HapSolo: An optimization approach for removing secondary haplotigs during diploid genome assembly and scaffolding.

Edwin A. Solares, Yuan Tao, Anthony D. Long, Brandon S. Gaut[1]

Dept. of Ecology and Evolutionary Biology
University of California, Irvine
Irvine, CA 92697-2525

[1] Address for Correspondence: bgaut@uci.edu

ABSTRACT

Background: Despite marked recent improvements in long-read sequencing technology, the assembly of diploid genomes remains a difficult task. A major obstacle is distinguishing between alternative contigs that represent highly heterozygous regions. If primary and secondary contigs are not properly identified, the primary assembly will overrepresent both the size and complexity of the genome, which complicates downstream analysis such as scaffolding.

Results: Here we illustrate a new method, which we call HapSolo, that identifies secondary contigs and defines a primary assembly based on multiple pairwise contig alignment metrics. HapSolo evaluates candidate primary assemblies using BUSCO scores and then distinguishes among candidate assemblies using a cost function. The cost function can be defined by the user but by default considers the number of missing, duplicated and single BUSCO genes within the assembly. HapSolo performs gradient descent to minimize cost over thousands of candidate assemblies. We illustrate the performance of HapSolo on genome data from three species: the Chardonnay grape (*Vitis vinifera*), a mosquito (*Anopheles funestus)* and the Korean Mudskipper fish (*Periophthalmus magnuspinnatus*).

Conclusions: HapSolo rapidly identifies candidate assemblies that yield dramatic improvements in assembly metrics, including decreased genome size and improved N50 scores. N50 scores improved by 26%, 8.5% and 21% for Chardonnay, mosquito and the mudskipper, respectively, relative to unreduced primary assemblies. The benefits of HapSolo were amplified by down-stream analyses, which we illustrated by scaffolding with Hi-C data. We found, for example, that prior to the application of HapSolo, only 39% of the Chardonnay genome was captured in the largest 19 scaffolds, which corresponds to the number of chromosomes. After the application of HapSolo, this value increased to 77%. The improvements for mosquito scaffolding was similar to that of Chardonnay but even more dramatic for mudskipper.

Keywords: Genome, assembly, alternative haplotype, haploid, purge, deduplication, scaffolding,

BACKGROUND

Traditionally, reference genomes have been produced from genetic materials that simplify assembly; for example, the first two plant species targeted for reference quality genomes, *Arabidopsis thaliana* [1] and rice (*Oryza sativa*) [2], were chosen in part because they naturally self-fertilize and are therefore highly homozygous. Other early genomes, such as those from *Caenhrabdidtis elegans* and *Drosophila melanogaster [3, 4]*, were also based on inbred, highly homozygous materials. Recent sequencing of additional model and non-model species have continued to rely on near-homozygous materials, either through inbreeding [5, 6] or by focusing on haploid tissue [7, 8].

The reliance on homozygous materials is fading rapidly, however, for at least three reasons. The first is that it has become clear that inbred materials can misrepresent the natural state of genomes. A dramatic illustration of this fact is that some lines of maize purged 8% of their genome in only six generations of selfed fertilization [9]; more generally, inbred genomes tend to be smaller than those based on outbreeding species [10, 11]. The second is that many species of interest cannot be easily manipulated into a homozygous state. Many animals fall into this category, such as mosquitoes [12], as do many perennial crops like grapes, which are highly heterozygous [13] and can be selfed but only with substantial fitness costs that limits homozygosity [14]. Finally, some important features and phenotypes - such as sex determination [15] and other important adaptations - can only be identified by analyzing heterozygous samples.

The resolution of highly heterozygous regions, which often contain large structural variants, is now possible due to improvements in sequencing technologies and their affordability. In theory, long-read sequencing technologies, like those from Pacific Biosciences and Oxford Nanopore, provide the capability to resolve distinct haplotypes in heterozygous regions, leading to the assembly of reference-quality diploid genomes [5, 16, 17]. Several genomes based on highly heterozygous materials have been published recently [13, 18–22], with many additional efforts ongoing.

Nonetheless, the assembly of heterozygous genomes still presents substantial challenges. One challenge is resolving distinct haplotypes in regions of high heterozygosity. Programs that assemble long-reads, such as FALCON and Canu [23], can fuse distinct haplotypes into the primary assembly. This haplotype-fusion produces genomes that are much larger than the expected genome size. When haplotypes are fused, either into the same contig or as different contigs into the primary assembly, the increased size and complexity of the assembly complicates down-stream approaches, such as scaffolding by Hi-C or optical

mapping. In theory, Falcon-unzip [19] solves some problems by identifying alternative (or 'secondary') haplotigs that represent the second allele in a heterozygous region and then providing a primary assembly without secondary contigs.

It remains a difficult problem to identify and remove alternative contigs during assembly, but there are some existing solutions. For example, Redundans identifies secondary contigs via similarities between contigs [24] and removes the shorter of two contigs that share some pre-defined level of similarity. Another approach, PurgeHaplotigs uses sequence coverage as a criterion to identify regions with two haplotypes [25]. The reasoning behind PurgeHaplotigs is that alternative alleles in a heterozygous region should have only half the raw sequence coverage of homozygous regions. Accordingly, the algorithm proceeds by first remapping raw reads to contigs, then flagging contigs with lower than expected read depth, and finally re-mapping and removing low-coverage contigs from the primary haplotype-fused assembly. A more recent approach, implemented in the purge_dups tool [26], builds on the coverage-based approach of PurgeHaplotigs.

Here we report another strategy, which we call HapSolo, to identify and remove potential secondary haplotigs. Our approach is similar to Redundans, in that it begins with an all-by-all comparison among contigs and uses features of sequence alignment as a basis to identify potential alternative haplotigs. However, HapSolo utilizes more features of sequence alignments and is unique in exploring the parameter space of alignment properties to optimize the primary assembly, using features of BUSCO scores as the optimization target. Here we detail the approach and implementation of HapSolo, demonstrate that it efficiently identifies primary vs. secondary haplotigs and show that it improves scaffolding outcomes. HapSolo has been implemented in python and is freely available (https://github.com/esolares/HapSolo).

METHODS

Approach & Implementation

Pre-processing: Our method begins with the set of contigs from genome assembly. In theory, HapSolo will work for any set of contigs from any assembler and from any sequencing type (i.e., short-read, long-read or merged assemblies); we focus here on assemblies from long-read data generated by the Pacific Biosciences platform. Given the set of contigs, the first step is to perform comparisons among all contigs (**Figure 1**). In practice, we exclude long contigs (> 10Mb) as queries (but not as references) to limit computational time, reasoning that it is unlikely that long contigs represent alternative haplotigs. We also parallelize alignment by separating

each contig as a single query using one CPU each. Although not technically a feature of HapSolo, our github release provides a script to run Blat v35 [27] using this parallel approach.

Steps within HapSolo: Given contig alignments, HapSolo imports alignment results into a PANDAS (https://pandas.pydata.org/) dataframe to form a table in which each row represents a pair of aligned contigs and the columns contain descriptive statistics for each pairwise comparison. These columns include the percent nucleotide identity between contigs (*ID*), which is a similar to those used in previous haplotig reduction programs; the proportion of the query contig length that aligns to the reference contig (*Q*); and the ratio of the proportion of the query aligned to the reference relative to the proportion of the reference aligned to the query (*QR*), which reflects properties of length (and potential structural variants) differences between contigs. We consider all three parameters (*ID*,*Q* and *QR*) because they help encompass information about various kinds of alignments (**Figure 1**). In practice, we do not just use *QR* for filtering but also *QR'* = $e^{-log(QR)}$. The four parameters - *ID*, *Q, QR* and *QR'* - are the basis for filtering query contigs from the table and defining them as putative secondary contigs. For simplicity, only *QR* will be discussed, because *QR'* is dependent on *QR*.

