid was reduced even further to $\sim\frac{1}{3300}$bp (Figure 4.5).

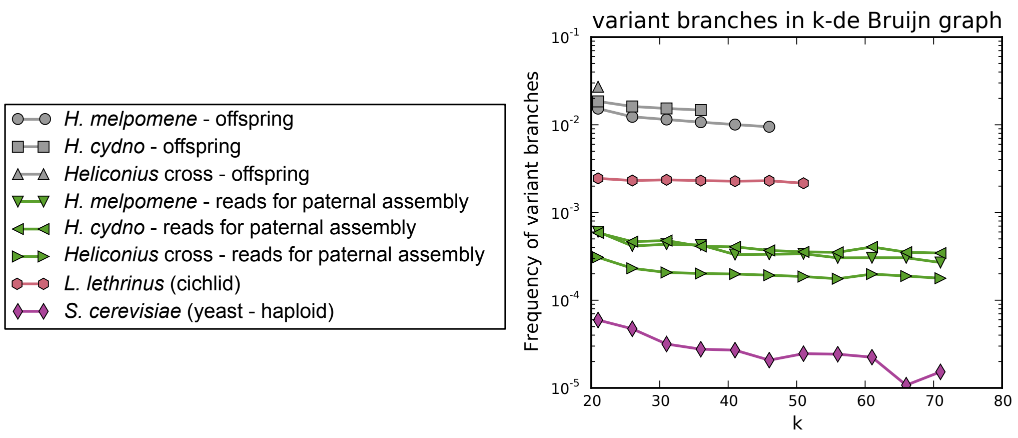

Fig. 4.5 **Read phasing reduces heterozygosity in *Heliconius* data.** Heterozygosities for the offspring samples (grey) and the cichlid *L. lethrinus* estimated from the same data as in Figure 4.4. The phased datasets (green; only paternal haplotype shown) have heterozygosity estimates up to two order of magnitude lower. Heterozygosity values for the haploid yeast species represent a misclassification rate ($<10^{-4}$) observed in `sga preqc` estimates [166].

Reads from the father consistent with the paternal haplotype were obtained by first error-correcting the father reads independently with `sga` and then using the following `trio-sga` command:

```
sga filter-parents --do-not-correct --paired -k 41 father_corrected.fastq.gz
offspring_paternal.fastq.gz
```

The reads were merged with the paternal haplotype reads, and then assembled into contigs using `sga`. This will be referred to as *trio assembly* in the rest of this section. I also assembled the offspring reads without using the parent's data. This will be referred to as *normal assembly*. In all cases, the `-r 10` parameter was used for the `sga assemble` subprogram, and assemblies were attempted with minimum overlap required between reads set to 70, 80, 90, 95, 100, 105, and 110bp. Then, I choose the assembly with the highest contig N50 statistic.

Table 4.1 ***Heliconius* contig assembly statistics.** Contigs of less than 500bp excluded.

Species	Assembly	Best minimum overlap (bp)	Total length (Mb)	Contiguity stats (bp)			
				N30	N50	N70	N90
H. melpomene	normal	80	367	1,908	1,258	879	618
H. melpomene	trio	95	269	12,185	7,626	4,181	1,225
H. cydno	normal	80	369	1,456	1,030	773	586
H. cydno	trio	90	263	13,516	8,675	4,951	1,467
Heliconius cross	normal	70	432	1,678	1,175	848	611
Heliconius cross	trio	90	259	17,297	11,259	6,844	2,731

Assembly statistics in Table 4.1 demonstrate that normal assemblies of the highly heterozygous *Heliconius* genomes are very challenging for `sga`. Contig N50 statistics are barely above 1kb and the total length of the assemblies is greater than the genome size estimate, suggesting that in many cases two copies of a single genomic region have been retained. In contrast, trio assemblies have contig N50 between 7.5 and 11.2kb and assembly lengths correspond to genome sizes.

Given that the normal assemblies were clearly very poor, I generated scaffolds only for the trio assemblies. The paired-end reads used for the trio assembly were aligned to the contigs (excluding contigs of less than 200bp) using `bwa mem v0.7.10` [132], and alignments processed by `samtools v1.1` [135] to generate sorted `bam` files as follows:

```
bwa mem -p -M contigs.fa reads.fa.gz | samtools fixmate -O sam - - | samtools
view -b -h -F 256 - > alignment.bam

samtools sort -@ 4 -T temp -O bam -o alignment_sorted.bam alignment.bam
```

and scaffolds then generated with the requirement for evidence from at least five pairs of reads before joining two contigs:

```
sga-bam2de.pl --prefix n5 -n 5 -m 200 alignment_sorted.bam

sga-astat.py -m 200 alignment_sorted.bam > alignment_sorted.astat

sga scaffold -m 200 -a alignment_sorted.astat --pe n5.de -o n5_scaffolds contigs.fa

sga scaffold2fasta --write-unplaced -m 200 -o n5_scaffolds.fa --use-overlap
-a contigs-graph.asqg.gz n5_scaffolds

sga gapfill -o n5_scaffolds_gapfilled.fa --prefix=reads n5_scaffolds.fa
```

Scaffold assembly statistics are shown in Table 4.2. Again, all the results are for paternal haplotypes. The paternal haplotype in the cross comes from the *H. melpomene* father.

