Genome assembly and comparison using de Bruijn graphs

Daniel Robert Zerbino

Darwin College

A dissertation submitted to the University of Cambridge for the degree of Doctor of Philosophy

European Molecular Biology Laboratory,
European Bioinformatics Institute,
Wellcome Trust Genome Campus,
Hinxton, Cambridge, CB10 1SD,
United Kingdom.

Email: zerbino@ebi.ac.uk

September 29, 2009

In memory of my mother.

This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and acknowledgements.

This dissertation is not substantially the same as any I have submitted for a degree, diploma or other qualification at any other university, and no part has already been, or is currently being submitted for any degree, diploma or other qualification.

This dissertation does not exceed the specified length limit of 300 pages as defined by the Biology Degree Committee.

This dissertation has been typeset in 12 pt Palatino using LaTeX2ε according to the specifications defined by the Board of Graduate Studies and the Biology Degree Committee.

September 29, 2009 Daniel Robert Zerbino

Genome assembly and comparison using de Bruijn graphs

Summary

September 29, 2009

Daniel Robert Zerbino
Darwin College

Recent advances in sequencing technology made it possible to generate vast amounts of sequence data. The fragments produced by these high-throughput methods are, however, far shorter than in traditional Sanger sequencing. Previously, micro-reads of less than 50 base pairs were considered useful only in the presence of an existing assembly. This thesis describes solutions for assembling short read sequencing data *de novo*, in the absence of a reference genome.

The algorithms developed here are based on the de Bruijn graph. This data structure is highly suitable for the assembly and comparison of genomes for the following reasons. It provides a flexible tool to handle the sequence variants commonly found in genome evolution such as duplications, inversions or transpositions. In addition, it can combine sequences of highly different lengths, from short reads to assembled genomes. Finally, it ensures an effective data compression of highly redundant datasets.

This thesis presents the development of a collection of methods, called Velvet, to convert a de Bruijn graph into a traditional assembly of contiguous sequences. The first step of the process, termed Tour Bus, removes sequencing errors and handles biological variations such as polymorphisms. In its second part, Velvet aims to resolve repeats based on the available information, from low coverage long reads (Rock Band) or paired shotgun reads (Pebble). These methods were tested on various simulations for precision and efficiency, then on control experimental datasets.

De Bruijn graphs can also be used to detect and analyse structural variants from unassembled data. The final chapter of this thesis presents the results of collaborative work on the analysis of several experimental unassembled datasets.

Preface

Oh it's such a perfect day,
I'm glad I spent it with you.
L. Reed

This dissertation presents more than three years of work and life at the European Bioinformatics Institute, during which I have the privilege of encountering many dynamic and interesting people.

I am grateful to Thomas Simonson for patiently accepting to discuss the field of bioinformatics with an aimless engineering student. I also owe to the late Ángel Ortiz and to Véronique Stoven for convincing me that a PhD was not such an irretrievable mistake.

This work would not have been possible without the help and support of Ewan, who unexplainably chose to give me a chance although I came from a completely different background and had not even heard of Ensembl prior to the interview. It has been a extremely positive working relationship, based on fruitful disagreements (read coherency?) and methodical improvisation.

Much of the work discussed in this thesis was done in collaboration with a large number of people. Zamin, always keen to ask questions and get to the depth of the problem, was heavily involved in analysing 1000 genomes data and in developing Cortex. Mario brought his experience of actual assemblies in developing Cortex. Craig and Vrunda were extremely efficient in pushing in a matter of weeks the colorspace implementation of Velvet exclusively through e-mail and despite an 8 hour time difference. Elliott and Gayle collaborated in perfecting Velvet by making suggestions and providing useful *Drosophila* test datasets. Marcel took on the task of carrying the Velvet project over to the much more tricky issue of RNA assembly. Matthias became the EBI expert on running huge jobs on EBI servers, for which I am certain all of the other behemoth2 and behemoth3 users are deeply grateful. Mark Chaisson provided technical assistance in testing EULER. Finally, many thanks to Grégoire Pau and Tim Massing-

ham for kindly going over my attempts at maximum likelihood estimators and putting them into shape.

Developing Velvet was for me an intense and daily interaction with many different users, whose names I will not even try to list exhaustively here. David Studholme was the very first and a regular correspondent since. Jasper Rees brought his enthusiasm into the project, systematically pushing each and every parameter up to eleven. Darren Platt provided systematic and well documented bug reports, going straight into the code to pinpoint the elusive null pointer. Other users such as Lisa Murray, Korbinian Schneeberger, Aylwyn Scally, Weldon Whitener chased down the many Velvet bugs in their pipeline and patiently sent me all their detailed reports.

Among the Velvet users, some even took the initiative to bring their contribution to the Velvet project. Paul Harrison developped a script which allowed the graphical visualisations of de Bruijn graphs based on Velvet's output. Simon Gladman contributed a number of scripts to analyse Velvet's output as well as to automate the setting of the parameters. Curtis Jamison set about to force Velvet and Consed into compatibility. Charles Plessy made Velvet more available by integrating it into the Debian-med package. Torsten Seeman suggested improvements to the useability of the program. Thomas Dan Otto provided results on his *Plasmodium* runs. Last but not least, Jeffrey Cook got his hands dirty in the depths of the code and started to parallelize it. I sincerely hope that my initial clumsiness will not discourage him from going through with his plans.

Thank you also to my thesis advisory committee members, Richard Durbin, Nick Goldman and Lars Steinmetz, for following the project and keeping an eye on Ewan.

This thesis was completed in a timely fashion thanks to the help of a high-throughput farm of parallel proof-readers. Thank you Alison, Micha, Vicky, Dirk, Zam, Marcel, Jeremy and Grégoire.

I am grateful to all my research group for day to day suggestions, ideas and general chit chat. Thank you to Ben, Michael, Alison, Dace, Markus and Micha along with the visitors Kai, Matt, Remo, Florian and Marcel. Many thanks to Shelley for always sorting out everything efficiently, even when I came to her at the last moment. Thanks to all the Ensembl people, especially the "Fish and Chips" volleyball team.

And of course, many thanks to the EMBL-EBI predocs, a cheerful crowd of international students and especially to bon vivant Sylvain, random beekeeping bloke Dom, Berlin (via Paris) bound Joern, punster

Michael, Alison, eh, cheese-eating Jeremy, singing dancing and playing Melanie, twin raising Michael and Daniela, also parenting Kevin, pole-vaulting Florence, procrastinating Heidi, London clubbing Jacky, chocolate addict Elin, boatclub organising Julia, cycling Judith, poker playing Michele and juggling Greg. Thanks for the coffee breaks! Thanks also to Nicolas and Catherine, Benoit, Georg, Grégoire, Mikhail, Kristen, Vicky and Nico for many good moments.

But a PhD is not only about hours in and around the lab. I've thoroughly enjoyed my years in Cambridge and am grateful to the Darwin College Boat Club people for getting me out on the river when I should have been in bed and on erg machines when I should have been at the pub.

Merci à toi Jeanne, qui n'a jamais cessé d'être une amie, et aux gars du BJJ, Denis, Mathieu, François (les deux), Matthias, Laurent, Pascal et Benoit pour des vacances mémorables.

Finalmente, agradezco a mi familia por su respaldo y soporte. A mi padre, quien pusó por un lado sus ansiedades mientras yo entraba en el mundo académico, a mi tía Eugenia, quien siempre fue disponible para mí cuando precisé ayuda, y a mis tías Marta y Mónica, quien siguieron mi trabajo desde el Uruguay, hasta leer las publicaciones científicas bajadas por el Internet.

Contents

List of Figures

List of Tables

List of Acronyms

ATP	Adenosine triphosphate
bp	Basepair
CPU	Central processing unit
ddNTP	Dideoxy nucleoside triphosphate
DNA	Deoxyribonucleic acid
Gb	Giga-basepair
Kb	Kilo-basepair
Mb	Mega-basepair
MLE	Maximum likelihood estimator
PCA	Principal component analysis
pdf	Probability density function
PPi	Inorganic pyrophosphate
RAM	Random access memory
RNA	Ribonucleic acid
SA	Simulated annealing
SBH	Sequencing by hybridisation
SBL	Sequencing by ligation
SBS	Sequencing by synthesis
SD	Standard deviation
SV	Structural variant
WGS	Whole-genome shotgun

Chapter 1

Genome sequencing and assembly

1.1 Introduction

Sequencing technologies have significantly improved since the first genome was read (Sanger et al., 1977a). They remain at the core of genomics and have many practical applications. They are of course used to determine the genome sequence of a new species or of an individual within a population. However, they also provide tools for readout assays in molecular biology, for example when studying RNA transcripts (Adams et al., 1991), gene expression (Velculescu et al., 1995), protein to DNA binding (Kim et al., 2005) or transcription (Carninci et al., 2005). Determining the complete genome sequence of a species is still an important application of sequencing. Despite the success in determining the human (The International Human Genome Sequencing Consortium, 2001; Venter et al., 2001), mouse (The Mouse Genome Sequencing Consortium, 2002) and numerous other genomes, most species in the biosphere have not been sequenced yet.

A critical stage in *de novo* genome sequencing is the *assembly* of *shotgun* reads, in other words putting together fragments randomly extracted from the sample to form a set of contiguous sequences, *contigs*, that represent the DNA in the sample. Algorithms were developed for whole genome shotgun (WGS) fragment assembly, including Atlas (Havlak et al., 2004), Arachne (Batzoglou et al., 2002), Celera (Myers et al., 2000), PCAP (Huang et al., 2003), Phrap (www.phrap.org) and Phusion (Mullikin and Ning, 2003). All these programs rely on the *overlap-layout-consensus approach* (Pevzner et al., 2001) where all the reads are compared to each other in a pair-wise fashion. They have proved their utility through numerous genome assemblies (The International Human Genome Sequencing Consortium, 2001; The Mouse Genome Sequencing Consortium, 2002).