In addition to the alignment table, HapSolo generates a table of BUSCO properties [28] for each contig. This BUSCO analysis is performed on each contig of the assembly prior to running HapSolo's reduction algorithm. To perform these analyses, contigs are split into individual fasta files and then BUSCO v3.0.2 is run in *geno* mode on each contig using the orthoDB9 datasets and the AUGUSTUS species option. BUSCO v3.0.2 relies on BLAST v2.2.31+, AUGUSTUS v3.3 and BRAKER v1.9. Because we run each contig separately and contigs are considered in parallel, the process of generating the BUSCOs is rapid (see Results). Ultimately, the BUSCO table generated by HapSolo contains a list of complete (C) and fragmented (F) BUSCO genes for each contig. This table is integral for rapidly evaluating potential candidate assemblies.

Given the alignment table and the BUSCO table, HapSolo begins by assigning values of *ID, Q* and *QR* randomly, with set default values or with values defined by the user. These values are applied to the alignment table to discriminate primary from second contigs; the remaining primary contigs comprise the candidate assembly. For each candidate assembly, HapSolo calculates the number of Fragmented (F), Missing (M), Duplicated (D) and Single-Copy (S) BUSCO genes across all primary contigs, based on values in the BUSCO table. It then calculates the *Cost* of the candidate assembly as:

$$f(Cost) = (\theta_1 M+\theta_2 D+\theta_3 F)/ \theta_4 S$$

where θ_1, θ_2, θ_3, and θ_4 are weights that can vary between 0.0 and 1.0. Weights can be assigned by users; for all of our analyses below, we employ weights of 0.0 for F and 1.0 for M, D and S.

We then employ gradient descent to minimize *Cost* (Figure 1). Once *Cost* is calculated with random starting values, *ID, Q* and *QR* are modified by step functions, which in turn defines a new set of primary contigs for a new cost evaluation. If *Cost* does not change, or if parameters increase past their maximum limits of ID = Q = 1.00, new random values of *ID*, *Q* and *QR* are assigned. The process is repeated for *n* iterations, and the iteration(s) with the smallest *Cost* are used to define the final set of primary contigs. When there are multiple solutions that minimize *f(Cost)*, we retain all unique solutions; these additional solutions can be exported by the user for post-processing steps and evaluation. The dynamics of this minimization - e.g., the threshold for the number of consecutive cost plateaus, the number of *x* unique best candidate assemblies, the increases in step size, and the number of iterations - can be set by the user. The ability to perform random restarts of the gradient descent allows for the introduction of parallelization, allowing the user to run *n* number of iterations by *k* number of processes, thereby increasing search by multiples of *k*. As we will show below, parallelization and other efficiencies, since using static BUSCO and alignment tables, translate to rapid processing times.

To retain candidate assemblies with the smallest *Cost*, we implemented a unique priority queue (UPQ). The UPQ maintains a maximum number of *x* best assemblies, where *x* can be set by the user. The UPQ initially takes a list of one set of values, the score, primary contigs and other assembly information. The UPQ then takes the number of primary contigs for each of the candidate assemblies and sorts them by size. It then compares only the candidate assemblies of the same size, because assemblies of unequal size cannot be the same assembly. Therefore our algorithm, in order to reduce the number of contig set comparisons, only compares contig sets of the same size. Once it is established that the candidate assemblies of the same # of contigs are equal, only the candidate assembly with the lowest score is saved. The list is then sorted by score and returned. This allows retention of the max score of the best *x* number of assemblies by looking at the score of the last candidate assembly in the list, giving $O(1)$ access to this value. Sorting takes $O(x*log(x))$, where *x* is the best number of candidate assemblies to return, giving our UPQ a time complexity of $O(x*log(x))$. Since we can instantaneously access our worst case best *x* candidate assembly, we can then do an integer comparison of the score of our current candidate assembly with the worst score of our best *x* number of assemblies, reducing our computational time complexity. Only assemblies with the same or lower scores than the worst candidate assembly are then added to our UPQ.

This reduces our total time complexity to *O(i*x*log(x))* where *i* is the number of iterations which produce scores lower than our max of *x*, and *x* is the number of best candidate assemblies to keep.

Post-processing: Once HapSolo converges on a set (or sets) of primary contigs that minimize *Cost*, the contig set is employed for post-processing steps to evaluate the candidate assembly. Specifically, we run QUAST v4.5 [29] and BUSCO 3.0 on the set of primary contigs that represent the best (or set of best) candidate assembly. QUAST measures basic genome assembly statistics, such as, N50, total assembly length, L50 and the largest contig size. Although not technically part of the HapSolo method, we provide scripts that run QUAST and BUSCO to output their results into a single score file.

Implementation and requirements: HapSolo has been implemented in Python 2.7 and requires the input of a fasta file, a BLAT alignment output file as a PSL or compressed PSL.gz file, and an appropriate directory structure. HapSolo requires the PANDAS package to be installed prior to running.

RESULTS & DISCUSSION

Primary Assemblies

We illustrate the application and results of HapSolo on three diploid genome data sets. The three - including the Chardonnay grape (*Vitis vinifera*), the Anopheles mosquito (*A. funestus*) and the Korean Giant-fin Mudskipper fish (*Periophthalmus magnuspinnatus*) - represented a range genome sizes, at 606Mb, 752Mb and 211Mb for Chardonnay, mosquito and mudskipper, respectively; a range of raw sequence coverage, with 58x, 240x, and 70x coverage, respectively; and different assembly methods - a hybrid assembly for Chardonnay and Falcon_Unzip for mosquito and mudskipper. The sequencing data were all based on the Pacific Biosciences (PacBio) sequencing platform, but HapSolo should be applicable to any contig data.

For each species, we applied HapSolo with and without gradient descent and compared the outcomes to the original unreduced assembly. **Table 1** provides assembly statistics, and it clearly illustrated improvements from the unreduced assembly, to the assembly without gradient descent (-GD) based on default values, and finally to the assembly with gradient descent (+GD), which is based on random starting values and 50,000 iterations. Focusing on Chardonnay, for example, the primary genome size declined 17% from the unreduced assembly to the +GD assembly and 5% from the -GD assembly to the +GD assembly. Not surprisingly, as genome size decreased, so did the number of contigs included in the assembly, which fell from 2,072 to

1,369 (-GD) to 1,155 (-GD). However, contig N50 increased by 26% from 1.066 Mb to 1.441 Mb. Similar results were achieved after applying HapSolo to contigs from mosquito and mudskipper (**Table 1**). For both assemblies, the number of contigs, L50 and genome size decreased, while the contig N50 improved by 8.5% for mosquito and 21% for mudskipper. We note, however, that gradient descent did not increase N50 for the mosquito assembly much beyond that achieved by applying HapSolo for one iteration with its default values, suggesting that the default values performed well for the contig N50 with this dataset.

Although N50 did not decline for the mosquito data, it was clear that our implementation of gradient descent reduced *Cost,* as we expected, with the expected effects on BUSCO scores. **Figure 2** illustrates *Cost* as a function of the number of iterations, showing that *Cost* steadily decreased from the random starting conditions. It was also clear, however, that the biggest improvements occurred within the first 1,000 iterations for all three datasets. After ~1,000 iterations, improvements were minor but still ongoing. Overall, the continued improvement in *Cost* suggests there is value in applying gradient descent on new data sets, especially given that the computational costs are minor (see below). **Table 2** complements information about *Cost* by reporting BUSCO scores. HapSolo achieved its principal goal, which is to generally increase the representation of single copy (S) BUSCO genes and decrease duplicated (D) genes in reduced compared to unreduced assemblies. It is particularly interesting to note the differences between the -GD and +GD assemblies, because in some cases the -GD assembly had more single copy genes, but at the cost of also having many more duplicated genes. Thus, the +GD option can produce assemblies with lower *Cost* but with fewer BUSCO genes.

Finally, **Figure 3** plots the cumulative contig assembly length for the three assemblies for each of the three species, and it illustrates two important points. First, HapSolo reduced the total assembly length primarily by removing numerous contigs of small size. Second, differences between the +GD and -GD reduced assemblies were more evident for some species (e.g., mudskipper) than for others (e.g., Chardonnay). Nonetheless, when there were differences, gradient descent decreased both assembly size (Table 1) and *Cost* (Table 2).