Table 4.2 ***Heliconius* scaffold assembly statistics.** Scaffolds of less than 500bp excluded.

Species	Assembly	Total length (Mb)	Scaffold number	Gaps	Contiguity stats (bp)			
					N30	N50	N70	N90
H. melpomene	trio	273	44,740	14,944	31,535	19,406	10,276	2,408
H. cydno	trio	267	35,196	13,286	37,454	23,575	12,900	3,472
Heliconius cross	trio	260	22,284	10,331	45,640	29,456	17,738	6,802

It is interesting that in both contig and scaffold assemblies, the best (most contiguous) trio assembly is for the cross, followed by *H. cydno* and then *H. melpomene*. This pattern suggests that the more heterozygous the individual/species, the better trio assembly can be achieved.

The first set of *Heliconius* data was sequenced in February 2015, with average genome coverage ~40-50X per individual (~120-150X per trio). The coverage of the offspring reads drops when the `trio-sga` read phasing algorithm divides reads into two sets, with the magnitude of the drop depending on how many reads can be phased (the drop was 47% for the cross, 39% for *H. cydno*, and 34% for *H. mepomene*). Coverage is later recovered when reads from the father consistent with the paternal haplotype and reads from the mother consistent with the maternal haplotype are brought in. Nevertheless, I was concerned that the drop in coverage caused by `trio-sga` read phasing algorithm could lead to breaks in the assembly caused by insufficient coverage. Therefore, we sequenced more *Heliconius* DNA from the same samples in May 2015, doubling the coverage per individual to ~80-100X.

After doubling the coverage, I obtained trio assemblies in the same way as described above. Table 4.3 - Panel A lists statistics for the contig assemblies. The increase in coverage enabled me to increase minimum required overlap between reads from 90-95bp to 105-110bp. However, improvements in contig lengths have been small. The N50 of the paternal contigs increased from 8.7 to 9.3kb (~7%) for *H. cydno* and from to 11.3 to 12.7kb (~12.4%) for the cross. The *H. melpomene* paternal contigs could not be compared because of technical problems with this assembly. So far, two scaffold assemblies with high coverage data have been finished (Table 4.3 - Panel B). The N50 of the paternal scaffolds of the cross increased from 29.4 to 33.4kb, an increase of ~13.6%.

Table 4.3 ***Heliconius* assembly statistics - high coverage data.** Only trio assemblies were generated.

Panel A: Contig assemblies; contigs of less than 500bp excluded.

Species	**Haplotype**	**Best minimum overlap (bp)**	**Total length (Mb)**	**Contiguity stats (bp)**			
				N30	**N50**	**N70**	**N90**
H. melpomene	maternal	110	263	12,909	7,842	4,170	1,161
H. cydno	maternal	105	253	14,667	9,501	5,586	1,813
H. cydno	paternal	110	268	14,577	9,344	5,306	1,535
Heliconius cross	maternal	105	263	19,278	12,457	7,454	2,744
Heliconius cross	paternal	105	263	19,460	12,674	7,614	2,905

Panel B: Scaffold assemblies; scaffolds of less than 500bp excluded.

Species	**Haplotype**	**Total length (Mb)**	**Scaffold number**	**Gaps**	**Contiguity stats (bp)**			
					N30	**N50**	**N70**	**N90**
H. cydno	maternal	258	32,547	16,069	42,515	27,047	15,281	3,898
Heliconius cross	paternal	265	22,370	11,513	51,986	33,374	19,980	7,343

4.4 Cichlid trio genome assemblies

As shown previously in Figure 4.4 and discussed in more detail in chapter 5, the levels of heterozygosity in cichlids are approximately an order of magnitude lower than in *Heliconius* butterflies. Therefore, it was interesting to see if reducing heterozygosity using `trio-sga` can deliver improvements in cichlid genome assemblies. Deep coverage cichlid samples (Table 3.2) were assembled using both the normal and trio methods as described above for *Heliconius*. Because the read-length was 100bp, minimum overlap required between reads was set to 65, 70, 75, and 80bp.