Recently, new sequencing methods have emerged (Mardis, 2008). Commercially available technologies include pyrosequencing (454 Sequencing (Margulies et al., 2005)), sequencing by synthesis (Illumina (Bentley, 2006)) and sequencing by ligation (SOLiD - www.appliedbiosystems.com). Compared to traditional Sanger methods, these technologies function with significantly lower production costs and higher throughput. These advances have significantly reduced the cost of several sequencing applications, such as re-sequencing individuals (Hillier et al., 2008) or readout assays (e.g. ChIP-seq (Johnson et al., 2007), RNAseq (Mortazavi et al., 2008)).

However, the reads produced by these next-generation sequencing technologies are much shorter than traditional Sanger reads, currently around 400-500 basepairs (bp) for pyrosequencing, 50bp for Illumina and 35bp for SOLiD. Very short reads (typically $\leq$ 50bp) are not well suited to the assembly approach described above. Because of their length, they must be produced in large quantities and at greater coverage depths than earlier sequencing projects. The sheer number of reads makes the traditional approach, with a quadratic number of alignments with respects to the number of reads, extremely lengthy to compute. Whereas long reads provide long overlaps, to disambiguate repeats from real overlaps, short reads within repeats offer fewer differences to judge from (Whiteford et al., 2005). These issues have led several research teams to design *de novo* assembly tools specifically for these very short reads.

This chapter will decribe the different techniques used in sequencing and the constraints that they impose on the assembly stage. It will also briefly describe the general approaches used in *de novo* assemblers. More detailed comparisons of these algorithms will be provided in the following chapters.

1.2 Traditional genome shotgun sequencing and assembly

The sequencing field has for many years been predominantly based on the Sanger sequencing method (Sanger et al., 1977b). It has proved extremely successful in producing many completed genomes (The International Human Genome Sequencing Consortium, 2001; The Mouse Genome Sequencing Consortium, 2002), thanks to constant technological improvements (Collins et al., 2003). In short, to sequence a genome, one must first extract DNA, break it up into useable fragments, clone those fragments, then sequence their tips. In the end, one obtains a collection of DNA sequences which must be assembled together.

1.2.1 DNA extraction

In practice, collecting enough cells to obtain the desired amount of DNA can be difficult, according to the size of the available samples or the possibility of obtaining a clonal cell culture. It has been found that many bacterial organisms are not viable when isolated from their normal environment (Walker and Parkhill, 2008). Similarly, when analysing human tumors, researchers can be interested in mutations present in isolated tumorous cells (Klein and Hölzel, 2006). The possibilty to sequence a single cell would provide detailed insight into such studies.

The techniques described below require different amounts of DNA to function properly. Also, some protocols such as long inserts (cf. 1.2.2) are more wasteful of DNA. This is why researchers are trying to improve the sequencing techniques and make them more efficient in their use of samplesample material. Various techniques such as micro-fluidics (Meldrum and Holl, 2002), improved amplification (with, for example, Multiple Displacement Amplification (Lasken, 2007), polymerase cloning (Zhang et al., 2006) or emulsion PCR (Williams et al., 2006)) or single molecule sequencing (Harris et al., 2008) are being developed for this objective, but a complete operational pipeline has not been achieved yet.

1.2.2 Cloning strategy

Once extracted, the DNA is randomly fragmented into size-selected sequences. These pieces are inserted into vectors, which are then transfected into bacteria, generally *E. coli*. This process allows the construction of fragment libraries which can be consulted repeatedly according to experimental requirements (Brown, 2007). It is also a convenient way to amplify DNA samples for the next step, chain-termination sequencing.

The traditional method, *shotgun sequencing* (Anderson, 1981), consists of sequencing random fragments of the sample, then assembling them together computationally (see below). This strategy works well on short and unrepetitive sequences, such as bacterial genomes, but is significantly more difficult in long, repeat-rich genomes, such as mammalian ones (The International Human Genome Sequencing Consortium, 2001).

One solution is *hierarchical shotgun sequencing* (The International Human Genome Sequencing Consortium, 2001). To simplify the analytical assembly of the reads, the problem is biochemically subdivided into random clonal regions, at the cost of having to *map* them onto the genome. This is done by digesting the clones with a restriction enzyme and measuring the lengths of the fragments by electrophoresis to obtain a *fingerprint* of each clone. The fingerprints are then compared to determine overlaps and construct sequences of contiguous clones, also called *contigs*. These are then mapped onto the chromosomes using for example known locus-specific *sequence-tagged sites* (STS). Once the clones are mapped, a subset of them are selected that cover the genome while reducing redundancy.

These clones are then sequenced separately.

Another approach is *whole genome shotgun* (WGS) sequencing (Venter et al., 2001) where the shotgun strategy is extended to the scale of the genome, obviating the need for a clonal map. This approach makes the assembly stage much more complex, as all the reads have to be processed together. It requires a sufficiently robust algorithm to avoid *misassembling* non-consecutive regions of the genome together.

1.2.3 Paired-end reads

Paired-end reads, also called *mate pairs*, are often created to resolve and validate the assembly process (Edwards et al., 1990). These read pairs are produced by sequencing a DNA fragment from both ends. Given that sequencing reads are typically much shorter than the cloned fragments, they generally do not overlap. Nonetheless, prior size selection of the fragments provides important information about the relative placement of the reads on the genome.

The selection of the optimal insert length to be used in mate pair libraries depends on various factors, including the repeat structure of the genome being sequenced and practical limitations of the paired-end sequencing techniques (Roach et al., 1995). Generally, sequencing projects create libraries with different insert lengths in order to obtain better results than with just one length.

1.2.4 Chain-termination sequencing

This step is where the actual reading of DNA takes place. Commonly known as Sanger sequencing (Sanger et al., 1977b), dye-termination electrophoretic sequencing relies on an *in vitro* replication using DNA polymerase, ATP and nucleoside triphosphates. Modified nucleotides, dideoxy nucleoside triphosphates (ddNTP), are added into the sample. These do not have a 3′ OH group, thus preventing any extension, and are linked to a fluorophore which is specific to the base.

Multiple copies of the sequence of interest are generated in this mixture. They all start at the same primer but are randomly interrupted by a ddNTP nucleotide. These variable length copies are then separated depending on their size using electrophoresis. By associating the fluorescent signal to molecular weight, it is possible to obtain a reliable estimate of the DNA sequence. This process, called *base-calling* consists of transforming four simultaneous and continuous fluorescence intensity signals into a sequence of discrete nucleotides (Ewing et al., 1998). This sequence must finally be clipped to remove low-quality regions and sequences which belong to the cloning vector (Mott, 1998).

1.2.5 Assembly

Once sequencing reads have been produced, it is necessary to tie them together in a coherent manner or assemble them. Many tools have already been developed to address this question, for example the Phrap (www.phrap.org), ARACHNE (Batzoglou et al., 2002) and the Celera (Myers et al., 2000) assemblers. Without going into the specifics of each tool, they share a common paragdim, sometimes referred to as *overlap-consensus-layout* (Pevzner et al., 2001). This approach is quite similar to the one generally used when solving a jigsaw puzzle, as described below.

The first step consists of aligning the fragments two-by-two in an exhaustive fashion and establishing which pairs of reads present a consistent overlap with one another. This is analogous to searching for pieces of the puzzle which fit each other and have matching colours. Especially in eukaryotic genomes, the main difficulty at this stage is to distinguish inexact overlaps due to sequencing errors and those due to similarites within the genome, such as highly conserved repeats (Phillippy et al., 2008). The former should be maintained despite the signal noise, whereas the latter should be removed, to avoid misassemblies.

Sequence alignment is a well studied area of bioinformatics. It consists of providing the optimal alignment of two sequences for a given scoring function. Most methods are based on the Needleman-Wunsch algorithm (Needleman and Wunsch, 1970) which dynamically explores the space of possible alignments of progressively longer subsequences of the compared sequences. Many extensions have since been designed, for example for multiple alignments (Higgins and Sharp, 1988), local alignments (Smith and Waterman, 1981) or accelerated searches through large databases (Altschul et al., 1990). Current research is allowing these techniques to be run much faster on parallel processors (Fagin et al., 1993).

The assembler then detects clusters of reads which consistently align with each other, thus forming contiguous sequences (or *contigs*). This is equivalent having parts of the image put together in a puzzle. In both genome and puzzle assembly, the process is interrupted at either ambiguous areas, where several continuations are possible or at gaps, where no connecting piece has been found.

Finally, the assembler attempts to order and orient the contigs with respects to one another. Returning to the puzzle simile, this corresponds to placing the corners and identifiable parts of the image relative to each other. Using for example paired-end information, the assembler can estimate the distance that separates contigs. Sets of contigs which can all be plausibly placed together in the same region are sometimes called *supercontigs* (ARACHNE) or *scaffolds* (Celera).

1.2.6 Genome finishing

At the end of the assembly stage, a set of scaffolded contigs are available. However, for practical purposes, it is desirable to remove as many gaps as possible, eventually converging towards unbroken chromosomal assemblies. This stage, called finishing, can be quite costly and lengthy, as shown by the progress made by the Human Genome Project. Years after the official draft release (The International Human Genome Sequencing Consortium, 2001), the project is still confronted with a handful of gaps (Cole et al., 2008).

Gaps can be due to two factors: lack of coverage or sequence complexity. Coverage gaps are generally linked to sampling biases in the sequencing method, often referred to as cloning bias. For example, in the case of Sanger sequencing, some genomic regions could not be viably cloned into *E. coli* (Bentley, 2008). Next-generation sequencing technologies are not devoid of bias either. For example, A+T content and self-hybridisation by palindromes (i.e. nucleotide sequences which are their own reverse complements) is known to affect the coverage depth of Illumina sequencing projects (Hillier et al., 2008; Ossowski et al., 2008).

Complex regions actually include a wide variety of features, which appear at different scales. At the long end of the spectrum, segmental duplications and dispersed repeats create high complexity ambiguity, where long stretches of the genome are very similar, thus hard to distinguish. At the lower end, regions such as the centromeres typically present long repetitions of the same short motif, also creating ambiguity.