Hi-C Scaffolding Results

HapSolo focuses on the improvement of primary assemblies, but there are clear advantages to removing haplotigs for downstream operations such scaffolding. The impact of the failure to remove duplicate haplotigs on scaffolding could be dramatic due to false joins between duplicate haplotigs or redundant non-parsimonious joins between duplicate haplotigs and adjacent single copy regions. Here we illustrated the advantage of running HapSolo on

primary assemblies prior to Hi-C scaffolding. For these analyses, the unreduced assembly and both reduced assemblies (i.e., -GD and +GD) were scaffolded using the 3D-DNA pipeline [30], resulting in more contiguous assemblies overall. We compared the improvements of the two scaffolded HapSolo assemblies against the unreduced scaffolded assembly (**Table 3**). Gains in improvements to the largest scaffold were clear across all assemblies relative to the unreduced assembly. For example, the largest scaffold increased by 71% (-GD) and 91% (+GD) for Chardonnay, by 22% (-GD) and 118% (+GD) in mosquito, and by 24% (-GD) and 51% (+GD) for Mudskipper. Scaffold N50 improvements were mixed, because there were dramatic improvements in grape (227% and 188% for -GD and +GD assemblies) and mudskipper (1544% and 1362% for Mudskipper) but only moderate improvements for mosquito (10% for both assemblies).

Figure 4 illustrates scaffold improvements for each of the three species. Applying HapSolo prior to scaffolding results in more of the total genome being contained in the *k* largest scaffolds, where *k* is the haploid number of chromosomes for each species. For example, Chardonnay has 19 chromosomes, and the 19 largest scaffolds based on the unreduced assembly represented 33% of the genome size. Following HapSolo haplotig reduction, the largest 19 scaffolds encompassed ~76% of the total scaffolded genome size (**Table 3**). Similar results are seen for the two other species, and in all cases HapSolo scaffolded assemblies were superior to the unreduced scaffolded assemblies. This being said, although the value of gradient descent was particularly clear for mosquito, where the first 3 scaffolds (representing *n*=3 chromosomes) represented ~68% of genome with scaffolded -GD assembly *versus* 85% for the +GD reduced assembly, gradient descent offered negligible advantages in the other two species. Of course scaffolded chromosomes could occasionally be split (e.g., into two large arms for metacentric chromosomes), and it is reassuring in this regard that scaffolds larger than 10Mb account for 85.1%, 93.9%, and 86.8% of the total scaffolded genome size in HapSolo +GD corrected assemblies for Chardonnay, Mosquito, and Mudskipper genomes respectively (**Table 3**). Contrast this with 33.0%, 88.0%, and 28.2% for uncorrected scaffolded genomes which had many more small scaffolds that Hi-C was unable to connect.

Execution Time and Memory Efficiency

To measure runtime, HapSolo was run on an in-house custom built dual CPU Intel E5-2696 V2 Nodes containing 512GB of RAM and storage attached via a 40Gbe connection. CPU runtime depends on the number of iterations, but it is also dependent on the parameter values chosen. We measured runtime across the datasets, measuring different configurations in terms

of the number of cores and the number of iterations per core (**Table 4**). Under the conditions we used for empirical data (i.e., gradient descent on 10 cores with 5000 iterations per core), the total time was <2 hours for both Chardonnay and mosquito but substantially longer at 6 hours and 45 minutes for the mudskipper. Note that memory usage was dependent on the size of the alignment file and independent of the number of iterations, because alignments were stored in memory for rapid filtering at each step during gradient descent.

Conclusions

We have presented an implementation, HapSolo, that is focused on improving primary assemblies by removing alternative haplotigs. In theory, the HapSolo package can be applied to any set of contigs from any assembly algorithm. The approach implemented in HapSolo is intended to replace laborious manual curation [31], and it follows some of the logic of existing programs, like Redundans [24], PurgeHaplotigs [25] and purge_dups [26]. However, HapSolo differs from competing programs by at least three substantive features relative to one or all of these programs. First, it utilizes multiple alignment metrics, so that it is not reliant only on percent identity (ID) score. The goal of these multiple metrics is to better discriminate among some situations that may lead to high identity scores but nonetheless lead to the retention of different contigs in the primary assembly (Figure 1). Second, when the gradient descent option is utilized, HapSolo relies on a maximization scheme based on BUSCO values. The underlying assumption is that maximizing the number of single-copy BUSCOs establishes more complete and less repetitive genomes. We emphasize that this is not an assumption that we devised but one common to the genomics community. Most new genomes are reported with BUSCO scores to reflect their completeness and quality. Third, an important feature of HapSolo is the ability to modify the cost function, so that the user may choose to (for example) weigh duplicated BUSCO genes less heavily or perhaps even ignore them altogether. This flexibility may prove useful for some applications. For example, it may be useful to ignore costs related to duplicated BUSCO genes when assembling recent polyploid genomes, which can be technically challenging due to the presence of numerous duplicated genes and regions.

We have illustrated some of the performance features of HapSolo by applying it to data from three species that differ in genome size and complexity: Chardonnay grapes, a mosquito species, and the Korean Mudskipper. The common feature of these species is that they are difficult to inbreed, such that diploid assembly is necessary. They were also assembled by different methods, including Falcon_Unzip for mosquito and mudskipper. For all three species, we compared the unreduced primary assembly to two HapSolo assemblies, one that used

default values (-GD) and one that used gradient descent minimization (+GD). Both HapSolo assemblies reduced genome size and markedly improved standard statistics like N50 (Table 1 and Figure 3). As expected, the +GD assembly was generally better than the -GD assembly, but the most substantial differences occurred between the unreduced assembly and either of the two HapSolo assemblies.

Our reduced assemblies scaffold faster than unreduced assemblies and also led to much better genomes. For each of our three species, the cumulative genome length associated with first *k* scaffolds (where *k* is the chromosome number) were much larger based on reduced vs. unreduced assemblies. The percentage of the genome contained in chromosome length scaffolds increased by at least 25% in every case (Table 3).We conclude, then, that some reduction step is critical for curating a primary assembly and for downstream scaffolding, even when the primary assemblies are from Falcon_Unzip [19] that has already (in theory but perhaps not in practice) identified secondary haplotigs. We further advocate for the use of gradient descent, because the computational cost is relatively small but the gains - while not always substantive - can be large (Figure 3).

METHODS:

Species and Data: We illustrated the performance of HapSolo using assemblies from three previously published assemblies of *Vitis vinifera* (cultivar Chardonnay) [13], *Anopheles funestus* (mosquito) [20], and *Periophthalmus magnuspinnatus* (Korean Giant-fin Mudskipper) [32]. The data were all based on the Pacific Biosciences (PacBio) sequencing platform and either provided by authors or downloaded from public databases. The chromosome number for each species were found in various sources [20, 32, 33]

Contig Alignments: For each genome, the pre-processing step prior to application of HapSolo was to perform pairwise contig alignments, as described above. For this study, we used Blat v35 [27], although in principle other aligners, like minimap2 [34] could be used. Blat was run with default options after the reference was compressed into 2bit format. As mentioned above, BLAT was run using each contig as a separate query to reduce run time. After running on individual contigs, the resulting PSL files were concatenated into a single PSL file for input into HapSolo.

Draft Assemblies: For each species, three assemblies were used for comparisons. The first was the original unreduced assembly, consisting of contigs; the second was a reduced assembly generated by providing HapSolo with default parameters of 0.70 for ID, Q and QR'; and the third was an assembly generated using HapSolo's parameter optimization, which

started with random values of ID, Q and QR' and then minimized *Cost* using gradient descent over 50,000 iterations.

Hi-C data and scaffolding: To demonstrate the advantage of HapSolo analyses for downstream scaffolding, we obtained short-read Hi-C data from online public databases for scaffolding [13, 20]) https://vgp.github.io/genomeark/Periophthalmus_magnuspinnatus/. The Hi-C sequencing data were mapped to their respective assemblies using BWA [27]. The scaffolding of raw assembly and HapSolo processed assemblies were processed using the 3D de novo assembly pipeline v180419 [35], available from https://github.com/theaidenlab/3d-dna/.

