Chapter 2

From Sequence Mapping to Genome Assemblies

Thomas D. Otto

Abstract

The development of "next-generation" high-throughput sequencing technologies has made it possible for many labs to undertake sequencing-based research projects that were unthinkable just a few years ago. Although the scientific applications are diverse, e.g., new genome projects, gene expression analysis, genome-wide functional screens, or epigenetics—the sequence data are usually processed in one of two ways: sequence reads are either mapped to an existing reference sequence, or they are built into a new sequence ("de novo assembly"). In this chapter, we first discuss some limitations of the mapping process and how these may be overcome through local sequence assembly. We then introduce the concept of de novo assembly and describe essential assembly improvement procedures such as scaffolding, contig ordering, gap closure, error evaluation, gene annotation transfer and ab initio gene annotation. The results are high-quality draft assemblies that will facilitate informative downstream analyses.

Key words Mapping, De novo assembly, Assembly improvement, Local assemblies, Bin assemblies, Annotation

1 Introduction

The aim of sequence assembly is to join short sequences of nucleotides (sequence reads 35–1,000 bp in length) into contiguous sequences (contigs) that represent the sequenced DNA. Sequence assembly is needed when no reference genome is available, or when the sequenced DNA is too different from a potential reference genome. In contrast, when strains or isolates are similar enough to a reference sequence, reads can be mapped against this reference by finding the unambiguous place where an alignment generates the highest score for a given read, similar to a BLAST search. Figure 1 shows an example of reads from the *Plasmodium falciparum* IT clone mapped to the *MSP3* (Merozoite surface protein) gene of the reference genome. Most of the regions are covered by mapped reads and genetic variation is represented by red lines in the alignments. But some regions are too polymorphic for reads to map. In this case, only the comparison of the reference with the de novo assembly reveals an insertion in the *MSP3* gene in the IT strain.

Christopher Peacock (ed.), *Parasite Genomics Protocols*, Methods in Molecular Biology, vol. 1201,
DOI 10.1007/978-1-4939-1438-8_2,

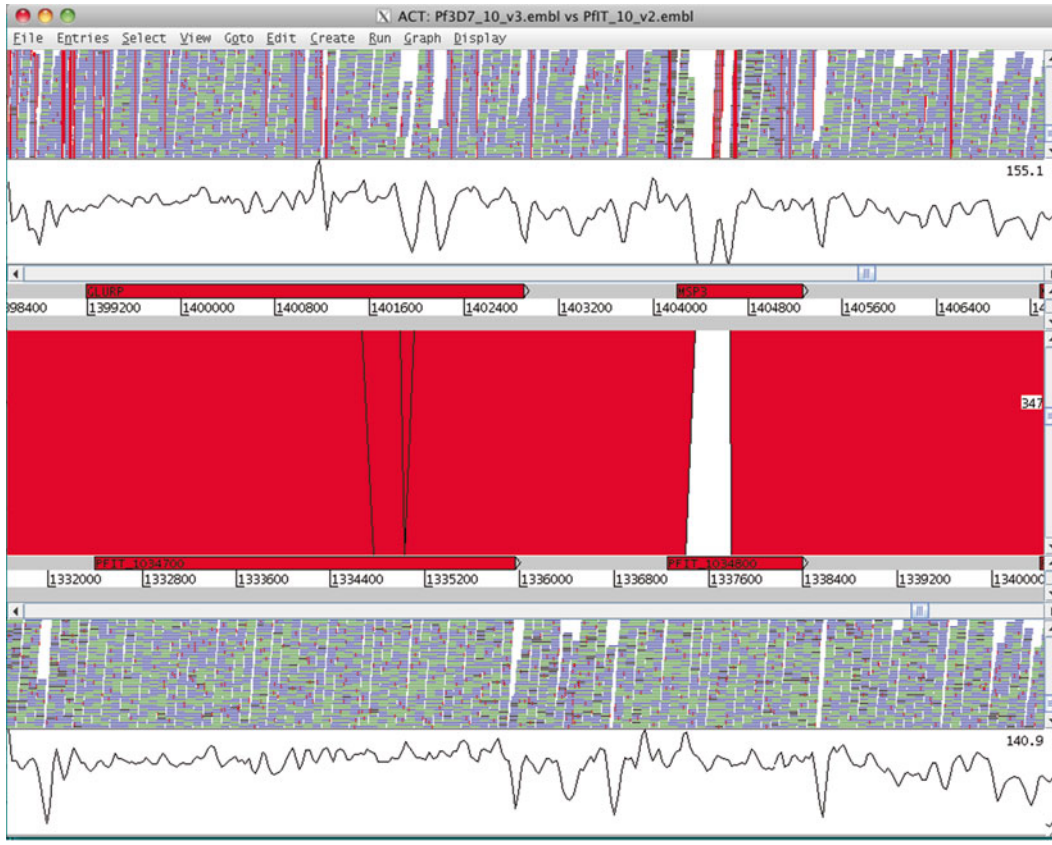

Fig. 1 Mapping versus assembly. Two genes of *P. falciparum* 3D7 (*red boxes*) can be seen at the top. The *horizontal green* and *blue lines* are mapped sequencing reads from the IT clone. *Red points* in the reads are differences between the IT reads and the 3D7 reference. The lower part shows the *de novo* assembly of IT. The *vertical bars* are blast hits. The graphs are the coverage plots. Some regions of *MSP3* in 3D7 are not covered by mapped IT reads. The de novo assembly has an insertion, indicated by the shape of the blast hit. Reads map even over this new assembled region

The basic idea of sequence assembly can be summarized as follows. First, all the reads are compared against each other to find shared identical sequences, as is done by the programs like CAP3 [1] and Celera [2]. Next, through joining reads by their overlaps (identical sequence) the consensus sequence, usually in discrete sequences called contigs, is generated (Fig. 2a). But due to the high number of reads generated through recent sequencing technologies, the step of comparing reads to each other takes too much time to be practical. One way around this is to use a more efficient representation of read similarity. Instead of looking for overlaps, it is more efficient to index all words of a specific length (k-mers) in all reads. Then an algorithm can generate contigs by traversing a graphical representation (de Bruijn graph) of the k-mers. Many high-throughput read assemblers use this approach, like ABYSS [3], Velvet [4] or see CITATION 1 in work document. A good introduction to the de Bruijn graph is

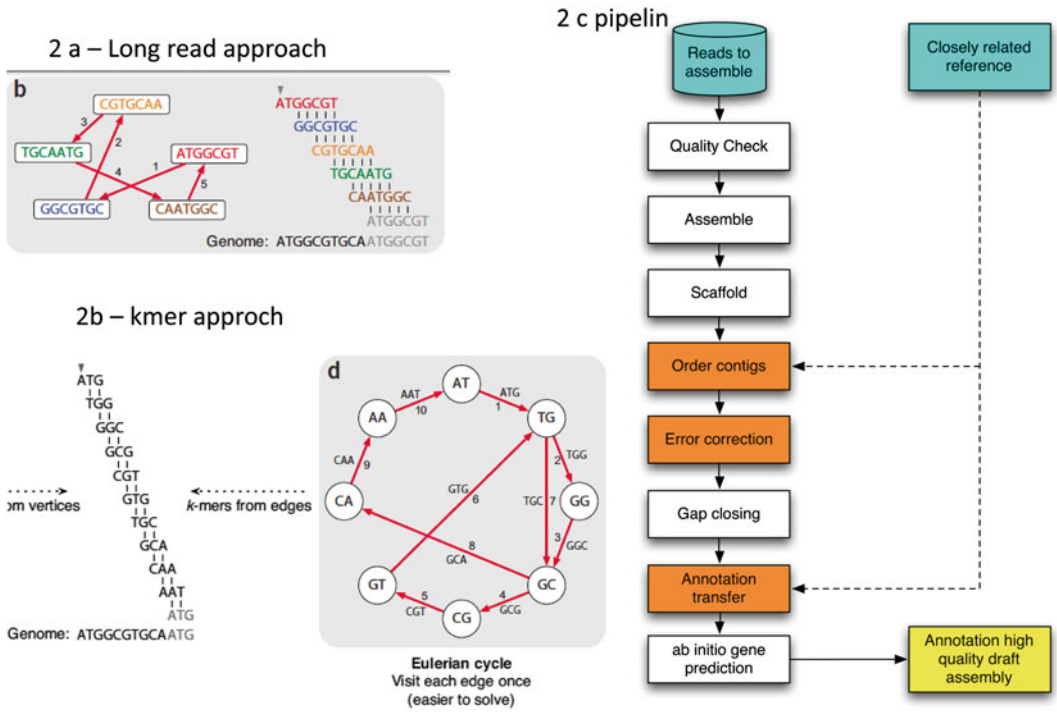

Fig. 2 (**a**) Assembly with longer reads: Nearly identical overlap between reads enable the generation of the consensus. (**b**) Assembly with short reads, using de Bruijn graph: First the reads are index and the k-mer are stored in a hash table, including the k-mer and the frequency. With a k-mer length of 3 the k-mer TCG is non unique. Due to this non unique k-mer, the graph quite complicated. (**c**) Overview of typical pipeline for de novo assembly and annotation

[5]. Unfortunately, assemblers rarely if ever generate one contig per chromosome using short reads. The causes are usually repetitive sequences, uneven or the complete lack of reads for particular genome regions [6]. For example, reads from different copies of a repeat will collapse into one contig, rather than into separate copies. To improve the contiguity of assemblies, large insert size libraries can be used to bridge the difficult regions and join contigs into scaffolds (also called supercontigs) [2, 7, 8]. The limitation is the insert size, i.e., the distance between the paired reads, which determines the size of a problematic region that can be bridged.