To span over these gaps, it is generally necessary to resort to additional experimental procedures on a case-by-case basis (The International Human Genome Sequencing Consortium, 2004). For example, if clonal libraries are available, it is possible to analyse individual clones which may contain a single copy of a repeat or a difficult region. Otherwise, it is necessary to use alternative chemistries or directed PCR.

1.3 Next-generation sequencing

Although reliable and considerably optimised, the complete Sanger process is quite long and involves major costs, in terms of infrastructure, reagents, sample DNA and labour. The overall monetary cost of a genome was estimated in 2005 as roughly US$12 million for a mammalian genome (Metzker, 2005). Because of the central role of sequencing in current biological research, significant resources are being invested to reduce its cost to below $100,000 and ultimately below $1000 per mammalian genome (Service, 2006).

Important breakthroughs have recently been made to streamline the sequence pipeline. Many instrument makers are trying to enter this lucrative market (Mardis, 2008) (Metzker, 2005). Below are briefly described the techniques which are currently commercialised, namely *454*, *Illumina* and *SOLiD*. They produce shorter reads with higher error rates, but are much faster and cheaper than Sanger sequencing. Moreover, they generally require less DNA sample to function.

1.3.1 454 sequencing

454 sequencing is a technology currently commercialised by the Roche Applied Science corporation. It was the earliest next-generation sequencing technology to enter the market in 2004. It achieves much higher throughput than Sanger sequencing by cheaper emulsion PCR amplification and massively parallel pyrosequencing reactions (Margulies et al., 2005).

Sample preparation and amplification by emulsion PCR

The DNA to be sequenced is first broken up into fragments which are fitted with adapter oligonucleotides on either end and denaturated. The resulting single-stranded fragments are then hybridised along their adaptor sequences to oligonucleotides on the surface of agarose beads. The solutions are prepared so that very few beads are bound by more than one sample fragment.

The DNA is then amplified on the surface of each bead. To avoid cross contamination between the sequences attached to different beads, the PCR process is performed in a mixed oily and aqueous emulsion (Williams et al., 2006). Each hydrophilic micelle forms a compartment containing beads and reagents. The process is adjusted so that there are many more micelles than beads, thus ensuring that few compartments contain more than one bead.

High-throughput sequencing on picotiter plates

The beads, now covered with numerous (up to 1,000,000) exact copies of the original sample DNA fragment, are then inserted into 44μm wide wells on the surface of the 454 *picotiter plate* (PTP). Each well is then filled up with smaller 1μ wide latex beads which are bound to the enzymes necessary for the final stage, pyrosequencing. This setup allows the system to simultaneously track hundreds of thousands of simultaneous reactions and is key to the high throughput of the sequencer. Because the wells are made from fibreoptics, they allow a direct observation of the pyrosequencing reaction.

Pyrosequencing

Pyrosequencing (Ronaghi et al., 1996) is based on detecting the incorporation of nucleotides during replication via the measure of the corresponding release of pyrophosphate molecules (PPi). The sulfurylase enzyme converts the released PPi into ATP which is then used by the luciferase enzyme to emit photons (Ronaghi et al., 1996).

To control which base is being incorporated, the nucleotides are added sequentially, then degraded locally by apyrase. The system associates the timing of the light emissions with the injection of nucleotides to infer which bases are being incorporated. The amount of light measured is correlated to the amount of bases being incorporated. In other words, if the sequence being extended integrates n identical nucleotides in a row, then the corresponding emission will be n times larger.

The accuracy of the system is sensitive to the precise measurement of the light intensity, especially at high values. 454 datasets have been observed to suffer especially from errors in the length estimate of poly-N runs (such as 'AAAAAA') Quinlan et al. (2008). For this reason, 454 datasets are generally provided as *flowgrams*. In this representation, reads are not reported as straightforward sequences of nucleotides, but as sequences of cycles, each with an estimated multiplicity, as shown on Figure

1.1. By explicitly separating the basecall from the estimation of multiplicity, flowgrams allow errors from each process to be analysed separately.

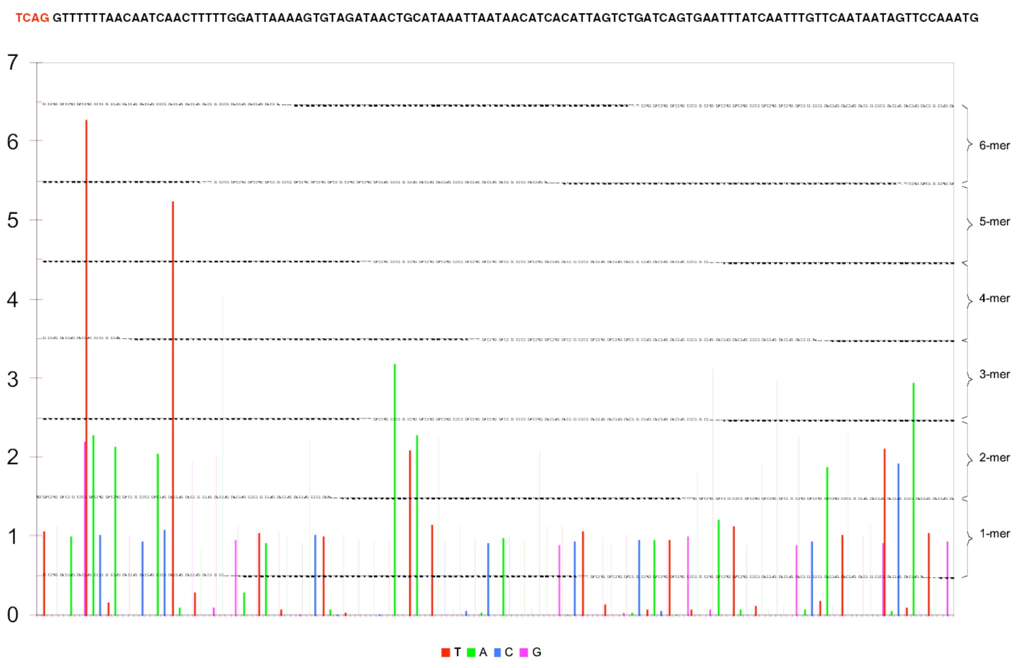


Figure 1.1: Example of flowgram (from (Ronaghi et al., 1996))

This flowgram represents the signal measured when pyrosequencing the sequence indicated at the top of the figure. The columns represent intensity measurements and are placed from left to right in chronological order, with four alternating colours to indicate the base being added. The system iteratively introduces one of the four nucleotides in the reaction wells and estimates how many are integrated per sequence copy.

1.3.2 Illumina sequencing

Illumina sequencing originally Solexa sequencing, was commercialised later in 2006. Although Illumina reads are much shorter than 454 reads, their cost per base is significantly lower and their throughput higher, thanks to very different amplification and sequencing techniques.

Bridge amplification

DNA is initially fragmented, attached to specific adapter sequences and denaturated. These modified single-stranded fragments are then poured into one of the eight lanes of a *flowcell*. The surface of each lane is treated so as to bind covalently to the end of the nucleotide chains. Reverse complementary copies of the adapters are also attached to the surface of the flowcell, but at a much higher concentration. The objective is that the fragments are randomly spread far apart (with regard to their length), but are surrounded by many oligos.

The sample fragments are then amplified locally, as shown on Figure 1.2. At each cycle, each single-stranded fragment hybridises to a nearby oligonucleotide, thus forming a *bridge*. This primer is used to start a standard replication reaction, thus producing a reverse complement copy of the sample fragment in its direct vicinity. After several iterations, the lane is covered with clusters of oligonucleotides which are identical or reverse complements of the single fragment which initiated the reaction. The diameter of the clusters on the flowcell is limited by the short length of the fragments, typically between 100 and 800 bp (Dirk Evers, pers. comm.)

Sequencing by synthesis

Now that the fragments have been turned into clusters, the system reads out the sequences of all the clusters simultaneously. This is achieved using standard replication machinery with modified nucleotides. The engineered nucleotides are attached to a fluorescent group corresponding to the base and their 3′ OH group is modified to temporarily interrupt replication after their incorporation. This modification is reversed chemically after each cycle to allow the process to iterate.

Due to these modified nucleotides, the replication process can be controlled step by step and followed visually, as shown on Figure 1.3. The reaction starts with primers which hybridise to the adaptor sequences. The engineered nucleotides are then added to the solution and incorporated,

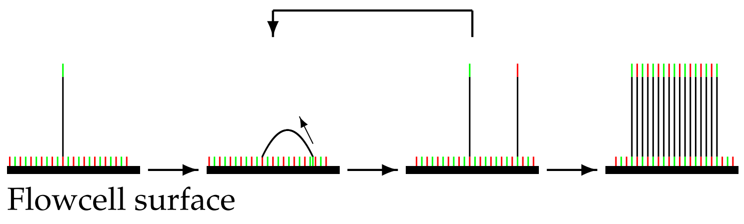


Figure 1.2: Schematic diagram of bridge amplification

In the first step, a single-stranded DNA fragment (thin black line) with ligated adaptors (red and green) is bound to the surface of the flowcell (thick black line), surrounded by many reverse complementary adaptors (red and green) which are also bound to the flowcell. In the second step, The unattached end of the fragment randomly hybridises to one of the nearby adaptors and a replication reaction is initiated. In the third step, once the replication is over, a reverse complementary copy of the sequence is attached to the flowcell near the template. The repetition of the second and third steps leads finally to a cluster of direct and reverse complementary copies of the template in the neighbourhood of the original sequence.

interrupting the extension of the synthesised strand. In the next step, the unincorporated nucleotides are washed away. Using the fluorescence of the nucleotides, amplified by the spatial clustering, a digital camera verifies which base was added to the sequence. Finally, the fluorophores are removed and the nucleotides unblocked, allowing for the next iteration to commence.