Computational Resources and Processing: For runtime analyses, HapSolo was run on dual CPU Intel E5-2696 V2 Nodes containing 512GB of RAM. The Blat pre-processing was run on these same nodes, as well as the UC Irvine High Performance Computing Cluster, Extreme Science and Engineering Discovery Environment (XSEDE) [36], the San Diego Supercomputer Center (SDSC) Comet and the Pittsburg Supercomputing Center (PSC) Bridges [37] clusters. We ran QUAST v4.5 [29] for our post processing example and to assess performance during program development.

ACKNOWLEDGEMENTS:

We thank A, Minio from the Cantù Lab at UC Davis for informing us of the haplotig issue in genome assemblies and Richard Kuo from the Roslin Institute, University of Edinburgh for coming up with the awesome name of HapSolo.

DECLARATIONS:

Ethics approval and consent to participate - Not Applicable

Consent for publication - Not Applicable

Availability of data and materials -

Vitis vinifera data: NCBI under the BioProject ID PRJNA550461.

Anopheles funestus data: NCBI under the BioProject ID PRJNA494870.

Periophthalmus magnuspinnatus data*:* Genbank ID GCA_009829125.1 https://vgp.github.io/genomeark/Periophthalmus_magnuspinnatus/

Software: https://github.com/esolares/HapSolo

Competing interests - The authors declare that they have no competing interests.

Funding - EAS is supported by a National Science Foundation (NSF) Graduate Research Program Fellowship under grant no. DGE-1321846. The work was supported by NSF grant NSF grant no. 1741627 to B.S.G. YT and ADL were supported by NIH grants R01OD010974 NIH R01GM115562. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562. Specifically, it used the Bridges system, which is supported by NSF award number ACI-1445606, at the Pittsburgh Supercomputing Center (PSC) under award no. TG-MCB180035.

Authors' contributions - EAS conceived of the approach, wrote the package and analyzed primary assemblies; YT performed Hi-C scaffolding; ADL and BSG provided ideas and feedback on the approach; EAS, ADL and BSG wrote the paper.

REFERENCES

1. Zhang X, Yazaki J, Sundaresan A, Cokus S, Chan SW-L, Chen H, et al. Genome-wide high-resolution mapping and functional analysis of DNA methylation in arabidopsis. Cell. 2006;126:1189–201.

2. Goff SA. A Draft Sequence of the Rice Genome (Oryza sativa L. ssp. japonica). Science. 2002;296:92–100. doi:10.1126/science.1068275.

3. C. elegans Sequencing Consortium. Genome sequence of the nematode C. elegans: a platform for investigating biology. Science. 1998;282:2012–8.

4. Adams MD. The Genome Sequence of Drosophila melanogaster. Science. 2000;287:2185–95. doi:10.1126/science.287.5461.2185.

5. Solares EA, Chakraborty M, Miller DE, Kalsow S, Hall KE, Perera AG, et al. Rapid low-cost assembly of theDrosophila melanogasterreference genome using low-coverage, long-read sequencing. doi:10.1101/267401.

6. Jiao Y, Peluso P, Shi J, Liang T, Stitzer MC, Wang B, et al. Improved maize reference genome with single-molecule technologies. Nature. 2017;546:524–7.

7. Daccord N, Celton J-M, Linsmith G, Becker C, Choisne N, Schijlen E, et al. High-quality de novo assembly of the apple genome and methylome dynamics of early fruit development. Nat Genet. 2017;49:1099–106.

8. Raymond O, Gouzy J, Just J, Badouin H, Verdenaud M, Lemainque A, et al. The Rosa genome provides new insights into the domestication of modern roses. Nat Genet. 2018;50:772–7.

9. Roessler K, Muyle A, Diez CM, Gaut GRJ, Bousios A, Stitzer MC, et al. The genome-wide dynamics of purging during selfing in maize. Nat Plants. 2019;5:980–90.

10. Price HJ. Evolution of DNA content in higher plants. Bot Rev. 1976;42:27.

11. Fierst JL, Willis JH, Thomas CG, Wang W, Reynolds RM, Ahearne TE, et al. Reproductive Mode and the Evolution of Genome Size and Structure in Caenorhabditis Nematodes. PLoS Genet. 2015;11:e1005323.

12. Ross PA, Endersby-Harshman NM, Hoffmann AA. A comprehensive assessment of inbreeding and laboratory adaptation in Aedes aegypti mosquitoes. Evol Appl. 2019;12:572–86.

13. Zhou Y, Minio A, Massonnet M, Solares E, Lv Y, Beridze T, et al. The population genetics of structural variants in grapevine domestication. Nat Plants. 2019;5:965–79.

14. Minio A, Lin J, Gaut BS, Cantu D. How Single Molecule Real-Time Sequencing and Haplotype Phasing Have Enabled Reference-Grade Diploid Genome Assembly of Wine Grapes. Front Plant Sci. 2017;8:826.

15. Massonnet M, Cochetel N, Minio A, Vondras AM, Lin J, Muyle A, et al. The genetic basis of sex determination in grapes. Nature Communications. 2020;11. doi:10.1038/s41467-020-16700-z.

16. Jain M, Koren S, Miga KH, Quick J, Rand AC, Sasani TA, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. Nat Biotechnol. 2018;36:338–45.

17. Chakraborty M, Baldwin-Brown JG, Long AD, Emerson JJ. Contiguous and accurate de novo assembly of metazoan genomes with modest long read coverage. Nucleic Acids Res. 2016;44:e147.

18. Vondras AM, Minio A, Blanco-Ulate B, Figueroa-Balderas R, Penn MA, Zhou Y, et al. The genomic diversification of clonally propagated grapevines. bioRxiv. 2019;:585869. doi:10.1101/585869.

19. Chin C-S, Peluso P, Sedlazeck FJ, Nattestad M, Concepcion GT, Clum A, et al. Phased diploid genome assembly with single-molecule real-time sequencing. Nature Methods. 2016;13:1050–4. doi:10.1038/nmeth.4035.

20. Ghurye J, Koren S, Small ST, Redmond S, Howell P, Phillippy AM, et al. A chromosome-scale assembly of the major African malaria vector Anopheles funestus. GigaScience. 2019;8. doi:10.1093/gigascience/giz063.

21. Koren S, Rhie A, Walenz BP, Dilthey AT, Bickhart DM, Kingan SB, et al. De novo assembly of haplotype-resolved genomes with trio binning. Nat Biotechnol. 2018. doi:10.1038/nbt.4277.

22. Rice ES, Koren S, Rhie A, Heaton MP, Kalbfleisch TS, Hardy T, et al. Continuous chromosome-scale haplotypes assembled from a single interspecies F1 hybrid of yak and cattle. Gigascience. 2020;9. doi:10.1093/gigascience/giaa029.

23. Koren S, Walenz BP, Berlin K, Miller JR, Bergman NH, Phillippy AM. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. Genome Res. 2017;27:722–36.

24. Pryszcz LP, Gabaldón T. Redundans: an assembly pipeline for highly heterozygous genomes. Nucleic Acids Res. 2016;44:e113.

25. Roach MJ, Schmidt SA, Borneman AR. Purge Haplotigs: allelic contig reassignment for third-gen diploid genome assemblies. BMC Bioinformatics. 2018;19:460.

26. Guan D, McCarthy SA, Wood J, Howe K, Wang Y, Durbin R. Identifying and removing haplotypic duplication in primary genome assemblies. Bioinformatics. 2020. doi:10.1093/bioinformatics/btaa025.

27. Li H, Durbin R. Fast and accurate long-read alignment with Burrows-Wheeler transform. Bioinformatics. 2010;26:589–95.

28. Simão FA, Waterhouse RM, Ioannidis P, Kriventseva EV, Zdobnov EM. BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs. Bioinformatics. 2015;31:3210–2.