Cichlid contig assembly statistics are shown in Table 4.4. Compared with normal `sga` assemblies, using trio data with `trio-sga` algorithms increases contig N50 by 35% to 45% in all three cichlid species.

Table 4.4 **Cichlid contig assembly statistics.** Contigs of less than 500bp excluded.

Species	**Assembly**	**Haplotype**	**Best min. overlap**	**Length (Mb)**	**Contiguity stats (bp)**			
					N30	**N50**	**N70**	**N90**
A. calliptera	normal	---	70bp	654	4,532	2,980	1,867	930
A. calliptera	trio	maternal	80bp	681	6,361	4,118	2,540	1,159
A. calliptera	trio	paternal	80bp	681	6,381	4,153	2,564	1,167
A. stuartgranti	normal	---	70bp	656	4,842	3,125	1,935	949
A. stuartgranti	trio	maternal	75bp	677	6,579	4,222	2,571	1,164
A. stuartgranti	trio	paternal	75bp	679	6,644	4,253	2,588	1,167
L. lethrinus	normal	---	65bp	640	4,054	2,691	1,723	890
L. lethrinus	trio	maternal	75bp	673	5,873	3,852	2,407	1,126
L. lethrinus	trio	paternal	75bp	673	5,888	3,861	2,410	1,130

Scaffold assemblies with paired-end reads were obtained in the same way as described above for *Heliconius*. Paired-end scaffold assembly statistics are shown in Panel A of Table 4.5. The statistics reveal that the most of `trio-sga` N50 improvements at the contig level are carried forward to the paired-end scaffolds, with increases of 25% to 35%. At this stage, with N50 of ~9-12kb, the cichlid assemblies are much more fragmented than the corresponding *Heliconius* assemblies which have N50 of ~20-33kb. This may be partly due to lower genome coverage and shorter read lengths, but also probably due to more complex repeat structure of cichlid genomes.

For scaffolding with mate-pair reads, I aligned them to the paired-end scaffolds using `bwa mem v0.7.10` [132], processed the alignments by `samtools v1.1` [135] to generate sorted `bam` files as follows:

```
bwa mem -p -M paired-end-scaffolds.fa mate-reads.fa.gz | samtools fixmate -O
sam - - | samtools view -b -h -F 256 - > mate-alignment.bam
```

```
samtools sort -@ 4 -T temp -O bam -o mate-alignment_sorted.bam mate-alignment.bam
```

and then generated mate-pair scaffolds with the requirement for evidence from at least three mate-pairs before joining two paired-end scaffolds:

```
sga-bam2de.pl --prefix n3-mate -n 3 -m 200 mate-alignment_sorted.bam
```

```
sga-astat.py -m 200 mate-alignment_sorted.bam > mate-alignment_sorted.astat

sga scaffold -m 200 -a mate-alignment_sorted.astat --mate n3-mate.de
-o n3_mate-scaffolds n5_scaffolds_gapfilled.fa

sga scaffold2fasta --write-unplaced -m 200 -o n3_mate-scaffolds.fa
-f n5_scaffolds_gapfilled.fa n3_mate-scaffolds

sga gapfill -o n3_mate-scaffolds_gapfilled.fa --prefix=reads n3_mate-scaffolds.fa
```

The effect of repeat structure in cichlid genomes of assembly contiguity becomes apparent when mate-pair reads with larger insert sizes are used for generating scaffolds. The contiguity of mate-pair scaffolds of the trio assemblies (Table 4.5 - Panel B), as measured by N50, is approximately 5-fold better than when scaffolding with paired-end reads alone. Cichlid mate-pair insert sizes are generally around 1-2kb; full distributions are shown in Figure 4.6. It is therefore clear that the long range information present in mate-pair reads was enabled the assemblies to span over a large number of repeat sequences in this size range. It is also interesting to note that *A. stuartgranti*, the species with the longest mate-pair inserts, has the best mate-pair scaffold N50.

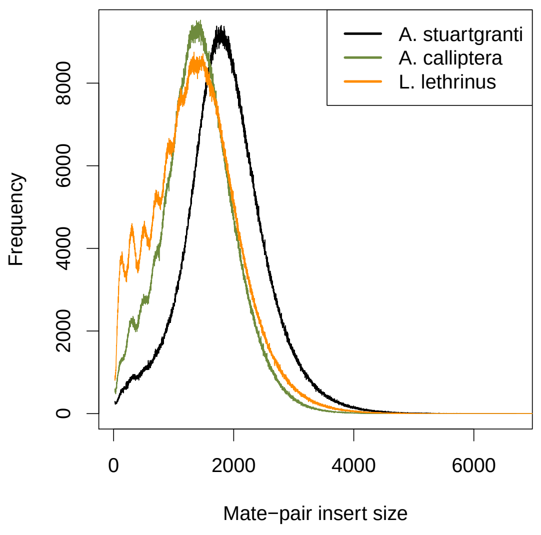

Fig. 4.6 **The distributions of cichlid mate-pair insert sizes**

Table 4.5 **Cichlid scaffold assembly statistics.** Scaffolds/contigs < 500bp excluded.