If a sequence is reasonably similar to a reference, scaffolds can be further joined by ordering them against the reference [9, 10]. Comparing against a reference helps to reveal where the genomes are different, such as synteny breakpoints, insertions/deletions, or differences in gene content. Other post-assembly improvements are to close sequencing gaps (in the scaffolds) [11, 12] and to correct single-base errors [13, 14]. Methods for fixing the latter are based on mapping the reads against the assembly. The distance between the two mates of a mapped read pair can also be used to identify assembly errors [3]. If gene annotation (i.e., the positions of exons and introns) is available for the reference this annotation

can be transferred to the new assembly in regions where the two genomes are syntenic [15]. For regions of the new assembly without synteny, ab initio gene prediction and function annotation must be done. The resulting genome models can be merged with the transferred gene models.

In this chapter we present methods to perform local assemblies (for example of single genes), an assembly of unmapped reads (a so-called bin assembly, for example of very diverse gene families), and a complete assembly of genomes. Further, we describe methods to improve the quality of an assembly and do a first pass annotation.

2 Materials

2.1 Installation and Resources

Bioinformatics analysis, especially in the assembly process, requires not only appropriate computers, but also the right environment with many installed tools and the knowledge of how to run them. This chapter should help you to understand and apply the different tools. To do so, we generated a tarball that contains all the needed software packages (Table 1) of the work described here. This protocol is designed to work with the Linux operating system. To facilitate the application of the protocol, we generated a test data set that can be used to go through the different steps. Finally, you will need to bear in mind how much memory your computer will need to process the data. For genomes up to 5 Mb we would recommend up to 6 GB of memory. Genomes of 20, 100, and 200 Mb require up to 20, 200, and 500 GB of memory. Those numbers will vary depending on the structure of the genome, the quality of the reads, the software used, and preprocessing of the reads. For the presented example the computer would require around 2 GB of memory.

2.1.1 How to Install the Programs

The best way is to download the latest version of the programs from their web sites (Table 1) and install them. Nevertheless need to download the tar ball, see blow, as it contains some custom scripts used in this chapter. The custom scripts from the tarball, which are described in Table 2.

Alternatively it is possible to use a preinstallation, where all tools are already installed and the necessary dependencies are set. The requirement is a 64 bit Linux operating system. If this is the case, do following steps to download and install it. Switch to the bash shell and create a directory in which to install the software:

```
$ bash
$ mkdir -p ~/bin/Assembly
$ cd ~/bin/Assembly
```

Next download the file contains the software from the ftp server. This file has to be extracted in a directory and the system-wide variables have to be set:

Table 1
Description of the tools used in this chapter

Name	Description	
Read quality		
SGA [19]	String graph assembler that has functions to quality trim and correct reads	https://github.com/jts/sga
Trimmomatic [8]	Trims adapter from sequences	www.usadellab.org/cms/?page=trimmomatic
Mappers		
SMALT	Maps reads to a reference	ftp://ftp.sanger.ac.uk/pub4/resources/software/smalt/
SAMTOOLS [20]	Processes alignment files (SAM/BAM)	http://samtools.sourceforge.net/
Assemblers		
Velvet[4]	Assembler based on de Bruijn graphs	http://www.ebi.ac.uk/~zerbino/velvet/
Post-assembly genome improvement		
REAPR [3]	Assesses quality of sequences and can break assemblies	ftp://ftp.sanger.ac.uk/pub4/resources/software/reapr/
SSPACE [7]	Scaffolder	http://www.baseclear.com/landingpages/basetools-a-wide-range-of-bioinformatics-solutions/sspacev12/
ABACAS [9]	Tools to order contigs against a reference sequence	http://abacas.sourceforge.net/
IMAGE [11]	Closes sequencing gaps and extends contigs by local assembly	http://sourceforge.net/projects/image2/files/
ICORN [13]	Corrects 1–3 bp errors in sequences	http://icorn.sourceforge.net/
PAGIT [24]	Toolkit that joins ABACAS, IMAGE, ICORN, and RATT	http://www.sanger.ac.uk/resources/software/pagit/
Annotation		
RATT [15]	Transfers annotation from a reference to a query, based on synteny	http://ratt.sourceforge.net/
AUGUSTUS [23]	Gene prediction software for Eukaryotic organisms	http://augustus.gobics.de/binaries/
Glimmer [22]	Gene prediction software for bacteria	http://www.cbcb.umd.edu/software/glimmer/
Prokka [7]	Software to annotate bacterial genomes	http://vicbioinformatics.com/

```
$ wget
ftp://ftp.sanger.ac.uk/pub/resources/software/
pagit/ParasiteProtocols.tgz
$ tar xzf ParasiteProtocols.tgz
$ ./installme.sh
```

Table 2
Custom scripts from the tarball

Scripts from the tarball	
stats	Returns assembly statistics
map.smalt.sh	Wrapper script for smalt
revcompFastq.pl	Reverse complements fastq files
sga2readpair.pl	Generates two paired fastq files from a merged fastq file (SGA correction output)
deNovoPlus.sh	Script to run the assembly and the correction step in one call
gff2gb.sh	Transforms an Artemis gff to a genbank file
augustusAnnotate.sh	Takes an Augustus gtf and annotates it with the first blast hit as embl file
annotation.MergeAnnotationSecondAway.pl	Joins gene models as embl, and excludes a model from the second set, if it overlaps with a model from the first set
excludeGeneEMBL.pl	Deletes gene models in an EMBL file from a given list.
annotation.giveIDCDS.pl	Generates automatically geneIDs
AllCommands.sh	All the commands used in this chapter, adapted to the latest version of the software and correct for possible errors
RemoveSequencesSmaller.pl	Removes fasta entries that are smaller than a given parameter
BAM2consensus_reads.pl	Script that takes mapped reads and returns reads with consensus sequence

Each time you want to run one of the programs, do following step:

```
$ source ~/bin/Assembly/sourceme.sh
```

Alternatively, include the last command at the end of the ~/.bashrc system file.

2.1.2 Software

There are several software components installed in the package, which are summarized in Table 1. They are ordered by groups: read processing tools, assemblers, scaffolders, post-assembly improvement tools, annotation tools, and custom Perl scripts that will be needed in this protocol, Table 2. All the tools will be discussed in detail through this chapter.

2.1.3 Test Dataset

To help the user to understand the protocol, we included a test dataset for each section. The data are from a re-sequencing project, concerning the Malaria parasite *P. falciparum*. Here we only consider chromosome 10 of the IT clone. The complete genome

can be found on Gene DB [16]. As reference we will be using the 3D7 clone.

To work with the data change to the directory:

```
$ cd $ASSEMBLY_HOME/testdata
$ ls
```

The test reads are called reads_1.fastq, reads_2.fastq, reads_3k_1.fastq, and reads_3k_2.fastq. The reference chromosome 10 from 3D7 is called ref.fa. There are four scripts: Mapping.sh, LocalAssembly.sh, BinAssembly.sh, and deNovoAssembly.sh. Type:

```
$ cat *.sh
```

to see all the commands

2.2 Sequencing Technology

Several sequencing technologies are available to date but it is not our aim to discuss them here [4]. For a successful assembly the reads should ideally have the following properties:

1. Be fairly uniformly distributed across the genome sequence.
2. Have enough coverage of the genome, i.e., 80× coverage with 100 bp reads. Longer reads will reduce the coverage need.
3. Read pair information seems to be vital to make a good assembly. The fragment size should be around 500 bp.
4. Large insert size libraries will help to scaffold more complex and larger genomes. Those libraries are called from time to time also mate pairs.

This protocol should be applicable to sequences of the length of 76–250 bp in sufficient depth, as provided by SOLID, Illumina, or Ion torrent. For scaffolding the large insert size libraries (8/20 kb) of the 454 technology are also helpful.

The importance of uniformity of the read distribution is often underestimated. This means that the amount obtained from each region of the genome should be similar. But due to PCR amplification steps, extreme GC content is amplified differently. This can result in uneven coverage that hinders the performance of assemblers. The following publication might be useful for further details [17]. In our experience, good DNA quality and good library preparation are the most crucial steps for a good de novo assembly.

3 Methods

Here we describe the methods of the sections: read preprocessing, mapping, local assembly, de novo assembly, and annotation. For most of the step we provide further information in Subheading 4.

3.1 Read Preprocessing

Before the reads can be used for assembly, sequencing adapters have to be trimmed. Bad quality regions of reads could also be trimmed. When not enough coverage is available (<80×), it is advisable to correct the reads. However, we would recommend trying the assembly without read correction first. If the assembly fails due to runtime and memory requirements, you should use the read correction.