29. Gurevich A, Saveliev V, Vyahhi N, Tesler G. QUAST: quality assessment tool for genome assemblies. Bioinformatics. 2013;29:1072–5.

30. Wolff J, Bhardwaj V, Nothjunge S, Richard G, Renschler G, Gilsbach R, et al. Galaxy HiCExplorer: a web server for reproducible Hi-C data analysis, quality control and visualization. Nucleic Acids Res. 2018;46:W11–6.

31. Chakraborty M, VanKuren NW, Zhao R, Zhang X, Kalsow S, Emerson JJ. Hidden genetic variation shapes the structure of functional elements in Drosophila. Nat Genet. 2018;50:20–5.

32. Rhie A, McCarthy SA, Fedrigo O, Damas J, Formenti G, Koren S, et al. Towards complete and error-free genome assemblies of all vertebrate species. bioRxiv. 2020;:2020.05.22.110833. doi:10.1101/2020.05.22.110833.

33. Jaillon O, Aury J-M, Noel B, Policriti A, Clepet C, Casagrande A, et al. The grapevine genome sequence suggests ancestral hexaploidization in major angiosperm phyla. Nature. 2007;449:463–7.

34. Li H. Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics. 2018;34:3094–100.

35. Dudchenko O, Batra SS, Omer AD, Nyquist SK, Hoeger M, Durand NC, et al. De novo assembly of the Aedes aegypti genome using Hi-C yields chromosome-length scaffolds. Science. 2017;356:92–5.

36. Towns J, Cockerill T, Dahan M, Foster I, Gaither K, Grimshaw A, et al. XSEDE: Accelerating Scientific Discovery. Computing in Science & Engineering. 2014;16:62–74. doi:10.1109/mcse.2014.80.

37. Nystrom NA, Levine MJ, Roskies RZ, Scott JR. Bridges: a uniquely flexible HPC resource for new communities and data analytics. In: Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure. New York, NY, USA: Association for Computing Machinery; 2015. p. 1–8.

TABLES:

Table 1: Contig assembly statistics for three primary assemblies for each of three species.

Species	Chardonnay			Mosquito			Mudskipper		
Assembly Type	No HapSolo[1]	HapSolo -GD[2]	HapSolo +GD[3]	No HapSolo	HapSolo - GD	HapSolo +GD	No HapSolo	HapSolo - GD	HapSolo +GD
# of Contigs	2,072	1,369	1,155	1,073	674	666	2,488	1,306	1,227
Contig Assembly Size (Mb)	655.188	569.333	539.045	212.022	200.414	199.965	979.058	789.477	771.500
Largest Contig (Mb)	11.612	11.612	11.612	7.561	7.561	7.561	8.646	8.646	8.646
Contig N50 (Mb)	1.066	1.335	1.441	0.630	0.689	0.689	1.043	1.321	1.327
Contig L50	141	106	95	86	77	77	259	180	175

[1] Results in this column were based on the primary assembly without application of HapSolo

[2] Results in this column were based on application of HapSolo without gradient descent (-GD) and with default parameters of ID, Q and QR = 0.70.

[3] Results in this column were based on application of HapSolo with 50,000 cycles of gradient descent (+GD).

Table 2: Starting and Ending BUSCO values for the three species for primary contig assemblies.

	No HapSolo		HapSolo (-GD)		HapSolo (+GD)	
Species	GS[1]	BUSCO[2]	GS	BUSCO	GS	BUSCO
Grape	655.2	C:1642 S:1211 D:431	569.3	C:1344 S:1224 D:120	539.0	C:1357 S:1205 D:55
Mosquito	212.0	C:2640 S:2493 D:147	200.4	C:2609 S:2548 D:61	200.4	C:2621 S:2566 D:55
Mudskipper	979.1	C:4333 S:3567 D:766	790.8	C:4251 S:3934 D:317	771.5	C:4228 S:3929 D:299

[1] Genome Size (GS) based on the sum of all contigs for the primary assembly.

[2] Busco based on all contigs prior to the application of HapSolo. The four values represent the complete (C), the single (S) and duplicated (D) BUSCO genes.

Table 3: Scaffolded assembly statistics after Hi-C analysis, for three primary assemblies for each of three species.

Species	Chardonnay			Anopheles funestus			Korean Giant-fin Mudskipper		
Assembly Type	No HapSolo[1]	HapSolo -GD[2]	HapSolo +GD[3]	No HapSolo	HapSolo -GD	HapSolo +GD	No HapSolo	HapSolo -GD	HapSolo +GD
# of Scaffolds	1,332	2,748	2,403	1,211	603	611	5,362	2,651	2,033
% of Genome in *k* largest scaffolds[4]	38.8%	76.3%	76.6%	58.3%	68.0%	85.6%	10.4%	85.8%	86.8%
% of Genome in Scaffolds > 10Mb	33.0%	80.7%	85.1%	88.0%	94.3%	93.9%	28.2%	85.7%	86.8%
Scaffold Assembly Size (Mb)	656	570	540	213	201	200	981	791	773
Largest Scaffold (Mb)	19.1	32.6	36.5	43.6	53.0	95.0	27.4	34.1	34.6
Scaffold N50 (Mb)	7.2	23.5	20.7	37.9	41.5	41.6	1.7	28.1	28.3
Scaffold L50	28	11	11	3	3	2	71	13	13

[1] Results in this column were based on the primary assembly without application of HapSolo

[2] Results in this column were based on application of HapSolo without gradient descent (-GD) and default parameters of ID, Q and QR.

[3] Results in this column were based on application of HapSolo with 50,000 cycles of gradient descent (+GD).

[4] Percentage of genome in the largest *k* scaffolds, where *k* is equal to the number of chromosomes expected for each species.

Table 4: Run time and performance statistics for gradient descent analyses

	Cores	Iterations	Total Iterations	Avg. Time (h:m:s)[1]	RAM (GB)
Chardonnay	1	5,000	5,000	00:55:34	12.2
	10	5,000	50,000	01:26:43	12.2
Mosquito	1	5,000	5,000	00:46:32	8.8
	10	5,000	50,000	01:01:16	8.8
Mudskipper	1	5,000	5,000	04:39:28	54.7
	10	5,000	50,000	06:45:40	54.7