Panel A: Scaffolds assembled with paired-end reads (300-500bp insert size).

A. calliptera	normal	---	Not assembled				
A. calliptera	trio	maternal	690	19,720	12,155	6,811	2,432
A. calliptera	trio	paternal	689	20,075	12,310	6,876	2,428
A. stuartgranti	normal	---	671	13,679	8,485	4,855	1,835
A. stuartgranti	trio	maternal	686	17,579	10,838	6,136	2,251
A. stuartgranti	trio	paternal	687	17,781	10,879	6,115	2,233
L. lethrinus	normal	---	655	11,039	6,980	4,089	1,601
L. lethrinus	trio	maternal	679	15,034	9,364	5,355	2,006
L. lethrinus	trio	paternal	679	15,034	9,391	5,374	2,008

Panel B: Scaffolds after adding mate-pair reads (~1-2kb insert size).

A. calliptera	trio	maternal	720	97,843	54,823	26,262	5,366
A. calliptera	trio	paternal	719	98,689	55,879	26,524	5,366
A. stuartgranti	trio	maternal	732	104,746	59,945	29,520	5,985
A. stuartgranti	trio	paternal	733	102,317	59,588	29,285	5,801
L. lethrinus	trio	maternal	722	90,820	51,728	25,624	5,020
L. lethrinus	trio	paternal	722	90,316	51,956	25,647	5,038

4.5 *Andinoacara coeruleopunctatus* genome assembly

The genome of the Central American cichlid *Andinoacara coeruleopunctatus* was assembled using normal `sga` pipeline as described in Section 4.3. Assembly statistics are shown in Table 4.6. The total length of the assembly is similar to East African cichlids, while assembly contiguity is better, likely due to higher coverage (~60X vs. ~40X) and longer read lengths (125bp vs 100bp).

Table 4.6 ***A. coeruleopunctatus* assembly statistics.** Scaffolds/contigs < 500bp excluded.

Panel A: Contig assemblies.

Species	Best minimum overlap (bp)	Total length (Mb)	Contiguity stats (bp) N30	N50	N70	N90
A. coeruleopunctatus	90	687	7,780	5,123	3,219	1,487

Panel B: Paired-end scaffold assemblies.

Species	Total overlap (bp)	Contiguity stats (bp) N30	N50	N70	N90
A. coeruleopunctatus	699	26,236	16,890	10,174	4,132

Figure fig:PanamaScaffoldDist shows the full distribution of scaffold lengths in the *A. coeruleopunctatus* assembly.

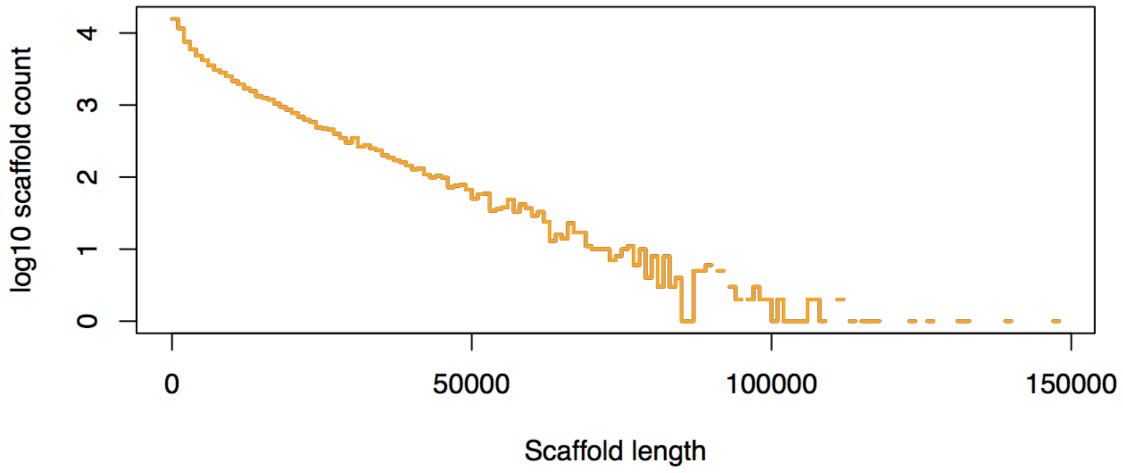

Fig. 4.7 **Distribution of scaffold lengths in *A. coeruleopunctatus* assembly.**