1.3.3 SOLiD sequencing

SOLiD sequencing originally developed by Agencourt Genomics and now produced by Applied Biosystems, was the latest machine to be distributed, in 2007. It is characterised by its di-nucleotide encoding which can be used to improve the reliability of alignments. Its reads are shorter than Illumina reads, but the throughput per machine is higher (Mardis, 2008).

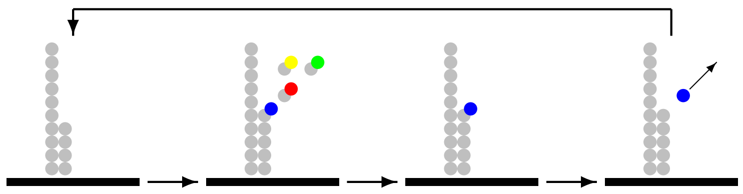


Figure 1.3: Schematic diagram of sequencing by synthesis

In the first step the single-stranded DNA to be sequenced is hybridised by a primer. In the next step, all modified nucleotides are introduced into the flowcell and one is ligated to the primer, along the template. Because of its modified 3′ OH group, elongation is blocked. In the third stage, unincorporated nucleotides are washed away, leaving only the ligated ones to be observed by fluorescence. Finally, the fluorophore is removed and the 3′ OH group of the artificial nucleotide is reactived, allowing the process to reiterate, one basepair farther downstream.

Sample preparation and fixation

The preparation of SOLiD samples uses some of the techniques described above. DNA fragments are linked to adapter sequences, denaturated, then attached to micro-beads. The sequences around the beads are then amplified by emPCR. The beads, surrounded by multiple copies of the same fragment, are then attached to a slide that is inserted into the reaction cell.

Sequencing by ligation

Once in the reaction cell, the fragments are bound by a universal primer which hybridises onto a specific section of the adapter, close to the ligation site with the actual sample DNA. Semi-degenerate probe sets are then iteratively added to the reaction cell and tested.

The design of these 8bp probes is quite complex, as shown on Figure 1.4. All of the nucleotides are degenerate, except those at positions 4 and 5. On the 3′ end, a fluorescent group is attached, the colour of which is associated to the combination of the fixed nucleotides (see below). A cleavage site is inserted between positions 5 and 6.

$$\begin{tikzcd}
 & & & & & & & & Fluo \\
N \arrow[r] & N \arrow[r] & N \arrow[r] & A \arrow[r] & T \arrow[r, bend left, "\text{Cleavage site}"] & N \arrow[r] & N \arrow[r] & N \arrow[ur] &
\end{tikzcd}$$

Figure 1.4: Schematic diagram of a SOLiD probe (first 3 cycles)

The initial SOLiD probe is made of 6 degenerate bases (N) surrounding 2 predetermined bases (in this example A and T). A cleavage site separates the 5^{th} and 6^{th} bases. A fluorophore is attached on the 3′ end of the sequence, charactesing the fixed bases as described on Figure 1.7

As the probes are inserted, they can bind wherever possible on the sample DNA. However, when a probe binds right next to the primer, then a ligase protein joins the two together, stabilising the probe. Once this reaction is over, a fluorescence readout determines the nature of the probes around each bead. The probes are then cleaved between positions 5 and 6, thus serving as primers, offset by 5bp, during the next iteration.

Because the iterations are repeated at 5bp intervals, it is necessary to read the fragment in five cycles, each offset by 1bp from the previous one. After finishing a cycle, all the probes and primers are removed. In the next two cycles, slightly different primers are used, which terminate 1bp and 2bp respectively upstream of the termination site of the original primer.

For the final two cycles, instead of using 2 different kinds of primers, offset by 3 and 4bp respectively, the system uses different probes, where fixed nucleotides are at the 5′ end, as shown in Figure 1.5.

After all these operations, all di-nucleotide words of the sequence have been queried, as represented in Figure 1.6.

Colourspace analysis

One of the peculiarities of the SOLiD method is the reporting of di-nucleotide sequences, familiarly known as *colourspace*. In the sequencing

$$\begin{tikzcd}
 & & & & & & & & Fluo \\
{\color{red}A} \arrow[r] & {\color{red}T} \arrow[r] & N \arrow[r] & N \arrow[r] & N \arrow[r, bend left, "\text{Cleavage site}"] & N \arrow[r] & N \arrow[r] & N \arrow[ur] &
\end{tikzcd}$$

Figure 1.5: Schematic diagram of a SOLiD probe (last 2 cycles)

The design of the alternative SOLiD probe is identical to the first, except that the fixed bases (in this example A and T) are placed on the 5′ end of the probe.

by ligation step, each fluorescent colour is not associated to a specific base, as in most other methods, but to a set of two nucleotide sequences (e.g. AT, CG etc.). Because there are only four fluorophores available and 16 possible di-nucleotide combinations, the code is necessarily degenerate, meaning that each colour can be the result of several different sequences (cf. Figure 1.7 for a summary of the code).

Knowing the first nucleotide of the sequence (which is part of the adapter) and the sequence of transitions, it is possible to recreate the nucleotide sequence of any read. However, this direct approach is error-prone. Any error in the colour sequence creates not only an isolated error, but completely transforms all the sequence downstream. This is why subsequent analysis of SOLiD reads is generally performed in colourspace.

Although seemingly counter-intuitive, this approach can be useful as an error-correction code when aligning two colour encoded sequences. Supposing that most biological variants are isolated SNPs, it is possible to search for them by aligning the reads to the reference, both of them in colourspace. At the locus of any actual SNP, one would observe two di-nucleotide changes, one either side of the SNP. Because of the design of the code, this results necessarily into a two colour change. In other words, any isolated colour change can be disregarded as a sequencing error. Moreover, given the assumptions, the set of possible two colour transitions is limited to a small set, providing an extra layer of error detection.

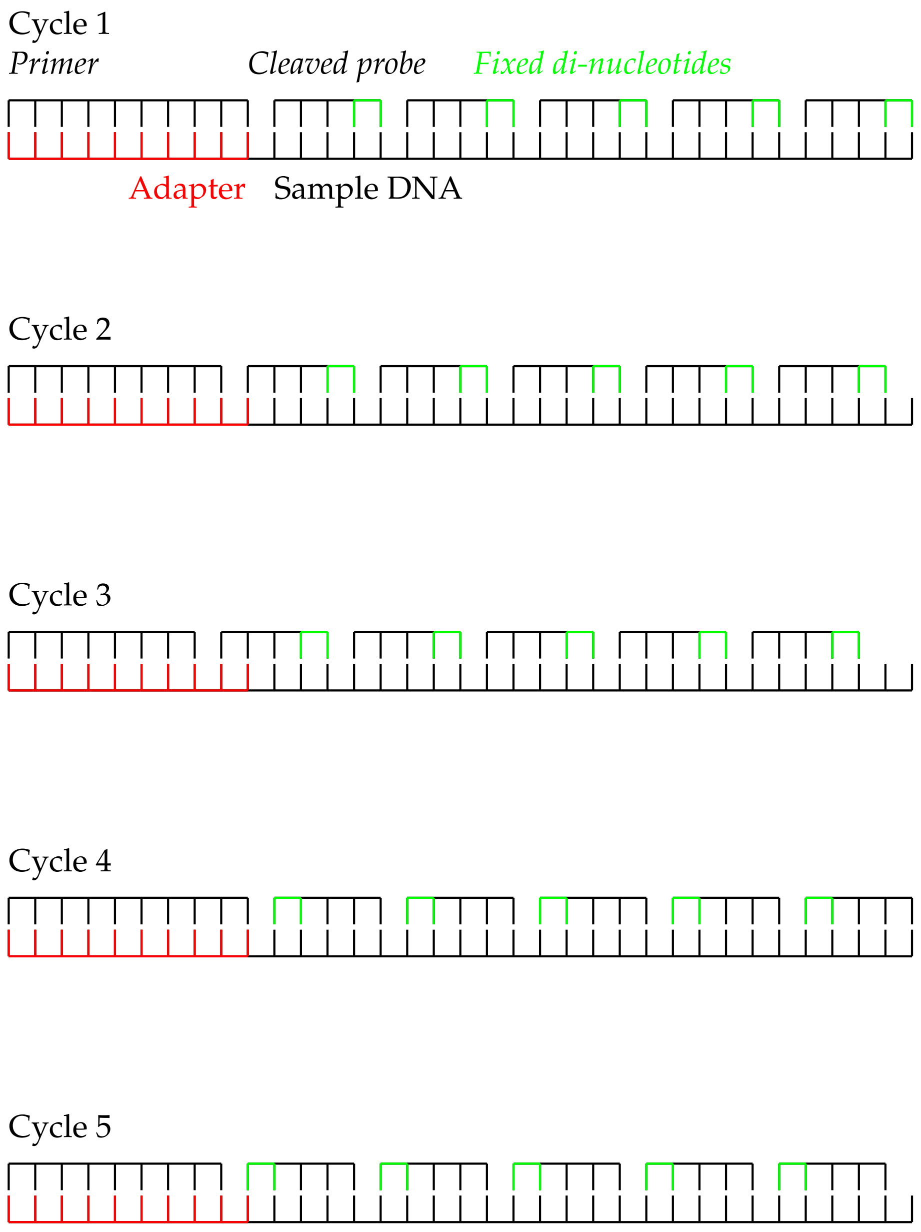


Figure 1.6: Schematic summary of SOLiD ligation cycles

This series of diagrams represents the ligation of probes onto the template. During each cycle, the template sequence, below, is hybridised from left to right by a specific universal primer, then by a series of probes. Each probe is ligated onto the template after the previous one was cleaved. The fixed bases of each probe are colored in green.