[1] h = hour, m = minute, s = seconds

FIGURE LEGENDS

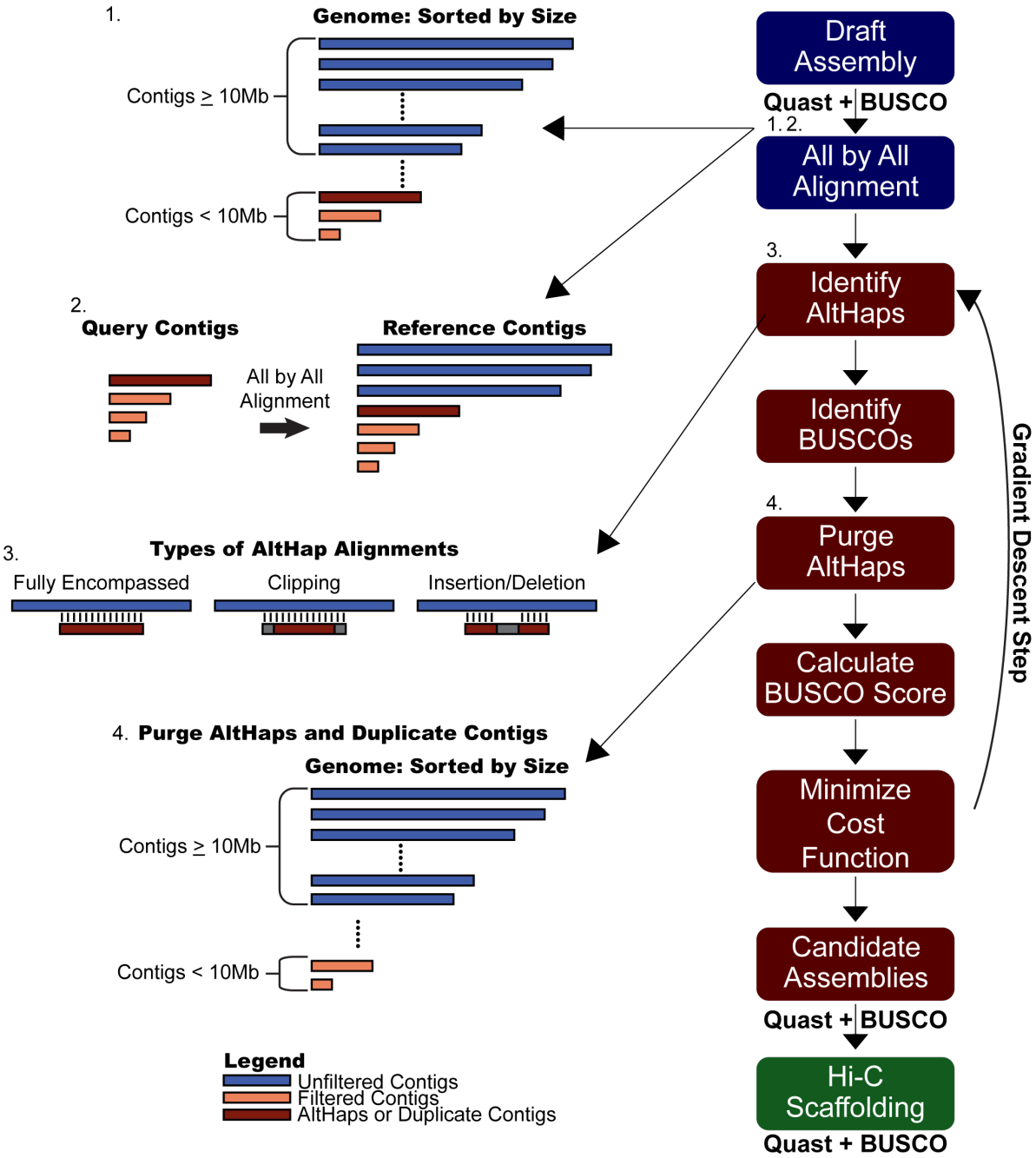


Figure 1: A schematic showing the basic workflow and ideas behind the approach. The rectangles on the right illustrate the basic steps, including pre-processing (blue rectangles),

steps within HapSolo (black rectangles) and post-processing (green rectangle). Some of the HapSolo steps include iterations to perform gradient descent calculations, as described in the text and shown by the arrow. On the left, step 1 shows the contigs from the primary assembly, and step 2 illustrates the all-by-all alignment of contigs. Step 3 provides examples of some properties of potential alignments. Our metrics - ID, Q and QR - were defined to help capture some of the variation among these conditions. Step 4 illustrates that new primary assemblies are formed by dropping putative secondary contigs.

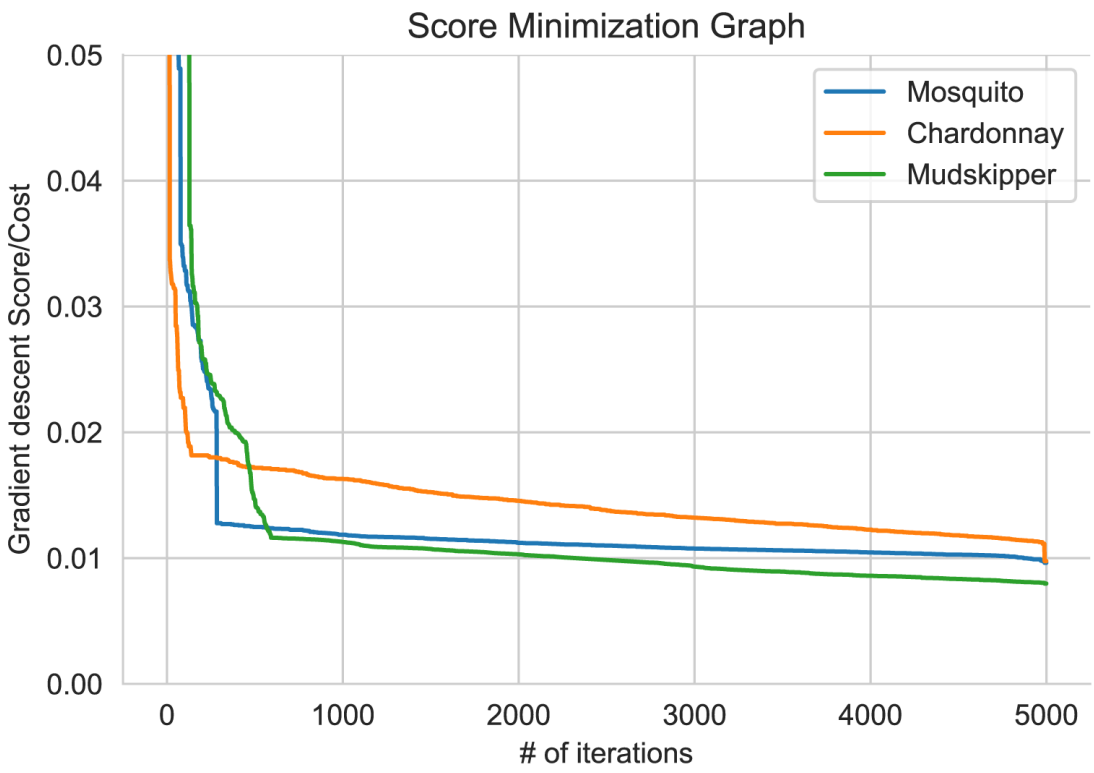


Figure 2: A graph of the performance of gradient descent over 5,000 iterations, with a normalized *Cost* on the y-axis and the number of iterations on the x-axis. The most dramatic improvements in *Cost* occurred within the first 1,000 iterations, although *Cost* continued to improve slowly from 1,000 to 5,000 iterations. For most of our analyses with GD, we performed 5,000 iterations on each of 10 cores; here we are showing results from one core. The top of the graph was removed in order to have a better visualization of the minimization of the cost function via purging of contigs using different parameters for ID, Q and QR. Costs were normalized by dividing the cost with the maximum cost observed for each respective assembly. This graph represents the cost minimization on one CPU core.

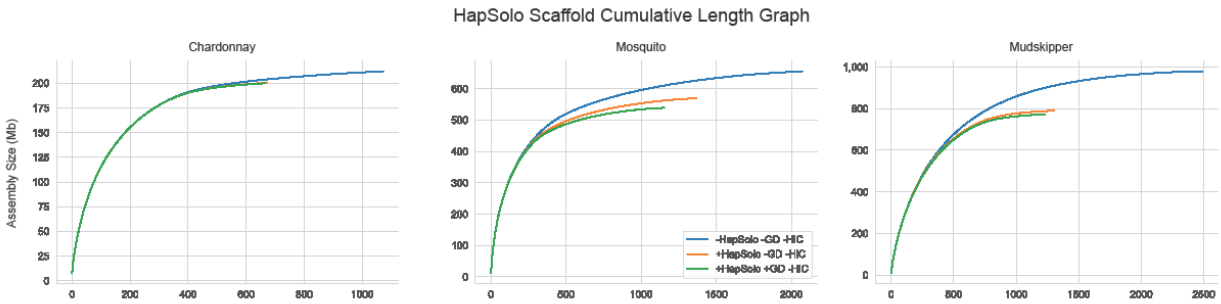


Figure 3: The cumulative assembly size (cdf) based on contigs. For each species three different reduction strategies are depicted: an unreduced assembly, HapSolo applied with default parameter values and no gradient descent (-GD), and HapSolo with random starting values and 50,000 iterations of gradient descent (+GD).