1. To clip the reads for adapter you can use a program like trimmomatic [8]. Assume that your reads are called reads_1.fastq

   ```
   $ java -Xmx1000m -jar $PAGIT_HOME/trimmomatic-0.32.jar PE reads_1.fastq reads_2.fastq trimmed_1.fastq trimmo_unpaired_1.fq trimmed_2.fastq trimmo_unpaired_2.fq ILLUMINACLIP:adapters.fasta:2:10:7:1:MINLEN:50
   ```

   Though the call looks a bit long, the reads with the trimmed adapters are in the files trimmed_1.fastq trimmed_2.fastq. The file "adapters.fasta" contains the adapters used in the sequencing process.

2. To cut low-quality ends of reads, it is possible to use the program "preprocess" from the assembler SGA [19].

   ```
   $ sga preprocess -m 51--ermute-ambiguous -f 3 -q 3 -p 1 reads_1.fastq reads_2.fastq>reads_trimmed.fastq
   ```

3. To correct reads from sequencing errors, SGA also has a function. But first the reads have to be indexed.

   ```
   $ sga index reads_trimmed.fastq
   $ sga correct -k 51 -x 5 -o
   reads_corrected.fastq reads_trimmed.fastq
   ```

   Here, a k-mer of 41 bp in the reads (-k) that occurs less than five times (-x), will be corrected to a k-mer that occurs with the expected frequency.

4. The output of SGA will be one merged file, where reads might have been discarded due to general bad quality. To generate again forward and reverse reads (or read one and two), use following command:

   ```
   $ sga2readpair.pl reads_corrected.fastq reads_corr
   ```

3.2 Mapping the Reads

A good first analysis step is to map the reads against a closely related reference (i.e., 90 % nucleotide identity), if one exists. Here we are going to use the mapper SMALT. The final output of this mapping process will be a BAM file, which contains all the reads, including their sequence and quality, as well as mapping information (*see* Fig. 3). Assuming your reference is called ref.fa and your reads reads_1.fastq and reads_2.fastq, the following steps need to be done:

1. For pragmatic reasons, it might be good to link the name of your files to the names used here:

   ```
   $ ln -s YOUR_REFERENCE ref.fa
   $ ln -s YOUR_READS_1.fastq reads_1.fastq
   $ ln -s YOUR_READS_2.fastq reads_2.fastq
   ```

2. To see the differences between the fasta and the fastq format (Fig. 3) use the Linux command "head":

   ```
   $ head ref.fa reads_1.fastq
   ```

3. First your reference has to be indexed:

   ```
   $ smalt index -k 17 -s 3 ref.fa ref.fa
   ```

   Here, all words (so-called k-mer) of 13 bp in the genome are counted and stored in an efficient way through hashing. A short summary of the number of k-mers and the runtime is given. The output will be two files: ref.fa.sma and ref.fa.smi.

4. Comparing those k-mers between the reference and the reads, the mapper is able to decide where in the genome each read could be placed.

   ```
   $ smalt map -i 1000 -m 50 -r 0 -f samsoft -o
   Mapped.sam ref.fa reads_1.fastq reads_2.fastq
   ```

a

```
>Pf3D7_10_v3
taaaccctgaaccctaaaccctgaaccctaaaccactaaccctaaaccctgaaccctgaa
ccctgaaccctaaaccctaaaccctaaaccctgaaccctaaaacctaaaccctgaaccctg
aaccctaaaccctaaaccctaaaccctaaaccctgaaccctaaaccctaaaccctgaacc
ctaaaccctgaaccctgaaccctaaaccctgaaccctgaaccctaaaccctgaaccacta
```

b

```
@IL39_6014:8:110:3699:4595#3/1
TATTTGAACTGACAATTTTATAAGATCCATATATATGAAGATCTCAAAAAAATATATGTTTTTTTGAAAATTTTCA
+
CCB@@*CCC<GCHHEHGGGGGEEGDDGGGGGEGGHEGCEGGFFCHGDBCB?BH?GGBH=BEBB@@E=B>=EBECGA
@IL39_6014:8:88:4857:8768#5/1
TATTTAAACTGACAATTTTATAAGATCCATATATATGAAGATCTCAAAAAAATATATGTTTTTTTGAAAATTTTCA
```

c

```
IL39_6014:8:110:3699:4595#3   83        PfIT_10_v2          1404405   60        76M
          =          1404336   -145
          TGAAAATTTTCAAAAAAACATATATTTTTTTGAGATCTTCATATATATGGATCTTATAAAATTGTCAGTTCAAATA
          AGCEBE=>B=E@@BBEB=HBGG?HB?BCBDGHCFFGGECGEHGGEGGGGGDDGEEGGGGGHEHHCG<CCC*@@BCC
          AS:i:73
```

Fig. 3 Examples of different file formats. (**a**) fasta: Each sequence starts with a ">" and a name. Then the sequence is followed. (**b**) fastq: Similar to fasta, but with the quality coded in ASCII. (**c**) SAM format: First column is the name of the read. Next column is the mapping flag that can be used for querying a BAM file. Third and fourth, seven and eight columns are mapped to the reads and its mate, respectively. Column nine is the fragment size. The information how well the reads map is in column five and six, mapping quality and cigar string, respectively. The sequence and the quality of the reads are stored in column ten and eleven. The last column can have many different information, like an alignment score, other possible position to map repetitively. This depends on the mapper

As parameters you can set the maximum expected fragment size for a read pair to be properly paired (-i), place reads repetitively (-r), and exclude reads that map with a lower Smith-Waterman alignment score than 50 (-s). The reads are stored in the file Mapped.sam in SAM format [20] (-o -f samsoft). An example of the SAM format can be seen in Fig. 3. It is a well-defined format, including for each read how and where it is mapped (column 2–6), where its mate is mapped, the sequence of the read and its quality and finally some tags.

5. If you want to map a large insert library (more than 1 kb), first the reads have to be reverse complemented.

```
$ revcompFastq.pl reads_3k_1.fastq rev.reads_3k_1.fastq
$ revcompFastq.pl reads_3k_2.fastq rev.reads_3k_2.fastq
```

   For the mapping the settings for the fragment size have to be adapted. With a library of 3 kbp, a limit of 5 kbp should be set.

```
$ smalt map -i 5000 -j 1000 -m 50 -r 0 -f samsoft -o Mapped_3K.sam ref.fa rev.reads_3k_1.fastq rev.reads_3k_2.fastq
```

   The newly introduced -j parameter limits the minimal distance for mates. If you have more libraries, repeats this step.

6. Next, we will transform the SAM file into a binary version, called BAM. This will enable us to do more analysis, and save disc space, as long as you delete the SAM file after the transformation:

```
$ samtools view -b Mapped.sam -t ref.fa.fai | samtools sort - Mapped
$ samtools index Mapped.bam
$ rm Mapped.sam
```

   For mapping of the large insert, just adapt the commands by changing the file name Mapped to Mapped_3K.

7. If you would like to visualize the mapping you could use Artemis BAMview [21],

```
$ art -Dbam=Mapped.bam ref.fa
```

3.3 Local Assemblies

Although mapping is a powerful method, there are limitations: Some regions in the reference might be too polymorphic for reads to be mapped. Nor can larger insertions be detected. In this section we will first present steps showing you how to analyze those polymorphic regions by reassembling reads that map around it. Then we show how to assess larger insertions or new DNA elements through the assembly of non-mapped reads, the so-called *bin assembly*.

1. To reassemble a specific region we will need to gather the reads of this region (or at the border of it) and save them in the SAM format. We use samtools for this:

```
$ samtools view Mapped.bam Chr:From-To | sort>Region1.sam
```

   "Chr" is the name of the replicon and "From"-"To" the position of the target region. For this example use the Pf3D7_10_v3:1404400-1405500.

2. It is always good to have a look at the extracted reads to check if they come from the correct region, see column three and four:

```
$ head Region1.sam
```

3. Those reads can now be assembled.

```
$ velveth Assembly.55 55 -sam -short Region1.sam
$ velvetg Assembly.55 -exp_cov auto
```

   The first step generates the so-called de Bruijn graph. The next step is to generate the contigs from it. The parameters specify the input format (-sam), short reads (-short), and the expected median k-mer coverage (-exp_cov auto) here determined automatically.

4. Both programs generate a lot of output: # of reads, # k-mers, average coverage etc. To obtain statistics of the assembly, look at the last line: The number of nodes indicates the number of contigs, so the pieces obtained from the assembly. The *n50* is a continuity metric, *max* is the length of the largest contig, total the size of the assembly, and the last two numbers are the amount of reads used in the graph versus the total amount. Another way to look at the same statistics is the program stats:

```
$ stats Assembly.55/contigs.fa
```

5. As explained in Subheading 4.3, **step 2** the k-mer has the strongest impact on the assembly. It is good to iterate through different k-mer values in an automated fashion to optimize the assembly:

```
$ for ((kmer=31;$kmer<=73;kmer+=6)) ; do
velveth Assembly.$kmer $kmer -sam -short Region1.sam>out.velh.$kmer.txt;
velvetg Assembly.$kmer -exp_cov auto> out.velg.$kmer.txt
done
```

   This time each assembly output is written to a different file, through the ">" command.

6. To analyze the different assemblies, we "grep" the line that starts with "Final" in all the output file of velvetg and different k-mers:

```
$ grep "^Final" out.velg.*.txt
```

Which assembly is the best? For local assemblies you would expect one contig that represents the targeted region.