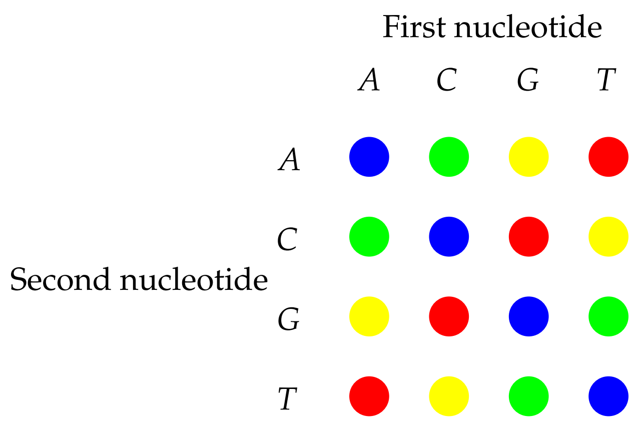


Figure 1.7: Di-nucleotide to colourspace encoding

This table of colours determines the colour of the fluorophore attached to a probe depending on the fixed bases on the probe.

1.4 *De novo* assembly of next-generation sequencing reads

1.4.1 Traditional methods

The traditional approach to assembly can be formalised using the overlap graph (Myers, 1995). This structure represents each read as a separate node, where two reads presenting a clean overlap are connected by a bi-directed edge. The overlap-consensus-layout approach is both intuitive and robust, especially in the case of long reads. It is possible, through various heuristics, to determine reliably whether an overlap is genuine because each pair-wise alignment is made over long sequences (cf. Phrap documentation www.phrap.org).

However, this approach presents several disadvantages, especially in the case of next-generation microread sequencing. Indeed, the computation of pairwise overlaps, although it can be optimised by heuristics (Pearson and D.J.Lipman, 1988) or filters (Rasmussen et al., 2005), is inherently quadratic in complexity. Only one microread assembler, EDENA (Hernandez et al., 2008) was developed using this approach.

Included reads, i.e. reads which align over their whole length onto another read, have to be removed from the graph (Myers, 1995). This entails that mixed-length sequencing cannot be performed directly with an overlap graph. Various programs attempt to perform mixed-length assemblies on overlap graphs (Chevreux, 2005; Reinhardt et al., 2009), but they have to introduce an asymmetry in the use of the reads. Either short reads are simply mapped onto long read contigs or they are assembled separately.

1.4.2 Hash searches

To cut down on the quadratic amount of calculations required to construct the overlap graph, some developers opted for a well studied class of data structures in computer science, the prefix tree.

The basic concept is to iteratively extend a contig through a greedy search. The program initially chooses a read at random to form a singleton. At each step, it searches through its database for reads which correctly overlap the current contig. From this reduced set of reads, the program then uses a heuristic to determine the appropriate extension to be added to the contig. After running this process several times and exhausting all possible reads, the contigs can be merged using traditional assembly software, such as Phrap. A summary of published methods is described in Table 1.1.

1.4.3 De Bruijn graph based approaches

In 1995, Idury and Waterman (Idury and Waterman, 1995) introduced the use of a sequence graph to represent an assembly. They presented an assembly algorithm for an alternative sequencing technique, sequencing by hybridisation, where an oligoarray could detect all the k nucleotide words, also known as k-mers, present in a given genome. Their resolution method consisted in creating a node for every detected word, then connect the nodes corresponding to overlapping k-mers. They could then report

Table 1.1: Comparison of search-based assembly programs

Program	Reference	Heuristic criterion
SSAKE	R.L. Warren *et al* (Warren et al., 2007)	Coverage, overlap length
VCAKE	W.R. Jeck *et al* (Jeck et al., 2007)	Coverage, overlap length, majority count
SHARCGS	J.C. Dohm *et al* (Dohm et al., 2007)	Overlap length, consensus.
QSRA	D.W. Bryant Jr. *et al* (Bryant Jr. et al., 2009)	Coverage, overlap length, quality scores, majority count.

chains of overlapping *k*-mers which unambiguously produced contigs, because of an absence of branching connections.

Pevzner et al. (2001) expanded on this idea. Firstly, they proposed a slightly different formalisation of the sequence graph, called the *de Bruijn* graph, whereby the *k*-mers are represented as arcs and overlapping *k*-mers join at their tips. It shall be presented in more detail in the next chapter. In a succession of papers the same group presented algorithms to build and correct errors in the de Bruijn graph (Pevzner et al., 2001), use paired-end reads (Pevzner and Tang, 2001) or short reads (Chaisson and Pevzner, 2008).

This thesis presents the development of a set of tools, collectively called *Velvet*, which were designed for the *de novo* assembly of microreads. Velvet implements a graph structure that is slightly different to Pevzner *et al*'s implementation as explained in Chapter 2. It contains novel algorithms for graph construction, error removal, mixed length assembly and paired-end assembly. Moreover, this program was designed to be robust and easy to run. It was successfully used by external users in several studies (Ossowski et al., 2008; Farrer et al., 2009; Wakaguri et al., 2008; Collins et al., 2008).

Simultaneously, other projects were adopting the use of the de Bruijn graph. Programs such as ALLPATHS (Butler et al., 2008) and SHORTY (Chen and Skiena, 2007) specialised in localising the use of paired-end reads. Other assemblers, for example ABySS (Simpson et al., 2009) and (Jackson et al., 2009) successfully parallelised the construction of the de Bruijn graph, thus removing practical memory limitations on assemblies.

1.5 Overview of this thesis

The following chapters describe in detail the different elements of Velvet. Chapter 2 describes its specific implementation of de Bruijn graph and the construction method used. Chapter 3 describes how errors are removed based on the topology and coverage of the graph. Chapter 4 describes how repeats are resolved using mixed length datasets and paired-end reads. The last chapter describes how the methods and tools developed in this thesis can be used to align and analyse unassembled data, in particular to detect structural variants.

Chapter 2

De Bruijn graphs

Although directly inspired by the ideas of Pevzner et al. (2001), the graph structure implemented in Velvet is different in several respects. Firstly, it maps k-mers onto nodes instead of arcs. Secondly, it associates reverse complementary sequences to obtain an implicit bi-graph (or bi-directed graph) (Medvedev et al., 2007), in other words a graph where an edge can indepentently enter or exit the nodes at either of its ends. Moreover, the construction method described here is novel.

2.1 Definitions

Each *node* N represents a series of overlapping k-mers (cf. Figure 2.1 for a small example). Adjacent k-mers overlap by $k-1$ nucleotides. The marginal information contained by a k-mer is its last nucleotide. The sequence of those final nucleotides is called the *sequence* of the node or $s(N)$. The sequence of a node is therefore an incomplete representation of the corresponding k-mers. In other words, two distinct sets of k-mers can be represented by two separate nodes having the same sequence. Despite having the same sequence, these two nodes are nonetheless kept separate, and the reads are mapped onto them according to the underlying k-mers, as explained in 2.4.2.

Each node N is attached to a *twin node* $\tilde{N}$, which represents the reverse series of reverse complement k-mers. This ensures that overlaps between reads from opposite strands are taken into account. It is important to note that the sequences attached to a node and its twin do not need to be reverse complements of each other.

The union of a node N and its twin is called a *block*. From now on any change to a node is implicitly applied symmetrically to its twin. A block therefore has two distinguishable sides, in analogy to the k-mer edges described in Pevzner's 2001 paper (Pevzner et al., 2001). The blocks can be considered as the nodes of an implicit bi-graph (Medvedev et al., 2007).

Nodes can be connected by a directed edge or *arc*. In that case, the last k-mer of an arc's origin node overlaps with the first of its destination node. Because of the symmetry of the blocks, if an arc goes from node A to B, a symmetric arc goes from $\tilde{B}$ to $\tilde{A}$. Any modification of one arc is implicitly applied symmetrically to its paired arc.

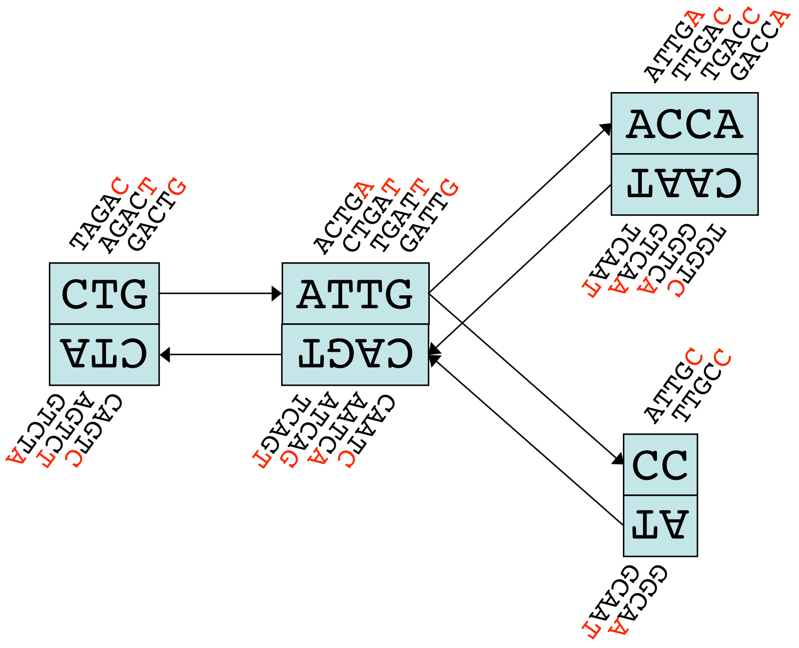


Figure 2.1: Schematic diagram of the de Bruijn graph implementation

Each node, represented by a single rectangle, represents a series of overlapping k-mers (in this case, $k = 5$) listed directly above or below. The last nucleotide of each k-mer is coloured in red. The sequence of those final nucleotides, copied in large letters in the rectangle, is the sequence of the node. The twin node, directly attached either below or above the node, represents the reverse series of reverse complement k-mers. Arcs are represented as arrows between nodes. The last k-mer of an arc's origin overlaps with the first of its destination. Each arc has a symmetric arc. The two nodes on the left could be merged into one without loss of information, because they form a chain.

2.2 Properties

2.2.1 Representation of individual sequences

In the de Bruijn graph, there is a one-to-one mapping of sequences onto paths traversing the graph. Extracting the nucleotide sequence from a path is straightforward given the initial *k*-mer of the first node and the sequences of all the nodes in the path. Conversely, for every read there exists exactly one path which goes sequentially through the nodes corresponding to the sequence's *k*-mers. It is important to remember this bijection between sequences and paths as it underlies all the algorithms presented in the following chapters.