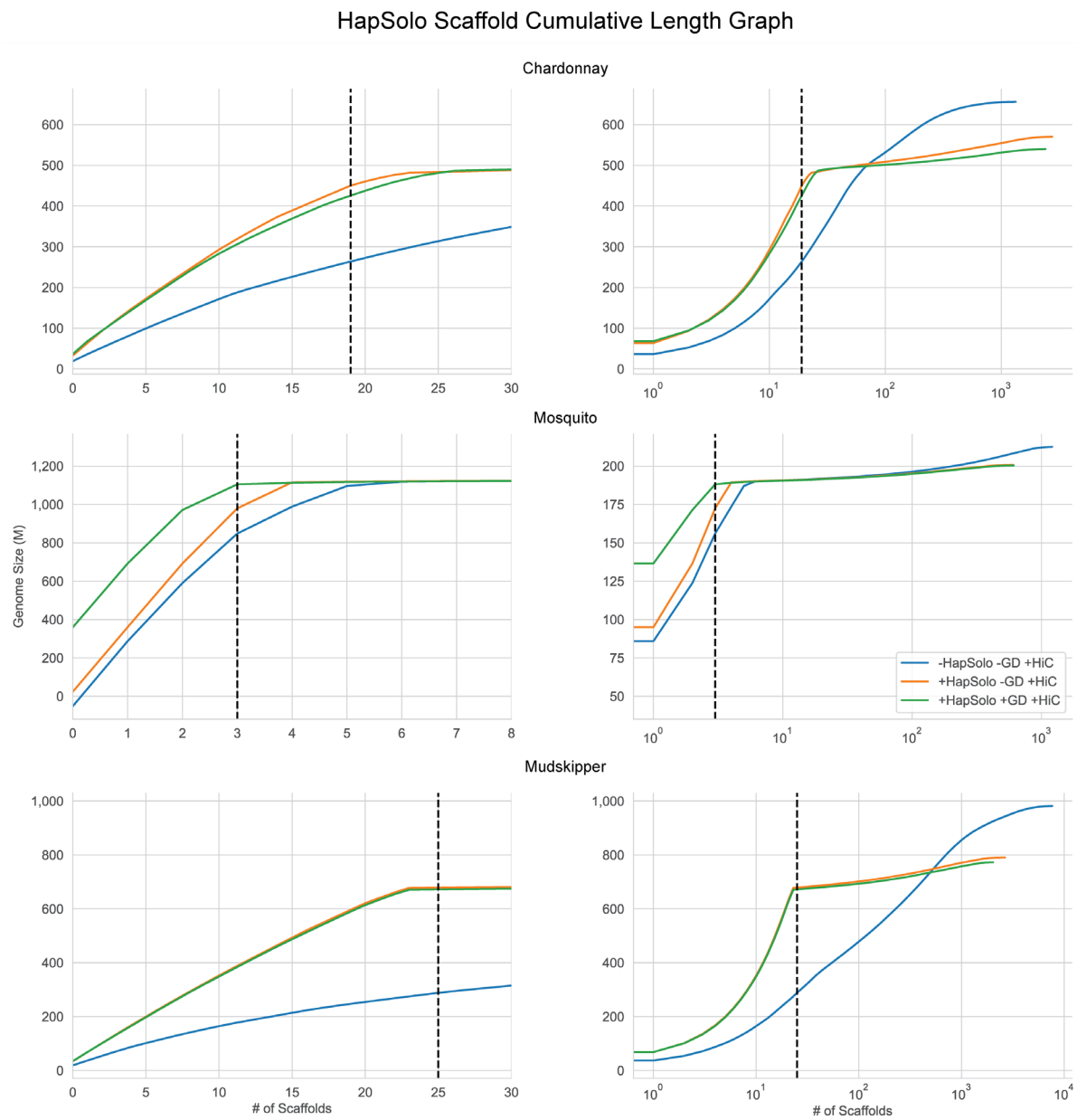
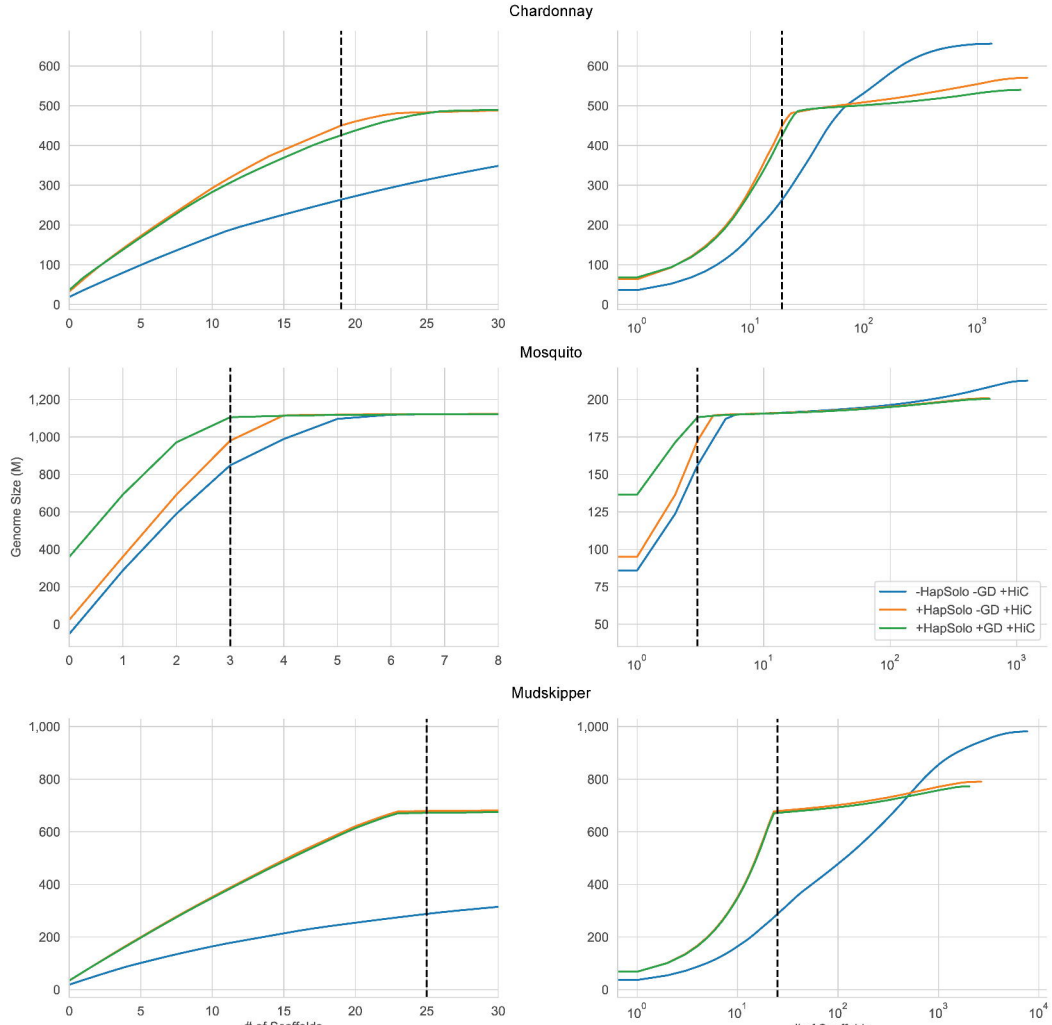
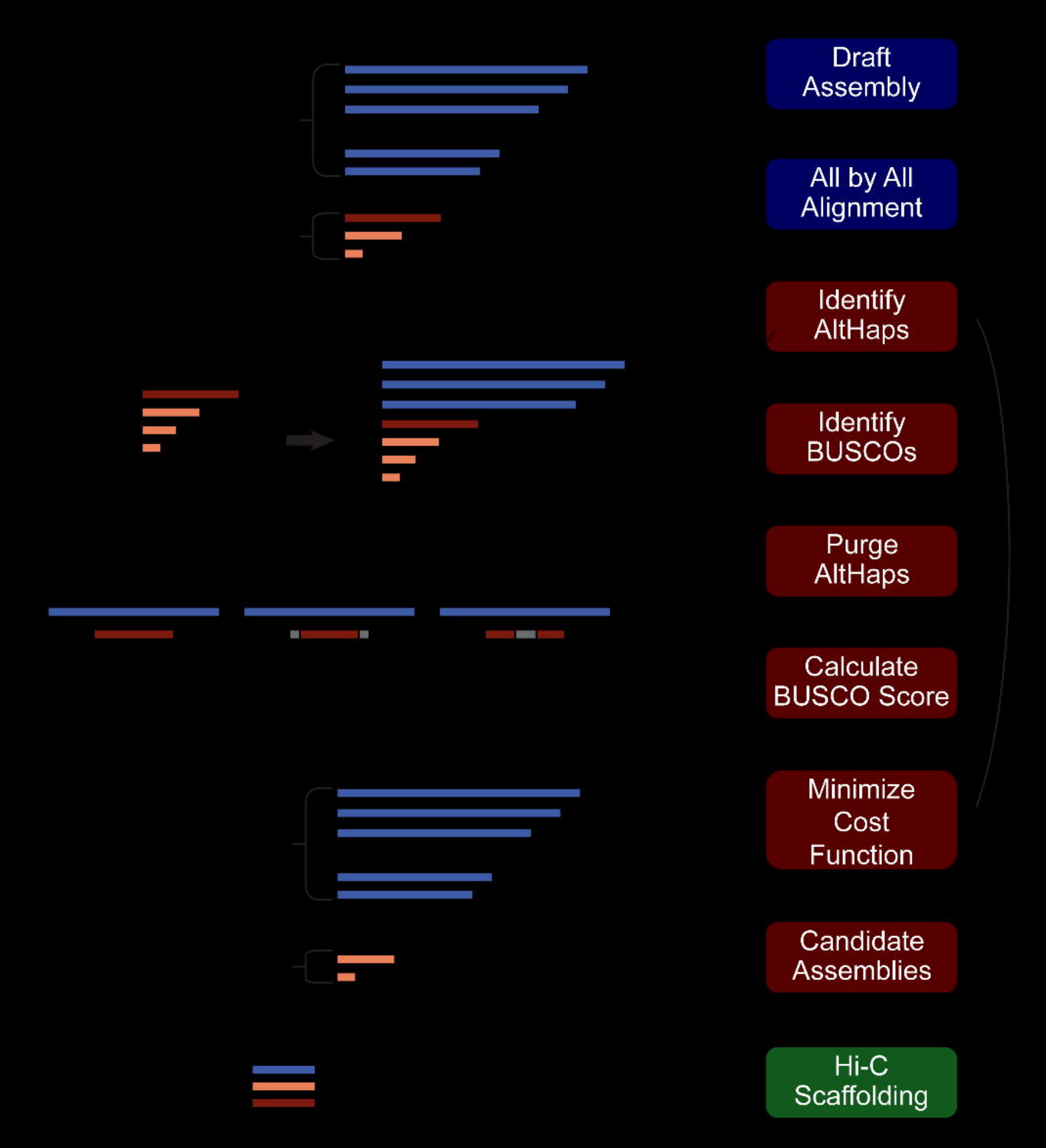
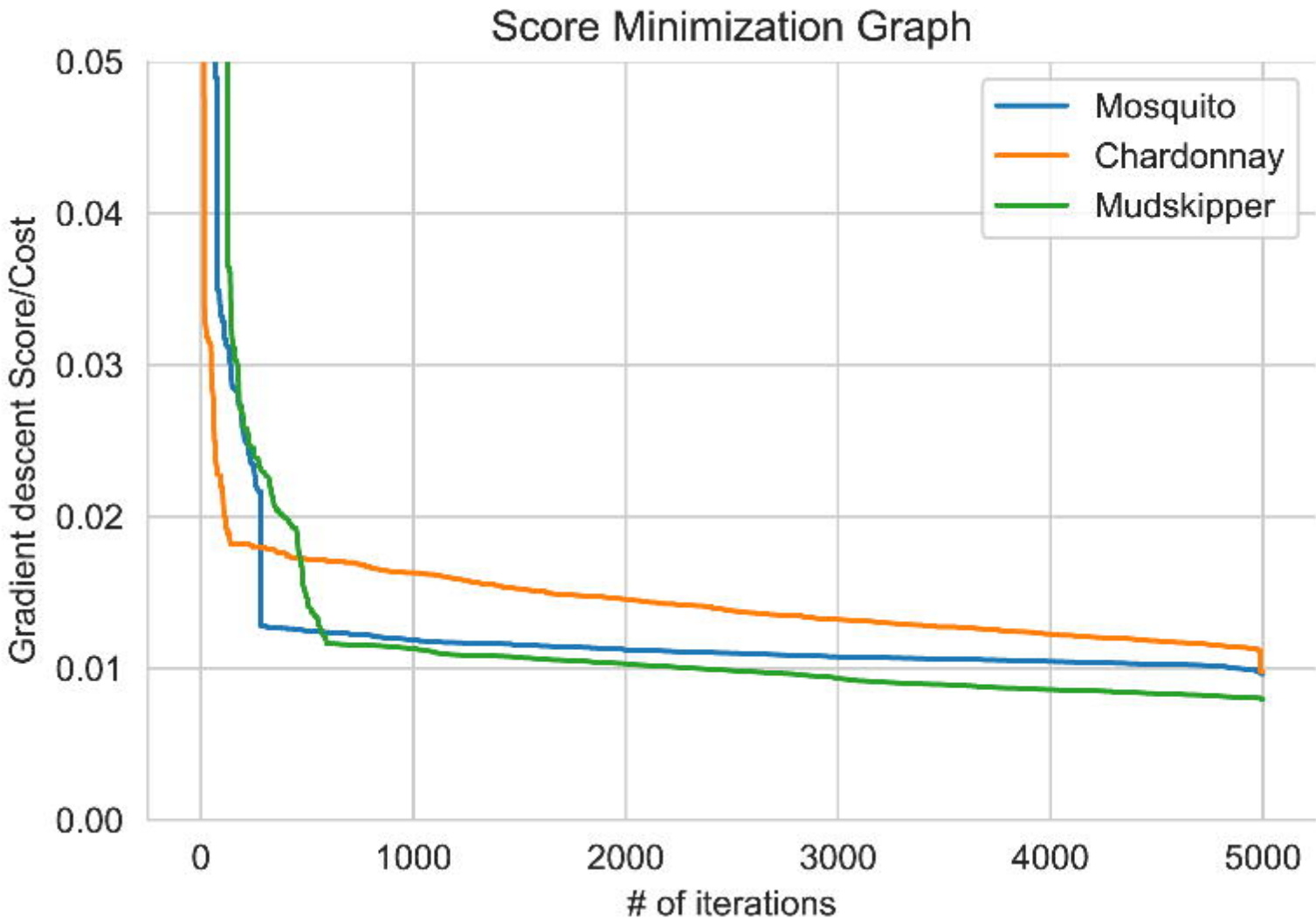