7. The result of the assembler is the fasta file Assembly.55/contigs.fa (or another k-mer depending on your genome). One way to analyze it would be to load it into Artemis or blast it against a public database. But in some cases the local assembly didn't return one contigs, but several. Our approach has two caveats: Some reads are too divergent to map, or an insertion occurred and we are not using the mate pairs. The following command will pull in the read's mate, even if it doesn't map:

   ```
   $ samtools view Mapped.bam | awk '($3=="Chr" && $4>=From && $4<=To) || ($7=="Chr" && $8>=From && $8<=To)' | sort>Region2.sam
   ```

   If you are following the example use Pf3D7_10_v3, 1394400, and 1400000 for the parameters "Chr," "From," and "To," respectively.

8. To assemble paired reads, just adapt the Velvet call as follow:

   ```
   $ velveth AssemblyRP.55 55 -sam -shortPaired Region2.sam
   ```

   ```
   $ velvetg AssemblyRP.55 -exp_cov auto -ins_length 400 -ins_length_sd 30 -min_pair_count 15
   ```

   The changes tell Velvet that the input file contains mate pairs and that their fragment size is 400 with a standard deviation of 30 % of the library. "-min_pair_count" is the number of read pairs needed to join two contigs into a scaffold. n.b. Here the fragment size is the median, rather than the maximal fragment size, as is the case with SMALT.

9. In case of large insert libraries, repeat **step 8** to gather those reads:

   ```
   $ samtools view Mapped_3K.bam | awk '($3=="Chr" && $4>=From && $4<=To) || ($7=="Chr" && $8>=From && $8<=To)' | sort>Region2_3K.sam
   ```

   The following parameters are added to the Velvet commands to include a second library:

   ```
   $ velveth AssemblyRP_3K.55 55 -sam -shortPaired Region2.sam -sam -shortPaired2 Region2_3K.sam
   ```

   ```
   $ velvetg AssemblyRP_3K.55 -exp_cov auto -ins_length 400 -ins_length_sd 30 -ins_length2 3000 -ins_length2_sd 30
   ```

   Again, have a look at the statistics. The number of bases in the assembly should have increased significantly.

10. The next step would be to iterate again through the k-mers as shown in **step 6**. In this part we showed you how to assemble a specific region of the genome. We would encourage the

reader to apply those commands to the example of the *MSP3* and S-antigen gene from the exercise, *see* Subheading 2.1.1. This procedure is appropriate if reads map to the reference but have many differences. Next we show how to get hold of the sequences that are completely different to the reference, like plasmids or very divergent multigene families.

11. The following command returns all the reads that don't map as proper pairs:

```
$ samtools view -F 2 Mapped.bam | head
```

But as discussed before, we would like to get just the mate pairs that don't map. The flag 4 is set if the read is not mapping and the flag 8 is set if the mate is not mapping. Adding the flags will return when both don't map:

```
$ samtools view -f 12 Mapped.bam | sort> NotMapped.sam
```

12. Now we can assemble the reads as before. You might want to run the last call also for the large insert library. Here is the assembler call for one library:

```
$ velveth Bin.55 55 -sam -shortPaired NotMapped.sam
$ velvetg Bin.55 -exp_cov auto -ins_length 400 -min_contig_lgth 300 -cov_cutoff 5
```

Two new parameters are introduced. First we want to ignore contigs smaller than 300 bp (min_contig_lgth). Next, regions (or nodes in the de Bruijn graph) that have a coverage of less than 5 k-mer are ignored from the assembly (cov_cutoff). This will minimize the possibility of false joins. If the read coverage is even you can set this option to (auto). The value will be set to half of the "exp_cov" parameter. We chose the name "bin" as many users tend to forget about the non-mapped reads.

13. Depending on the size of the organism whose genome you are assembling, the number of contigs might be significantly higher than in our little example. Now it is even more important to use different k-mers. The call would look like:

```
$ for ((kmer=31;$kmer<=73;kmer+=6)) ; do
velveth Bin.$kmer $kmer -sam -shortPaired NotMapped.sam>out.velh.$kmer.txt;
$ velvetg Bin.$kmer -exp_cov auto -ins_length 400 -min_contig_lgth 300 -cov_cutoff 5>out.velg.$kmer.txt
done
```

14. To look at the results you could again use the grep call, or use the little stats script:

```
$ stats Bin.*/contigs.fa
```

It is important to keep in mind, that those statistics do not tell you, how good (in terms of errors) the assembly really is. In the next section we are going to introduce a tool called REAPR that can evaluate the quality of the assembly, and return corrected assembly statistics.

15. Now include the mate pair library. Redo the **step 11** with the Mapped_3K file and run the assembly like in **step 9**. Use the stats command to see the impact of the library, especially the N_count.
16. At this step you might have generated larger sequences (several kbp) of the target region. In the example those will be subtelomeric regions with the genes of different gene families. To look at it, you could for example load it into Artemis (file Bin.55/contigs.fa) and detect open reading frames (ORF), *see* Subheading 4. Alternatively you run ab initio gene finding tools like Glimmer [22] or Augustus [23], *see* Subheading 3.5.

3.4 Whole Genome Assembly

Although the bin and local assemblies are powerful ways to get results quickly, in many cases a complete de novo assembly is necessary. Reasons are that no reference is available or is too divergent, or the mapping and SNP calls aren't accurate enough. Also, it is more difficult to combine a local assembly with the reference into a contiguous sequence than to do a de novo assembly. We assume that the reader has understood the earlier steps, as this section builds on them.

In the sections before, we performed de novo assemblies on a limited read set. Now we are going to use all the reads. The assembly call won't be very different, but we are going to do improvement of the assembly. Let's again assume that your short insert reads are called reads_1.fastq and reads_2.fastq and the reads of the large insert library are reverse complemented and called rev.reads_3k_1.fastq rev.reads_3k_2.fastq, *see* Subheading 3.2, **step 5**.

1. To run the assembly is straight forward:

```
$ for ((kmer=31;$kmer<=73;kmer+=6)) ; do
velveth deNovo.$kmer $kmer -fastq -shortPaired
-separate reads_1.fastq reads_2.fastq -fastq
-shortPaired2 -separate rev.reads_3k_1.fastq
rev.reads_3k_2.fastq>out.velh.$kmer.txt;
velvetg deNovo.$kmer -exp_cov auto -cov_cut-
off 5 -ins_length 400 -min_contig_lgth 300
-ins_length2 3000 -ins_length2_sd 30>out.
velg $kmer.txt;
done
```

2. As before you can now look into the assembly with the stats script, or grep the line in the output files, *see* Subheading 3.3, **step 4**.
3. You might not necessarily want to optimize the assembly based on the n50. One example would be to increase the number of

genes of a specific gene family, which is very repetitive. So instead of looking at a large n50 you want to increase the numbers of genes. In general, a higher k-mer will better separate the different copies. Also the modification of the "-max_divergence" parameter might help.

4. The next step is to check the quality of the assembly. Just because the assembly has good statistics, doesn't mean it is a good one with no error.

   REAPR [3] is a tool that can find errors in assembled sequences by remapping the reads. First, the reads, ideally from a large insert library, have to be mapped against the assembly. This can be done with the commands of Subheading 3.2, or with this little program from the tarball:

   ```
   $ map.smalt.sh deNovo.55/contigs.fa rev.reads_3k_1.fastq rev.reads_3k_2.fastq MappedNovo55 5000
   ```

   This will generate the BAM "MappedNovo55.bam" of the mapped mate pairs on the chosen assembly. As parameter the script uses "-x," "-r 0," and "-y 0.8."

5. Now we can run REAPR:

   ```
   $ reapr pipeline deNovo.55/contigs.fa MappedNovo55.bam Reapr.55
   ```

   Different metrics are going to be applied to decide which bases are correct and which are wrong; scaffolds will be broken where there are errors. The important outputs are in the report file with the new statistics of the assembly (05.summary.report.txt) and the new assembly file 04.break.broken_assembly.fa.

6. In choosing the best assembly it is better to compare the corrected n50s rather than those given by the assembler. For each assembly the mapping and REAPR would need to be run.

7. Once we choose the best assembly, we are going to do another round of scaffolding using the program SSPACE. Though assemblers themselves have a scaffolding step, other scaffolding might improve the assembly. First we are going to iterate through different settings of the short library, and then the mate pair library. To prepare the call type:

   ```
   $ echo "LIB1 reads_1.fastq reads_2.fastq 400 0.3 FR">lib1
   ```

8. The resulting file will provide SSPACE with the fragment size (400 bp), the standard deviation (30 %), and the read orientation FR (Forward/Reverse). If your library has a different fragment size, adapt the command in **step 7**. Now run SSPACE:

   ```
   $ SSPACE_Basic_v2.0.pl -l lib1 -s Reapr.55/04.break.broken_assembly.fa -k 200 -n 31 -b out.200
   ```

The parameters indicate the nature of the reads (-l lib1), the input file, the result from REAPR (-s), the number of mates needed to join two contigs/supercontigs (-t) to a supercontig, how many bases must overlap to merge two contigs rather than scaffolding them and -b, the output. The file out.200.summaryfile.txt gives a summary of the mapping and scaffolding and out.200.final.scaffolds.fasta holds the current assembly.