2.2.2 Representation of alignments

By virtue of this relationship, two overlapping sequences are represented as two paths which overlap. The intersection of the paths corresponds to the overlap between the sequences. The two paths form a sub-graph the topology of which is directly linked to the type of alignment between the sequences. If one sequence is a substring of the other, then its path is a sub-path of the other's path. If the sequences align along their tips, then their paths will also be connected at their extremities.

When adding more sequences, all of the above properties remain valid. This means that all sequences which share a common substring are constrained to go through the path corresponding to that substring. This property will be useful when searching for sets of overlapping reads through repeats, as they all follow the same path.

2.2.3 Representation of self-alignments

It is important to note that alignments of a read onto itself or self-alignments, are also represented in the de Bruijn graph. For example, if a sequence contains an exact repeat longer than *k*, then this repeat is represented by a unique path, through which the sequence path goes multiple times, looping as necessary.

Since we have implemented de Bruijn graphs as bi-graphs, reverse complementary alignments also affect the topology of the graph. For example, a genomic palindrome can be considered as the repetition of a sequence which is present in its direct form then in its reverse complementary form. Once again, this substring is represented as a single path through the graph. The overall sequence therefore goes twice through that path, once in either direction. This creates a "hairpin" structure in the graph, analogous to the secondary structure of RNA loops.

2.2.4 De Bruijn and overlap graphs

The "consensus-layout-overlap" assembly methods use the overlap graph structure which is quite similar to the de Bruijn graph (Medvedev et al., 2007). Nodes represent sequences and paths possible assemblies. In their original paper, Pevzner *et al* stressed the fact that the edges were labelled, not the nodes. However, a de Bruijn graph can be obtained with labelled nodes (Medvedev et al., 2007) and an overlap graph can be represented with labelled edges (Myers, 2005). Seemingly, the de Bruijn graph could be identified to the overlap graph.

However, the fundamental difference lies in the scale of resolution of these different structures. A de Bruijn graph is inherently *k*-mer centric, meaning that its topology is unaffected by the fragmentation of the reads. For example, supposing perfect reads and complete coverage, whether one is building a graph from short or long fragments or even the complete genomic sequence, does not affect the structure of the graph. On the contrary, the structure of an overlap graph depends directly on the length and placement of the fragments.

The first consequence is that a de Bruijn graph can accommodate sequences of very different lengths. This is especially useful when attempting mixed-length sequencing or comparative genomics. No *ad hoc* approximation has to be made to mix short reads, long reads, pre-assembled contigs or even finished genomes.

Secondly, because of the one-to-one relationship between paths and sequences, overlapping sequences necessarily follow the same path. This simplifies the search for consistently overlapping sets of reads, as described in Chapter 4.

Finally, de Bruijn graphs have been regarded as heavy structures which require a lot of physical memory, but recent advances (Simpson et al., 2009; Jackson et al., 2009) have proven that they can efficiently be distributed on a cluster of small commodity computers.

On the other hand, the de Bruijn graph does not integrate all the calculations needed to produce the overlap graph. Especially with long reads, global pairwise alignments can be extremely useful to determine reliably if two reads really come from the same genomic locus. Constrained by the k-mer length, the de Bruijn graph can accumulate false-positive overlaps which could have easily been detected by pairwise alignment. These false-positive overlaps make the resolution of repeats as described in Chapter 4 even more complex.

2.2.5 De Bruijn graphs and string indices

It is interesting to note that conceptually, the de Bruijn graph of a sequence can be considered as a simplification of that sequence's *affix tree*. This structure is a bidirectional generalisation of the suffix tree (Maaß, 2008). Using the atomic tree representation (Giegerich and Kurtz, 1997), each non-empty substring of a given sequence is mapped onto a separate node. Each node is connected by an *edge* to its longest prefix and by a *suffix link* to its longest suffix. Nodes corresponding to sequences of length 1 are directly connected to a root node, corresponding to the empty string.

If we rank the nodes by distance from the root, the k-mer nodes of the de Bruijn graph correspond to the nodes of rank k in the affix tree. It is easy to demonstrate that two k-mers are connected in the de Bruijn graph if and only if the corresponding nodes in the affix tree are connected by a path composed of an edge and a suffix link, going through a node of rank $k + 1$. Traversing the de Bruijn graph is therefore equivalent to traversing the affix tree across its breadth.

To our knowledge, the analogies between the two data structures have not been analysed. It could yield interesting results, as analysis in the de Bruijn graph framework is sometimes limited by the choice of the parameter k. Manipulating the affix tree can be compared to analysing all possible de Bruijn graphs simultaneously.

2.3 Implementation

To better underline the connections which can be used to explore a dataset stored in a de Bruijn graph, below is a brief bottom-up description of the implementation of de Bruijn graphs within Velvet.

2.3.1 Graphs

```
struct graph_st {
        IDnum sequenceCount;
        IDnum nodeCount;
        Node **nodes;
        Arc **arcLookupTable;
        ShortReadMarker **nodeReads;
        IDnum *nodeReadCounts;
        Coordinate insertLengths[CATEGORIES + 1];
        double insertLengths_var[CATEGORIES + 1];
        int wordLength;
        GapMarker **gapMarkers;
};
```

Figure 2.2: Definition of the Graph structure

The overall de Bruijn graph can be accessed through a *Graph* structure. It stores general information such as the highest sequence index (*sequenceCount*), the highest node index (*nodeCount*), the hashing length k used (*wordLength*) or the insert length of the various libraries and their respective variances (*insertLengths_var*).

This wrapper is also an access point to various arrays. The graph stores pointers to all of its nodes organised by index. Also, it stores an array of arrays called *nodeReads* of short read markers (see below), grouped by node. The lengths of the latter arrays are kept in *nodeReadCounts* for practical purposes.

Finally, the graph stores pointers to gap markers, whose use will be explained in Chapter 4.

2.3.2 Nodes

```
struct node_st {
        IDnum ID;
        Node *twinNode;
        Arc *arc;
        IDnum arcCount;
        Descriptor *descriptor;
        Coordinate length;
        PassageMarker *marker;
        boolean status;
        boolean uniqueness;
        Coordinate virtualCoverage[CATEGORIES];
        Coordinate originalVirtualCoverage[CATEGORIES];
};
```

Figure 2.3: Definition of the Node structure

The atomic element of this implementation of the de Bruijn graph is the node, identified by its *ID* number. To ensure the symmetry of the bi-graph, each node stores a pointer to its twin node, whose *ID* is the node's *ID* multiplied by -1. For this reason, node *ID* numbers cannot be equal to 0.

The node's descriptor or $s(N)$, corresponds to the final nucleotides of the *k*-mers represented by that node. To save memory, those *k*-mers are discarded. The node's *length* is simply the length of $s(n)$.

To each node is attached a doubly linked list of outgoing arcs. Given the symmetry of the graph, all incoming arcs are twinned with an arc coming out of the twin node. For this reason, only pointers to outgoing arcs are stored.

Some de Bruijn graph based assemblers such as ABySS (Simpson et al., 2009) assume that only 4 arcs can come out of a given node. While this is true for the original graph, this limit is no longer valid after the error correction algorithm, Tour Bus (cf. Chapter 3). For this reason we opted for a more versatile linked list of arcs. The number of elements in that list is stored in the *arcCount* variable for practical purposes.

Reads are associated to nodes in two different manners. Long reads are represented by the *PassageMarker* objects (cf. 2.3.4), in a linked list whose first element is stored within the node. Short reads are represented by *shortReadMarker* ojects (cf. 2.3.5), which are stored in a specific array attached to the graph.

Finally a number of flags are attached to the node. *Uniqueness* and *status* are boolean variables which allow the various algorithms to track down their progression throughout the graph.

2.3.3 Arcs

```
struct arc_st {
        IDnum multiplicity;
        Node *destination;
        Arc *twinArc;
        Arc *next;
        Arc *previous;
        Arc *nextInLookupTable;
};
```

Figure 2.4: Definition of the Arc structure

Whereas much information is stored in the nodes, the arcs are relatively simple, but heavily interconnected by pointers.

Each individual arc is characterised by its *destination* node and its *multiplicity*, that is the number of reads that go through the arc or its reverse complement.

It is firstly connected to its twin arc, which shares the same multiplicity. It is also part of a doubly linked list of arcs (*next* and *previous* pointers) coming out of its source node.

Finally, in the Tour Bus algorithm (cf. Chapter 3), frequent random arc searches made it necessary to speed up this process. Arcs are therefore stored in linked lists (*nextInArcLookupTable* pointer) which are attached to the bins of a simplified hash table.

Figure 2.5 summarises the connections between nodes and arcs. This representation of the graph is non-redundant but non-specific, in that only one arc can connect any two nodes, but there is no information as to which read traverses which arc. This synthetic structure is especially useful to exploit short read datasets with minimal memory consumption. Each arc represents locally the paths of numerous collinear short reads, obviating the need for costly searches.

2.3.4 Long read markers

A passage marker represents the mapping of a substring of the sequence identified by *sequenceID* onto a *node*. To take into account direction, the reverse complement of a sequence is identified by the sequence index multiplied by -1. This is why sequence indices, like node indices, cannot be equal to 0. The passage marker stores the *start* coordinate of the alignment within the read and the finish offset which separates this alignment from the 3′ end of the node. As explained below, to avoid redundancies, the finish coordinate within the read and start offset within the node are not stored.