Figure 4: The cumulative assembly size (cdf) based on scaffolds for a different species in each row. There are two graphs for each species; the one on the left focuses on the chromosome length scaffold portion of the assembly, (number of scaffolds), while the one on the right is the complete assembly on a $\log_{10}$ (number of scaffolds) scale. Each graph has three lines, representing the scaffolded genome based on the unreduced assembly (-HapSolo -GD +HiC), scaffold based on the HapSolo reduced assembly without gradient descent (+HapSolo -GD +HiC), and the scaffold based on the HapSolo reduced assembly with gradient descent

(+HapSolo +GD +HiC). In all graphs, the dotted line indicates the estimated number of chromosomes for the species.


Chardonnay
Mosquito
Mudskipper
Genome Size (M)
# of Scaffolds
-HapSolo -GD +HiC
+HapSolo -GD +HiC
+HapSolo +GD +HiC


Draft Assembly
All by All Alignment
Identify AltHaps
Identify BUSCOs
Purge AltHaps
Calculate BUSCO Score
Minimize Cost Function
Candidate Assemblies
Hi-C Scaffolding


Score Minimization Graph
Mosquito
Chardonnay
Mudskipper
Gradient descent Score/Cost
0.05
0.04
0.03
0.02
0.01
0.00
0
1000
2000
3000
4000
5000
# of iterations