9. In the step before, we used 200 mates to join two contigs. This might sound a lot, but we are looking at fragment coverage, rather than read coverage. Also, the way SSPACE works, the best results are obtained by first making the most high scoring joins and then running SSPACE again with a decreasing k parameter.

```
$ SSPACE_Basic_v2.0.pl -l lib1 -s out.200.final.scaffolds.fasta -k 100 -n 31 -b out.100
$ SSPACE_Basic_v2.0.pl -l lib1 -s out.100.final.scaffolds.fasta -k 50 -n 31 -b out.50
$ SSPACE_Basic_v2.0.pl -l lib1 -s out.50.final.scaffolds.fasta -k 10 -n 31 -b out.10
```

Looking at the statistics of the scaffolding results you should see a clear decrease in the number of contigs/scaffolds.

```
$ stats out*.final.scaffolds.fasta
```

We would encourage the reader to try to scaffold the output directly with 10 read pairs for the -k parameter to compare the effect on their assembly.

10. Now we are going to scaffold with the mate pair library. One important point must be made. Small contigs of less than 500 bp, which belong between two larger contigs, might not be included in the scaffold, as the number of large-insert reads between the large contigs is higher than between them and the smaller contig. To our knowledge no scaffolder solves this problem in a satisfying manner. Therefore we normally exclude contigs smaller than 500 bp. The hope is that in the later stages we can regenerate the sequence by doing gapclosing, **step 18**. Leaving the contigs in would make this more difficult. The size limitation can be added in the velvetg step or with the following PERL script:

```
$ RemoveSequencesSmaller.pl out.10.final.scaffolds.fasta 500>SSPACE.1.fasta
```

11. Here the command to prepare and run SSPACE on the mate pair library:

```
$ echo "LIB2 reads_3k_1.fastq reads_3k_2.fastq 3000 0.3 RF">lib2
```

Note, you don't have to use the reverse complemented reads; you can set the direction to RF rather than FR.

```
$ SSPACE_Basic_v2.0.pl -l lib2 -s
SSPACE.1.fasta -k 500 -n 31 -b out2.500
```

Next rerun the command, decrease k, as shown before for the short insert library.

12. Compare the number of scaffolded contigs between the use of short and large insert size libraries. Generally, large insert libraries have a strong impact on the contiguity of the sequence. They enable bridging of repetitive regions. This is very valuable for parasites which may have repetitive subtelomeric sequences and to improve comparison between different isolates for structural variation.
13. Remember that the assembler already did some scaffolding. It might be advantageous to tell velveth not to use the large insert size library for scaffolding. Scaffolder can use the complete length of the reads to place reads (rather than just the k-mer) and they can deal with PCR duplicates. The call would look as follows for a k-mer of 55:

    ```
    $ velveth deNovoSE.55 55 -fastq -shortPaired
    -separate reads_1.fastq reads_2.fastq -fastq
    -short -separate reads_3k_1.fastq rev.
    reads_3k_2.fastq
    ```

    ```
    $ velvetg deNovoSE.55 -exp_cov auto -cov_cut-
    off 5 -ins_length 400 -min_contig_lgth 300
    ```

14. In some projects a mix of different sequencing technologies are used. The scaffolding step might be the best step at which to combine the different technologies. Assuming you have a BAM file of the mapped reads, do:

    ```
    $ samtools view -F12 Mapped454.bam | awk
    '$7!="="' | sort | BAM2consensus_reads.pl
    Assembly.fa Reads_Scaff
    ```

15. We are again using awk, sort, PERL, and pipes. As parameter for the PERL program BAM2consensus.pl you have to provide the assembly sequence (Assembly.fa) and the result prefix for the new read files (Reads_Scaff).
16. As mentioned before most of the errors are introduced to the assembly in the scaffolding step. Therefore we recommend that you rerun REAPR. Caution must be taken for the fact that SSPACE renames the scaffolds, including a pipe symbol. The following function of REAPR can be used to rename them:

    ```
    $ reapr facheck out2.10.final.scaffolds.fasta
    ForReapr.fa
    ```

    Now we have long scaffolds with sequencing gaps and some base errors. In the next steps we are going to try to close sequencing gaps with IMAGE ([11]) and correct base errors using ICORN ([13]). If you have a closely related reference

you can order your scaffolds against it and transfer the annotation, using the tools ABACAS and RATT. These programs are part of the PAGIT pipeline [24]. PAGIT has an automated way to invoke the tools; however here we are presenting the specific program calls. For more in-depth information we recommend to read the PAGIT protocol paper.

17. If you don't have a reference sequence available, do

```
$ PAGIT.noRef.sh ResultReapr.fa read_1.fastq
reads_2.fastq 500 Final.fa
```

To call the script successfully give it your current assembly, the reads, the insert size, and the final result name.

18. The PAGIT script will first run nine iterations of gapclosing and contig extension, decreasing the k-mer length every three iterations from 71, to 55 and then 41. The calls look like:

```
$ image.pl -scaffolds ResultReapr.fa -prefix
reads -iteration 1 -all_iteration 3 -dir_
prefix ite -kmer 71 -smalt_minScore 60
$ restartIMAGE.pl ite3 55 3 partitioned
$ restartIMAGE.pl ite6 41 3 partitioned
```

Local assemblies are done for each sequencing gap and at the ends of contigs, by including the mate pairs that don't map, as done in Subheading 3.3, **step 7**. The k-mer for the assembly can be changed as can the minimal score for a read to be placed with SMALT (smalt_minScore - the "-s" parameter in Subheading 3.2, **step 4**). Again, we encourage the reader to change the parameters and analyze the impact. In the end, the contigs are joined and placed in the file Res.image.fasta by

```
$ contigs2scaffolds.pl ite9/new.fa
ite9/new.read.placed 300 10 Res.image
```

19. After the IMAGE step, 50–80 % of the sequencing gaps should be closed and many scaffold ends extended. Next, we are going to apply the tool ICORN to correct base errors. Compared to REAPR it looks for 1–3 bp errors and corrects them. REAPR scores single bases rather than correcting them. Reads are mapped with SMALT and differences between reads and the reference are found and corrected.

```
$ icorn2.start.sh Reads 500 Res.image.fa 1 3
```

As before, the reads, fragment size, and the input reference from IMAGE are passed to the script. The last two parameters are the iteration start and stop. The output will be a summary file (Res.image.fasta.summary.txt) and the corrected sequence file (Res.image.fasta.4). If run through the PAGIT pipeline the final result will be called Final.fa. The next step for the analysis would be to start with the ab initio annotation of genes, *see* Subheading 3.5.

20. In cases where a reference sequence is available, the PAGIT pipeline has a tool to order the contigs against the reference, and to transfer the annotation. This following call will also invoke the gap closing (IMAGE) and correction (iCORN) steps:

```
$ PAGIT.sh ResultReapr.fa read_1.fastq reads_
2.fastq 500 Final ref.fa AnnotationDIR
```

The call is very similar to the above one, with the exception that the reference and a directory with the annotation of the reference are given.

21. In the first step, the scaffolds will be ordered against a reference. A certain caution must be taken however. If it is known that the species under study has many synteny breaks relative to the reference, the resulting order might not be correct. Furthermore, we recommend deleting regions (or replace them with n's—the symbol for an ambiguous base) where there is evolutionary pressure, as in virulence factors or the subtelomeres of species such as *Plasmodium* or *Trypanosomes*. To put it another way, you would like the scaffolds to be ordered only against the well-conserved parts of the reference.

22. As discussed, the PAGIT pipeline will order the contigs with the following command:

```
$ abacas.pl -r ref.fa -q ResultReapr.fa -p
nucmer -d
```

If you work with more divergent species, you might want to change *nucmer* to *promer* in the PAGIT.sh script. Instead of using nucleotide similarity, an amino acid comparison will be done. The result will be a multifasta file. Scaffolds ordered against a reference chromosome will be joined with n's and are now named after the reference replicon.

23. After this step, IMAGE and ICORN will be run again (**steps 16** and **17**).

24. Now it is possible to transfer the annotation onto the improved assembly with RATT. The call used by the PAGIT script is:

```
$ start.ratt.sh AnnotationDIR ForRatt.fa
Transfer Species
```

Similar to ABACAS, the last parameter determines the similarity. Due to the nature of the program it won't work with amino acid comparisons. The parameter *Species* is the most robust. If your reference is very similar, and the assembly is contiguous, in pieces larger than 10 kb, we would recommend the value "Assembly" or "Strain" for this parameter. The value "Transfer" is a prefix for the result files. "AnnotationDIR" is the position of the reference annotation in embl format. Note that you might need to adapt the configuration file of RATT, with a simple editor. Here is the position of the file:

```
$ echo $RATT_CONFIG
```

This configuration file enables you to set the start codons, splice sites and if pseudo genes should also be corrected.