Within a single node, passage markers are stored in a doubly linked

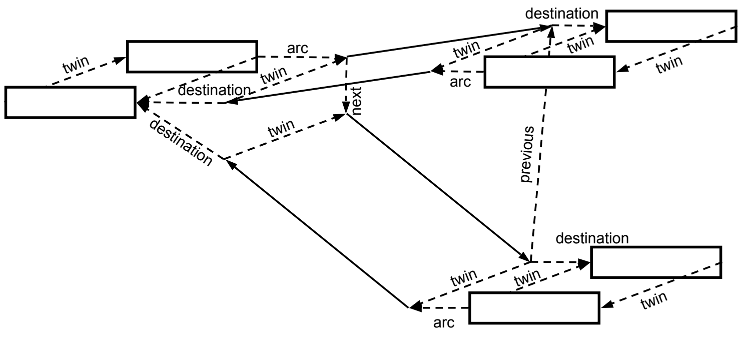


Figure 2.5: Summary of the overlying graph structure

Nodes are represented as rectangles and arcs as unbroken vectors. The dashed arrows represent the pointers connecting all these elements together. Each node is connected to its nearby twin and to a doubly linked list of outgoing arcs.

```
struct passage_st {
        IDnum sequenceID;
        Coordinate start;
        Coordinate finishOffset;
        Node *node;
        PassageMarker *nextInNode;
        PassageMarker *previousInNode;
        PassageMarker *twinMarker;
        PassageMarker *nextInSequence;
        boolean status;
};
```

Figure 2.6: Definition of PassageMarker structure

list (*nextInNode* and *previousInNode* pointers) allowing quick insertion and removal. Within the sequence, passage markers are stored as a linked list (*nextInSequence* pointer).

As most structures in Velvet, each passage marker is linked to a twin marker structure, which corresponds to the reverse complement sequence, mapped onto the twin node. The *start* and *finishOffset* coordinates of a passage marker correspond to its twin's finish and start offset respectively. To obtain the previous marker in sequence of a marker M, it is necessary to query the twin of the next in sequence of the twin of M.

Finally, passage markers carry a boolean *status* marker, to allow algorithms to mark their progression when they visit passage markers.

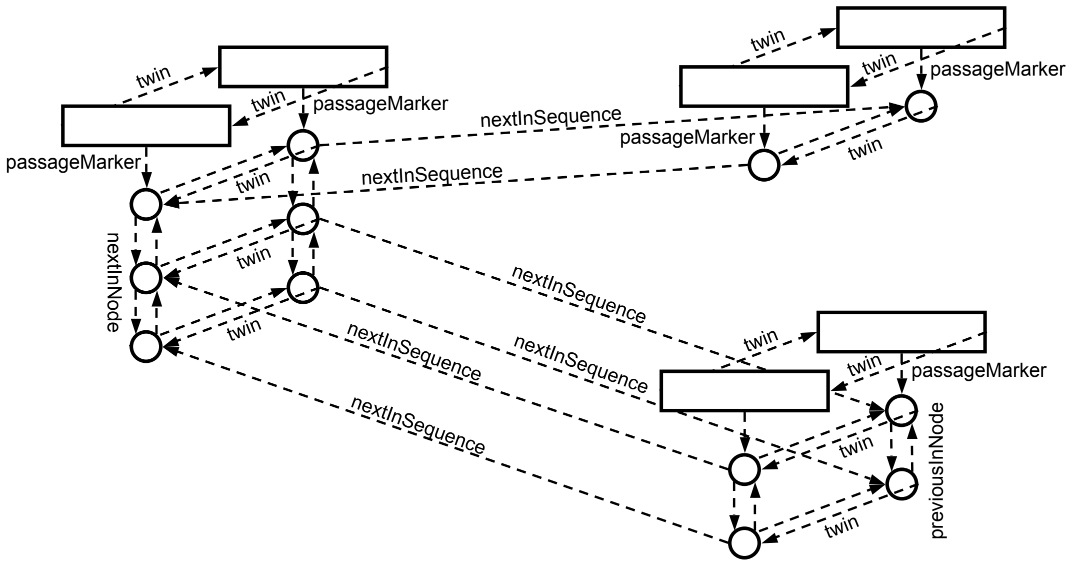


Figure 2.7: Summary of the passage markers and their connections to the graph

Nodes are represented as rectangles and passage markers as circles. Dashed arrows represent pointers. Each passage marker is simultaneously part of an alignment-specific pair, of a node-specific doubly linked list and of a sequence-specific linked list.

Figure 2.7 summarises the network of pointers which connect passage markers to the graph. These connections describe many paths through the graph. They are potentially redundant, but allow processes to follow a specific read throughout its length. This is especially useful when exploiting long reads (cf. Chapter 4). The path of each read can be followed in detail. Passage markers are also used as temporary and flexible markers when merging bubbles in the Tour Bus algorithm (cf. Chapter 3).

2.3.5 Short read markers

```
struct shortReadMarker_st {
        IDnum readID;
        Coordinate position;
        ShortLength offset;
};
```

Figure 2.8: Definition of ShortReadMarker structure

Because they are much more numerous and presumably less informative, short reads are tracked with much lighter datastructures, short read markers. Associated to a node, through the graph's *nodeReads* array of arrays, the short read marker only stores three pieces of information: the *readID* identifier of the sequence it represents, the *position* of that marker within the node and, if the read has been broken, the offset of that read marker from the start of the read.

The position and offset of a short read correspond to the start offset and start coordinate of a passage marker respectively, which is a possible source of confusion. The reason for this is that a short read is expected to be included within a node, whereas a long read is expected to span right across a node. In both cases, the offset is usually equal to 0, whereas the position of the short read within the node and that of the passage marker along the long sequence are much more variable.

Unlike the previous structures, short read information is not duplicated onto the twin node. This is to save memory, as duplication of the structures would not have brought any advantage.

2.4 Construction

2.4.1 Read indexing

Throughout this study, larger and larger datasets have been submitted to Velvet. It was therefore necessary to gradually adapt the hashing algorithm to the workload. Briefly, tested methods include an exhaustive index of possible *k*-mers, a standard hash function and a modified hash function from the Genewise package (Birney et al., 2004). However, as the *k*-mer length and amount of reads increased, each of these implementations had to be abandoned.

Finally, the bucket arrays of the last hash table implementation were replaced by binary search trees. Although more costly in memory, these trees allow for a logarithmic search time (Cormen et al., 2002). To reduce the implementation complexity, splay trees (Sleator and Tarjan, 1985) were chosen to fill this role. Although theoretically less efficient, they are much easier to implement and have proven in practice as efficient as other forms of binary search trees. This solution, although memory costly, proved robust even with very large datasets. The processing of reads was thus greatly accelerated and large datasets (up to 500.000.000 Illumina reads) could be hashed in a matter of hours instead of days.

2.4.2 Construction from overlap information

The reads are first hashed according to a predefined *k*-mer length. This variable k is limited by the length of the reads being hashed, to allow for a small amount of overlap, usually $k = 21$ for 25bp reads. Smaller *k*-mers increase the connectivity of the graph by simultaneously increasing the chance of observing an overlap between two reads because of errors and the number of ambiguous repeats in the graph. There is therefore a balance between sensitivity and specificity determined by k.

For each *k*-mer observed in the set of reads, the hash table records the ID of the first read encountered containing that *k*-mer and the position of its occurrence within that read. Each *k*-mer is recorded simultaneously to its reverse complement sequence. To ensure that each *k*-mer cannot be its own reverse complement, k must be odd. This first scan allows each read to be re-written as a set of new *k*-mers combined to overlaps with previously hashed reads. This new representation of the reads sequence is called a *roadmap*.

A second database is created with the opposite information. It records, for each read, which of its original *k*-mers are overlapped by subsequent reads. The ordered set of original *k*-mers of that read is cut each time an overlap with another read begins or ends. For each uninterrupted sequence of original *k*-mers, a node is created.

Finally, reads are traced through the graph using the roadmaps. Knowing the correspondence between original *k*-mers and the newly created nodes, it is possible to proceed from one node to the next, creating a new directed arc or incrementing the multiplicity of an existing one as appropriate at each step.

2.4.3 Alternative construction from the set of *k*-mers

An alternative construction procedure is possible, as demonstrated in ABySS (Simpson et al., 2009). In this method, all the reverse complementary *k*-mer pairs from the read set are indexed in a hash structure, possibly over a network of computers. Each *k*-mer pair is annotated by eight bits which indicate the existence or absence of the eight possible overlapping *k*-mers. Edges are not stored explicitly and neighbouring nodes are found through searches in the hash. Therefore, the graph is represented in memory in a manner much closer to the theoretical definition of de Bruijn graphs. It can be explored by iterative searches for overlapping *k*-mers.

This approach has the advantage of being parallelisable onto a collection of computers with arbitrarily small memory capacities. Given the cost of large memory machines, this seems to be a rational approach to analysing large datasets.

On a single machine, however, this approach was found to be slower, because of frequent searches. Velvet was initially designed as a proof of principle and it was assumed that it would initially be mainly used for small assemblies, typically on bacterial genomes. For these reasons, the construction through roadmaps was adopted. It is perfectly feasible to use both methods interchangeably and several groups are currently implementing this second construction approach within the Velvet framework (Jeffrey Cook, Zamin Iqbal and Mario Caccamo, pers. comm.).

2.5 Concatenation

After constructing the graph, it is generally possible to simplify it without any loss of information. Blocks are interrupted each time a read starts or ends. This leads to the formation of chains of blocks or linearly connected sub-graphs. This fragmentation of the graph costs memory space and lengthens calculation times.

These chains can be easily simplified. Whenever a node A has only one outgoing arc that points to another node B that has only one incoming arc, the two nodes (and their twins) are merged. Iteratively, chains of blocks are collapsed into single blocks.

The simplification of two nodes into one is analogous to the conventional concatenation of two character strings. This straightforward transformation involves transferring arc, read and sequence information as appropriate.

2.6 Using PreGraphs

All the previous operations only require a fraction of the pointers listed in the structures described earlier. This is why a more memory-efficient set of structures, *PreGraph*, *PreNode* and *PreArc* are in fact used for graph construction, tip clipping (cf. Chapter 3) and concatention.

Once the pre-graph has been built and significantly simplified, it is converted into the full graph structure, which is more expensive in memory but much more convenient to manipulate in the algorithms described in the next chapters.