25. The result of RATT is one annotation file for each replicon, starting with Transfer.*.final.embl (ordered and unordered scaffolds). You can open them in Artemis. To compare these with the reference, use ACT. ACT can be seen as two Artemis view joined by a similarity comparison, *see* Fig 1. First we will generate this comparison file for a single chromosome by extracting the chromosome sequence from the multifasta file, preparing it and blasting it.

```
$ samtools faidx ref.fa RefChr>RefChr.fa
$ formatdb -p F -i RefChr.fa
$ blastall -p blastn -m 8 -e 1e-6 -d RefChr.fa
-i Sequences/deNovoSuper -o comp.RefChr.blast
```

where RefChr.fa is the name of the reference chromosome. Without going into too much detail, the reference chromosome is being compared to scaffolds or ordered scaffolds, from the PAGIT pipeline. These sequences are in the folder "Sequences." To start ACT, use the reference file, which should be in the folder "embl," the comparison file you just generated. The result file from RATT is Transfer.deNovoSuper.final.embl.

```
$ act AnnotationDIR/RefChr.embl comp.RefChr.
blast Transfer.deNovoSuper.final.embl
```

26. In ACT, it is possible to see insertions, deletions, and rearrangements in the comparison window. To evaluate the RATT transfer, load onto the reference chromosome the file Transfer.RefChr.NOTTransfered.embl (click on File->2nd option ->Read an Entry). This file will show the gene models that weren't transferred. Lastly, load the GFF file from the "Query" folder, and look for the synteny tag. These regions have no synteny to the reference, and are probably insertions. Furthermore, those will need to be annotated separately. Figure 1 is an example of an ACT view.

27. Using ACT, it would be possible to look for Open Reading Frames (ORFs), not overlapping RATT-transferred annotation and follow the description of Subheading 4.3, **step 10**.

28. Although at this stage we have an improved, annotated genome, one quality check remains to be done. We recommend doing a "bin" assembly as described in Subheading 3.3. During the process, we deleted small contigs, which might contain important sequences. Also, some assemblers exclude reads with an extreme k-mer coverage, for example those derived from mitochondrial or plasmid DNA. Furthermore,

maybe something went wrong in the process—there could have been a truncated file. If the bin assembly contains interesting sequences, join it with the current, improved assembly:

```
$ cat ForRatt.fasta bin.55/contigs.fa>
Joined.Assembly.fasta
```

29. Each program generates temporary files, which use a lot of space. For example, IMAGE generates ite*/ directories, SSPACE creates several mapping directory or ICORN does ICORN2_*/ directories. Those should be deleted on a regular base with the rm command, i.e:

```
$ rm -rf ICORN2_*/ ite*/ reads/ bowtieouput/
```

30. Some of you may have noticed that it would be possible to first run IMAGE to extend contigs, then run the scaffolder. Or, perhaps it would be good to scaffold the bin contigs into the improved assembly. Both these points are true and we hope to have given the reader the impression that each process can be seen as a module. The order can be changed, and processes iterated. The aim of this protocol is to show the reader the possibilities of generating assemblies and improving them.

3.5 Annotation

In the previous section we showed how to improve the quality of a genome sequence and how to transfer annotation from a reference onto the assembly. But the genome is far from being well annotated: Where its sequence is not similar to the reference, like in multigene families or insertions, the annotation would be transferred poorly or not at all. To annotate those regions, an ab initio gene prediction must be done. (The same would need to be done, when no closely related reference exists.) Here we present a method of predicting gene models and a first pass functional gene annotation. In the end, we show how to merge the new annotation with the RATT transferred annotation.

Although the method presented here for gene prediction and functional annotation is valid when a closely related reference exists, we highly encourage the reader to further study the subject of gene prediction and functional annotation, as each program and method has its strength and weakness [5, 6].

1. First, the gene predictor has to be trained. For simplicity we assume that we can use the genes from the reference genome.
2. Load the annotation from the reference genome (you can use the one of the test dataset) into Artemis and save it in the GFF format (File ->Save An Entry As ->GFF Format). For simplicity, save it as ChrX.gff. Accept all the warnings that some classifiers won't be saved.
3. Next we require to know the names of each chromosome in the reference fasta file ref.fa. The command

```
$ grep '>' ref.fa
```

will do the job. Remember the name of the chromosome of which you generated the GFF file. In this example her for simplicity we assume it is called ChrX.

4. This command will transform the gff into the gb format needed for Augustus.

```
$ gff2gb.sh ChrX.gff ChrX ChrX.gb
```

In case that you have more than one chromosome, redo the steps from number 2 and concatenate the files with:

```
$ cat ChrX.gb ChrX2.gb … ChrXn>All.gb
```

5. Next, we initiate Augustus

```
$ new_species.pl -- pecies=NEW
$ etraining -- Species=NEW --
stopCodonExcludedFromCDS=false ChrX.gb
```

The first command will generate a training instance for your reference, called "NEW." This instance is than trained in with the gene models saved in Artemis second command. Read carefully the output of the programs. Gene models that seem to be wrong for Augustus will be excluded.

6. Now we can apply the trained model to the new assembly:

```
$ augustus -- Species=NEW ImprovedAssembly.
fasta>abintio.gtf
```

The output is a gtf file that contains all the predicted models.

7. Those models obviously don't have any functional annotation. The next command will attribute to each model the first BLAST hit with an E-value of at least 1e-40 of the new model against the proteome of the reference genome "ref.aa.fa" in the next command. The output will be EMBL files in the "Augustus" directory.

```
$ augustusAnnotate.sh ImprovedAssembly.fasta
abintio.gtf ref.aa.fa
```

At this point we have the annotation of the ab initio gene models. Now we present here how to merge them with the gene models of the RATT transfer.

8. The first step is to delete gene models that contain still errors in the RATT transfer. Although RATT tries to correct gene models, this step can fail. The next command will use the statistic of each gene stored in the "report" files:

```
$ cat Transfer.*txt | perl -nle '@ar=split(/\t/);
if    (($ar[8]+$ar[9]+$ar[10]+$ar[11]+$ar[12]+
$ar[13])>0){print $ar[0]}'>exclude.txt
```

This command counts the columns 8–13 that represent specific errors, such as wrong start codon, frame shifts, or incorrect splice sites.

9. The next call will exclude all the models that are flagged to be wrong from the EMBL files. (The new files will be stored in the directory RATT_excluded):

```
$ mkdir RATT_excluded
$ for x in ` ls Transfer.*final.embl ` ; do
cat $x | excludeGeneEMBL.pl exlude.txt> RATT_excluded/$x;
done
```

10. At this stage it is possible to join the models from the RATT model with the models from the gene prediction. The rule is that if a predicted gene overlaps with a RATT transferred model, it will be deleted.

    First we generate a directory to store the files:

```
$ mkdir Joined
$ for x in `grep '>' assembly.fa | sed 's/>//g'`; do annotation.MergeAnnotation SecondAway.pl
RATT_excluded/Transfer.$x.final.embl
Augustus/$x.embl>Joined/$x.embl;
done
```

11. Now it is possible to examine the annotation in Artemis.

```
$art Joined/ChrX.embl
```

    Further you can also add the models from the ab initio gene prediction as a separate track (Menu: File -> Read Entry, and select the gff from the "Augustus" directory. If genes are wrong, it is advisable to delete those from the RATT transfer and redo **step 10**.

12. The last step would be to give the genes systematic ids. This can be done in Artemis or with following script:

```
$ mkdir final
$ cat Joined/chrX.embl | annotation.giveIDCDS.pl NameID_01 T>final/chrX.embl
```

    This command has to be run for each new chromosome/supercontigs, in this case "chrX.embl." The first parameter (NameID_01) would be the ID plus the chromosome number, here 01. The second parameter is optional: RATT transfers also the locus tag from the reference. If this has the prefix of the second parameter (in this case "T") the reference locus_tag will be stored in the ratt_orthologs tag. If not, it gets deleted.

4 Notes

This protocol uses a wide range of tools and commands. We assume that a novice user will probably need a week to work through the protocol when assembling a bacterial genome. It is important to have a computer with all the tools installed and enough memory, around 6 GB. When using the preinstalled tarball (Subheading 2.1, **step 1**) be sure to run it on a computer with at least 6 GB of memory. To run through the provided example will take around 1 day.

4.1 Preprocessing

It is important to keep in mind that reads can be of poor quality and that sequencing might generate insufficient depth. Although assembly will still be possible, the representation of the genome as a whole will be poor, and the range of meaningful downstream analyses will be quite small.

Another important point is the possibility of contamination. Depending on the source of material, host contamination is common. If host contamination is present, and there is a reference sequence for the host, map the reads against the host genome (Subheading 3.2, **step 3** ff) and extract those from the results file with the command in Subheading 3.2, **step 5**. This will filter most of the contamination. For the rest, once the assembly is done, blast the contigs against the reference. You can also compare the GC content of the contigs. The contaminants normally have a different GC content from that of the target genome.

1. There are many programs to trim adapters. The key thing is to have a file of adapter sequences. In our experience, it is most often mate pair libraries and bad quality runs that have many adapters.
2. Trimming should be done for very bad quality reads. But the best cutoff is tricky to set. A base with a quality value of ten still has a 90 % chance of being correct. For the assembly, one out of ten reads will be excluded and this generates noise. With enough coverage, and without doing read correction, choose a quality cut off of 20.
3. The SGA correction will first index the reads and then do the correction by looking for k-mers in reads. A read is corrected where it contains a k-mer with too low abundance. Depending on the complexity of the genome and the read coverage, this step takes between 4 h and 4 days. The biggest impact of the correction will be on the memory and runtime requirement for the assembly.

4.2 Mapping Reads

There are many tools that can be used to map reads against a reference [25]. SMALT is a tool that gives us more control when a read is mapped, using the parameters for minimum score or fragment size.

1. The indexing step can be optimized in terms of speed and sensitivity. The k-mer is the length of the identical words and the parameter *s* is the step size. With a step size of one every k-mer is taken, with a step size of two only every second k-mer is used. Low k and *s* parameters will result in many small pieces that represent the genome. More divergent reads can be mapped, as if you have an SNP every 14 bases a k-mer of 13 can be found in the reads, but not a higher k-mer. The price will be the runtime. Higher parameters like $k=20$ and $s=11$ will result in a more sparse representation. Reads with many differences to the reference might not get mapped. But the mapping time is significantly shorter. A k-mer smaller than 13 should not be used, due to the runtime and the fact that k-mers which occur very often will be ignored.
2. For the mapping step many parameters can be set ($ smalt map -H). Interesting parameters include "-y" to limit the placement of the reads by identity, "-n" to use more than one thread, or "-d" to allow multiple hits for each read. In terms of runtime, for a 5 MB genome with 100× read coverage, we estimate a mapping time of 1 h. The runtime increases linearly with the number of reads or the genome size. Normally not more than 2 GB of memory is needed.
3. To map large insert libraries, reverse complement the Illumina reads. For 454 reads, you just need to reverse complement the second read. Although the mappers have functions to reverse complement, some just recalculate the flag, rather than really reverse complementing the reads. This is generally okay, but as our analysis builds on the mapped reads, we have to have them in the correct orientation.
4. The reason for sorting and indexing the reads is a faster access to reads at specific positions. Later this will become more obvious. This step takes around 10 % of the total mapping time. To get a statistic of the mapping, do $ samtools flagstat Mapped.bam.

4.3 Local Assemblies

1. The samtools command "view" prints all the reads mapped to the specified chromosome "Chr" between the positions "From" to "To." Select a region slightly larger than the one you are interested in. For example if you chose a specific gene, extend the border by 100 bp.
2. Those two commands will do the assembly. The first command builds the graphical representation of the reads. As for the mapping, we select a specific k-mer, which will represent the nodes of the graphs. Two reads are basically joined into a contig if they share an identical k-mer. The k-mer setting will have the strongest impact on the quality of the assembly. If the

k-mer is too long, two reads that should be joined might not get joined due to sequencing errors. On the other hand, shorter k-mers are more likely to be repetitive, which is a big issue for the assembler. This could lead to many small contigs, or in some rare cases also to mis-assemblies. The second command cleans the graph and finds an Euler path through the graph. A very important parameter is the expected coverage. This can be determined automatically by Velvet. The value can be obtained manually through the "stats.txt" file, see velvet manual. Obviously, there are many assemblers that could be used at this point. Although the results will vary slightly, we think that it will be more important here to explain the general procedure than to list all the different assembler calls. In the end, they have very similar parameters and ways to be called. Both Velvet programs have many parameters. The most important will be explained below. To see them all, just type ($ velveth or velvetg).

3. The overview values of the Velvet and the stats program have the following meaning. The sum is the total number of bases in the assembly. "n" represents the number of contigs in the assembly; "mean" and "largest" relate to the contig size. The N50 is the length L such that 50 % of the assembly lies in contigs of at least length *L*. N60 is defined analogously, but for 60 % of the genome, etc. The N_count is the amount of n's contained in the assembly.
4. The for loop is a feature of bash, the Linux environment you are working in. It basically iterates over several values of the variable $kmer from 31 to 73, with a step size of 6. This command will take roughly eight times as long as a single Velvet call. For larger read sets you might want to run different Velvet calls on different computers, but this kind of optimization is not part of this chapter.

   Depending on the quality of the reads, the amount of coverage and the base composition of the genome, a certain k-mer will generate a more contiguous and larger assembly. If you have coverage over 80×, it is likely to be a larger k-mer. It is possible to iterate the k-mer with a lower step size, around the optimum. Please note that a k-mer must always be an odd integer value.
5. This command will take some minutes to run. All reads are handed (or piped) to the awk command. This one looks to see whether a read or its mate map on the specified chromosome and position (columns 3, 4 and 7, 8). These two conditions are connected with a logical OR (||), so if at least the mate or the read fulfill the condition, the mapping information of the read is piped to a sort command. The sorting is necessary, as the assembly step will require a SAM file, sorted by read name, so all reads in a pair are together. To write the output into a file,

use the ">" command. As we ensure to collect both mates, a local assembly should be able to reconstruct an insertion of nearly twice the fragment size. As a look ahead, this method to pull in non-mapping reads through mate pairs is the basic idea of gap closing software like IMAGE, part of PAGIT that will be discussed later.

6. Velvet will finally try to scaffold the contigs using the mate pairs. Basically the reads are mapped back to the graph (using the k-mer as seeds), and if a certain number (value of -min_pair_count) maps to two contigs, these will be joined into a scaffold. Ns are inserted between the contigs, the number of which depends on the expected gap size. This step is the most likely source of mis-assemblies. Therefore higher values will tend to generate more conservative assemblies, with fewer errors and more contigs. A good setting is generally to set the value to the median coverage, which is returned in the second last line of the velvetg output. To set this value, the velvetg call would therefore need to be run twice. Finally, although the output of Velvet now has supercontigs, the file will still be called "contigs.fa" The quickest way to check if the results are contigs or scaffolds is to use the stats program—if the N_count is not zero then the results are scaffolds.
7. Although including the large insert size library reduced the number of scaffolds, the effect might not be seen in local assemblies.
8. We chose to select only read pairs where neither map, to avoid chimeras entering into the read set. Although these could be filtered out later (parameter -cov_cutoff), they would slow down the process and introduce noise. To obtain these reads we query their flags (the second column of the BAM file), in this case 12, read and mate don't map. Next, each line of the SAM file is passed via the pipe command to the sort program.
9. The mate pair library should make a huge difference to the statistics of the supercontigs. Later we are going to discuss further problems with scaffolding using large insert libraries. But depending on the organism, the final result should be one scaffold per amplicon, which is rarely achieved.
10. To detect and annotate ORFs do the following in Artemis: click on Menu Create ->mark open reading frame. Choose a minimum length of 200 and enable the option to break at contig boundaries. Next, you can blast the obtained ORFs against a uniprot database: Select ->Select all CDS; Run ->Run blastp on selected feature ->Uniprot_eukaryota. Choose Uniprot_bacteria, if you work with bacteria. The blastp gets run. Once done you can see the results by selecting a CDS and pressing the keys crtl+back quote.

4.4 De Novo Assembly

1. As all the reads are going to be used, the runtime will be increased. Genomes below 5 MB will run in around 20 min, and need just 2 GB of memory. Larger genomes, up to 30 MB, will take around 1 h and the memory can increase up to 30 GB. To lower the memory requirements, work with a higher k-mer (>49) and correct the reads before running the assembler (Subheading 3.1, **step 3**). For very large parasites, Velvet might need more than 200 GB memory. In this case, the SGA assembler is a useful alternative, which normally doesn't use more than 60 GB of memory. When using large insert size libraries, a fraction of reads can point in the wrong direction, and may result in incorrect scaffolding. To avoid this, set the shortMatePaired parameter to yes.

   Some users might want to try a tool called velvetoptimser.pl. It automatically tries different settings in velvet to optimize a specific value, such as a large N50 or assemblies with many bases. Interestingly, in our experience, manually iterating through the parameters generates still better results for larger genomes.

2. To test the different assemblies for the best representation, one could blast conserved motifs against the different assemblies and assume that the one with the highest number might best represent the gene family. Obviously, one must find a balance with the other statistics, such as the N50.

3. It is a very common procedure to join several steps of processing into one script. This reduces not only the amount of time, but also the likelihood of typing errors.

4. REAPR generates many statistics. A very useful feature is the generation of the per-base quality. Every base will be scored for correctness, try $ reapr perfectmap. REAPR with only break scaffolds if the error is over a gap (n's). If an error is within a contig, the bases are replaced with Ns and the deleted sequence is written into the "bin" file (Contig errors can also be broken with reapr break -a). Plots for the different errors can be loaded into Artemis. The command $ reapr plot<chromosomeName> will generate all the plots and a script to start Artemis automatically. For genomes under 4 MB REAPR takes less than 10 min. For larger genome, the runtime and memory requirement is similar to the mapping step.

5. The application of automatically mapping reads and correcting assemblies needs a bit of scripting. Supplied in the tarball is a script called "deNovoPlus.sh." It is a very trivial script, which has as parameters the k-mer and the insert size. The following commands will be run:

```
$kmer=$1
$insersize=$2
```

```
velveth deNovo.$kmer $kmer -fastq -shortPaired
-separate reads_1.fastq reads_2.fastq velvetg
deNovo.$kmer -exp_cov auto -cov_cutoff 5 -ins_
length $insersize -min_contig_lgth 300

map.smalt.sh  deNovo.$kmer/contigs.fa  rev.
reads_3k_1.fastq rev.reads_3k_2.fastq Mapped
Novo$kmer 5000

reapr pipeline deNovo.$kmer/contigs.fa Mapped
Novo$kmer Reapr.$kmer
```

The script basically joins all the commands explained before, and assumes a very stringent naming convention. This is a simple example how powerful and easy scripts can be. One would call it through a for loop and use different k-mers, or start the job on different computers to save time.

6. Compared to the assemblers, a scaffolder will use the complete length of the reads to determine its position in the assembly. Also it should ignore duplicate reads, where mate pairs that came from the same DNA fragment are overrepresented due to the PCR amplification step. We use SSPACE as it is straightforward to use and was one of the first using Illumina data. As mentioned later in Subheading 3.4, **step 12**, the scaffolding of SSPACE is better than that of velvet. Therefore one could disable the scaffolding with the large insert size library.
7. At first it might sound weird to iterate through different settings, decreasing the evidence. But again, this optimizes the results, without generating more errors.
8. As for most of the tools, there are limitations. To further scaffold you would need combinatorial PCR. Alternatively you can order contigs of at least 40 kb using optical maps.
9. This step will have more impact on larger genomes, where more k-mers are repetitive, more duplicate reads are expected, and the order of contigs is not so obvious. There are also limitations of the existing scaffolders. If three copies of a repeat are joined into one contig, the scaffolder should not join the contigs. It would be better to exclude those reads from the assembly, and hope that local assemblies (*see* Subheading 3.4, **step 17**) will regenerate them, or split the collapsed repeat into three copies. This is not a problem if the repeat is smaller than the largest mate pair library.
10. SSPACE is designed to work with Illumina reads. But you may have 454 8 kb/20 kb libraries available for scaffolding. Here we present a script that returns fake reads from a BAM file, with the consensus sequence taken from the reference using the correct read length. This preprocessing step will also reduce the reads that have to be analyzed by SSPACE, by excluding reads not holding scaffolding information (as mapping for

example onto the same contig). First you have to reverse complement your reads so that they point towards each other (*see* Subheading 3.2, **step 5**). Next map them with SMALT (Subheading 3.4, **step 4**). Then you can do **step 14**. The reads can now be given to SSPACE. If you have a large genome, you can also do this to speed up the runtime of SSPACE significantly.

11. There is always a trade-off between automated pipelines and running the tools one by one. Here we provide an automated approach, while also explaining the main parameters for each tool.
12. IMAGE is a powerful tool to improve your assemblies. The price is a long runtime. In each iteration, the reads are mapped, gathered for each gap or scaffold end, and a local assemblies are done. Subsequent iterations are faster, as properly paired reads will not be remapped and regions that could not be improved in the previous iteration will not be touched. But this method is still much faster than filling gaps by PCR. Interestingly, other gapfilling tools close different types of gaps than IMAGE. For the remaining sequencing gaps, you would need to do PCR, if considered necessary.
13. ICORN is an iterative tool that takes 30 min per iteration for genomes around 4 Mb and 8–24 h for genomes around 200 GB.
14. Some scaffolds will not get ordered. These may be the most interesting sequences, as they will be the most different from the reference (if they are not contamination). It is important not to forget them!

    After running ABACAS, you should rename your ordered scaffolds. Naming is always important and mostly needs to be done in a manual matter, through an editor. For those who would like a more automated way, look into the Linux "sed" program.
15. As mentioned RATT can only transfer annotation where synteny exists. But those regions without synteny are the ones to be examined in more detail. Furthermore, it will be necessary to annotate them separately, Subheading 3.5.
16. It might be surprising, but indeed errors occur that no one expected. To do a "bin" assembly is an efficient way to double-check the assembly for errors.

4.5 Gene Prediction

Although stated before, we must iterate that the ab initio gene prediction and functional annotation we present is not the most sophisticated, but valid as a first pass annotation if a closely related reference genome exists. The reference genome will be used to train the gene finder and as a database for the functional annotation. General errors of the gene prediction are overprediction,

missing exons, and wrong splice sites. In the functional annotation, it can be wrong to assume homology due to similarity to homology, especially when paralogous exists. Nevertheless, as a starting point (and merged with the RATT transfer) the method presented here will be extremely helpful.

For bacterial gene prediction we would recommend Glimmer or Prokka [7], which are easy to use.

1. In case that you don't have a closely related reference, you will need to generate 200–400 high-quality gene models. This training set should cover as many different type of genes, not just the core genes, like predicted from CEGMA [26].
2. A lot of time in bioinformatics is spent on transforming files to different formats. To ensure that this won't be a problem for this protocol, we generated the gff file through Artemis. Users that are more experienced with scripting will have their own methods, especially, when the genome has many chromosomes.
3. This script hides some ugly code. From the gff a gtf is generated, with the name of the sequence. This is then transformed to a genbank file, using an augustus script. For more advanced users, it might worth to look into the script and modify the parameters to obtain better results.
4. If the second command returns a lot of errors, like wrong models, then in the transformation step something went wrong. One solution might be to name the gene models with locus_tag in Artemis, without using any special characters in the name.
5. All the above steps run within minutes. This step might need several hours if the genome is over 30 Mb.
6. To generate a full EMBL file the program "`ratt.main.pl doEMBL`" can be used. It combines the sequence (fasta file) with the annotation (EMBL format).
7. In some case, it is desirable to exclude also specific gene families, as it is likely that the transfer will be wrong, for example missing exon, and the *ab initio* gene prediction might be better. `$ cat Transfer.*txt | grep "variant erythrocyte" | cut -f 1 >>exclude.txt` will add all genes ids that have the annotation "variant erythrocyte" as product to the list to exclude genes.
8. Depending on the expected quality of the annotation, here a lot of time can be spent. Further, several information from the reference transferred onto the new assembly might not be relevant. This information should be deleted.
9. Actually, the geneID should be obtained from a database like EBI, to agree with their submission format.

Acknowledgements

I would like to thank Adam Reid, Martin Hunt, and Bernardo Foth for proofreading the chapter.

References

1. Huang X, Madan A (1999) CAP3: a DNA sequence assembly program. Genome Res 9(9):868–877
2. Myers EW et al (2000) A whole-genome assembly of Drosophila. Science 287:2196–2204
3. Simpson JT et al (2009) ABySS: a parallel assembler for short read sequence data. Genome Res 19(6):1117–1123
4. Zerbino DR, Birney E (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Res 18:821–829
5. Compeau PE, Pevzner PA, Tesler G (2011) How to apply de Bruijn graphs to genome assembly. Nat Biotechnol 29(11):987–991
6. Alkan C, Sajjadian S, Eichler EE (2011) Limitations of next-generation genome sequence assembly. Nat Methods 8(1):61–65
7. Boetzer M et al (2011) Scaffolding preassembled contigs using SSPACE. Bioinformatics 27(4):578–579
8. Pop M, Kosack D, Salzberg S (2004) Hierarchical scaffolding with bambus. Genome Res 14:149–159
9. Assefa S et al (2009) ABACAS: algorithm-based automatic contiguation of assembled sequences. Bioinformatics 25(15): 1968–1969
10. van Hijum S et al (2005) Projector 2: contig mapping for effecient gap-closure of prokaryotic genome sequence assemblies. Nucleic Acid Res 33:560–566
11. Tsai IJ, Otto TD, Berriman M (2010) Improving draft assemblies by iterative mapping and assembly of short reads to eliminate gaps. Genome Biol 11:R41
12. Boetzer M, Pirovano W (2012) Toward almost closed genomes with GapFiller. Genome Biol 13(6):R56
13. Otto TD et al (2010) Iterative correction of reference nucleotides (iCORN) using second generation sequencing technology. Bioinformatics 26(14):1704–1707
14. Ronen R et al (2012) SEQuel: improving the accuracy of genome assemblies. Bioinformatics 28:i188–i196
15. Otto TD et al (2011) RATT: rapid annotation transfer tool. Nucleic Acids Res 39:e57
16. Logan-Klumpler FJ et al (2012) GeneDB—an annotation database for pathogens. Nucleic Acids Res 40(Database issue):D98–D108
17. Quail MA et al (2012) Optimal enzymes for amplifying sequencing libraries. Nat Methods 9:10–11
18. Simpson JT, Durbin R (2012) Efficient de novo assembly of large genomes using compressed data structures. Genome Res 22(3):549–556
19. Li H et al (2009) The sequence alignment/map format and SAMtools. Bioinformatics 25(16):2078–2079
20. Carver T et al (2012) BamView: visualizing and interpretation of next-generation sequencing read. Brief Bioinform 14:203–212
21. Delcher AL et al (1999) Improved microbial gene identification with GLIMMER. Nucleic Acids Res 27(23):4636–4641
22. Stanke M, Morgenstern B (2005) AUGUSTUS: a web server for gene prediction in eukaryotes that allows user-defined constraints. Nucleic Acids Res 22:W465–W467
23. Swain MT et al (2012) A post-assembly genome-improvement toolkit (PAGIT) to obtain annotated genomes. Nat Protoc 7(7): 1260–1284
24. Fonseca NA et al (2012) Tools for mapping high-throughput sequencing data. Bioinformatics 28:3169–3177
25. Parra G, Bradnam K, Korf I (2007) CEGMA: a pipeline to accurately annotate core genes in eukaryotic genomes. Bioinformatics 23(9): 1061–1067

http://www.springer.com/978-1-4939-1437-1

Parasite Genomics Protocols
Peacock, C. (Ed.)
2015, XII, 367 p. 49 illus., 34 illus. in color. With online files/update., Hardcover
ISBN: 978-1-4939-1437-1
A product of Humana Press