2.7 Visualising de Bruijn graphs

Designing and debugging methods for genomic de Bruijn graphs require an intuition for the actual topology of the graph. Whereas simple diagrams convey ideas quickly, they are generally oversimplified and were found to be misleading when trying to resolve problems. Unfortunately, because of its scale, the de Bruijn graph is generally unwieldily for human inspection. This is why alternative methods of explorations had to be developed.

2.7.1 Global visualisation

Paul Harrison of Monash University developed a Python script which, combined with GraphViz (Gansner and North, 1999), an open-source graph layout program, produces an image of a de Bruijn graph constructed with Velvet.

Figure 2.9 shows the graph diagram of the *Steptococcus suis* reference genome, hashed into 21bp words. What immediately becomes apparent is the connectedness of the graph. Instead of displaying separable modules, the graph presents a central tangle which it regularly goes back into. This observation led us to two conclusions. Firstly, parallelisation of the graph, although feasible, would require large quantities of cross-node communication. In other words, it is not possible to subdivide the graph into elements which could be transformed independently. Secondly, there are extremely few isolated spurious overlaps, with regards to the massive number of distant repeats or low complexity regions. This effect is even stronger in more complex genomes.

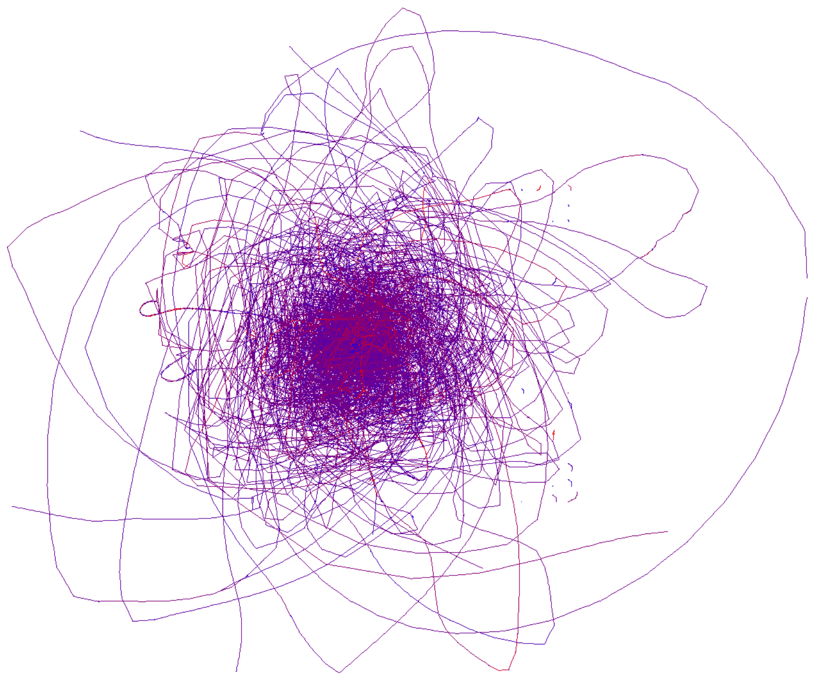


Figure 2.9: Graphical representation of the de Bruijn of the *Streptococcus suis* genome

In this representation, node sequences are represented as curves, which connect at their tips.

2.7.2 Local reference based visualisation

It quickly became clear that troubleshooting some of the algorithms described in the following chapters would require some adequate visualisation techniques. Given that most tests were done on species with a known reference, it was possible to use this sequence as a guide to the graph's complexity.

The first solution consisted in following the path of the reference sequence through the graph, recording the properties of the nodes being traversed. Figure 2.10 represents the length and multiplicity of successive nodes on the reference path.

In this diagram, long contigs are interrupted by two types of breaks. Sometimes, two long contigs are separated by a very short, isolated feature, presumably the random repetition of a short sequence within the reference. However, some interruptions are characterised by long series of very short nodes. Inspection of these seemingly non-random features revealed that they consist mainly of interspersed repeats. Indeed, these repeats are sufficiently conserved and numerous to present many overlaps, but sufficiently diverse to create many breakpoints which could not be smoothed by the Tour Bus algorithm (cf. Chapter 3).

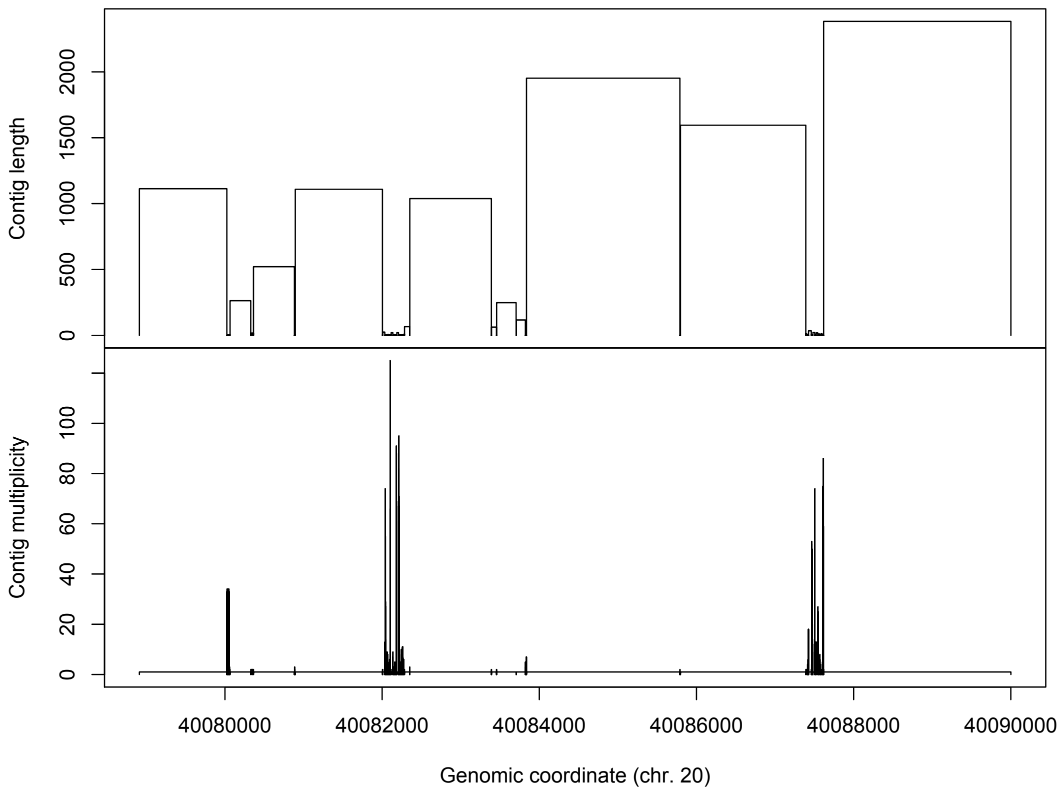


Figure 2.10: Graphical representation of the path of a human reference sequence through the de Bruijn graph

These two plots represent the properties of the successive nodes traversed by a small region of the reference sequence path. The complete graph was built using the 5Mb of human chromosome 20, so the multiplicity of the nodes refers to their multiplicity within that region.

2.8 Colorspace de Bruijn graphs

In collaboration with Craig Cummings and Vrunda Sheth of Applied Biosystems, an alternate version of Velvet compatible with SOLiD's colorspace encoding was developed.

One option which was quickly rejected, was to convert the reads into nucleotide sequences then assemble them with Velvet. However, this would have artificially augmented the error rate. Indeed, a single sequencing error in colorspace completely transforms the downstream sequence when translating into sequence space. This means that a nucleotide in the sequence would only be correctly called if all of the di-nucleotide colours upstream were correctly called.

It is therefore more reliable to assemble the sequences in colorspace and produce colorspace contigs. Converting these contigs to sequence space is less error-prone than converting individual reads, partly because the consensus error rate is much lower than the basic base-calling error rate, but also using the alignment of all the reads onto the contig's consensus sequence, as explained below.

The Velvet code was therefore modified to allow colourspace hashing. The main alteration consisted in changing the rules of opposite strand pairing. In sequence space, opposite strands of DNA have reverse complement sequences, whereas in colorspace, their sequences are simply reversed.

Velvet can then be used within a pipeline developed by Applied Biosystems. Colorspace sequences are first converted to a double-encoded format which can be read by Velvet. Double-encoding consists of replacing the digits used in colorspace files (0, 1, 2 and 3) directly into letters (A,T, G, C). This transformation is performed only to suit Velvet's file parsers which expect nucleotide sequences, but does not correspond to a translation into sequence space. Each letter does not represent a nucleotide, but a di-nucleotide color.

Velvet then loads these files and produces contigs, respecting the modified rule of inversion and producing colorspace contigs. Converting these contigs into sequence space is done by a final program which loads the alignment of the reads onto the consensus sequences. Each read is characterised by a known initial nucleotide, which can be mapped onto the reference. The converter then produces a sequence which respects as many of the colour transitions as possible, while maximising number of coinciding initial nucleotides.

The de Bruijn graph is therefore convenient for various sequence analysis tasks. Through the development of *ad hoc* techniques, the Velvet framework can build a de Bruijn graph from diverse datasets quickly and efficiently. However, the de Bruijn graph built from genomic data can be extremely complex, due to errors and conserved repeats. Specific algorithms to resolve these issues are presented in the next two chapters.

Chapter 3

Removing errors

3.1 Previous methods

In the traditional overlap graph approach, error correction is implicitly contained in the parameters used to validate overlaps between reads. Isolated sequencing errors, too small to cancel the global alignment of two reads, are simply ignored when constructing the overlap graph. These errors could always reappear when calculating the consensus sequence of a contig, but by that stage, the alignment of all the reads onto that contig is sufficient to reliably detect errors.

Because of the exact matching of *k*-mers, the de Bruijn graph is sensitive to variation. Error correction is therefore crucial to the use of those graphs in analysing sequencing data. Moreover, it is important that sequencing methods be robust to minor sequencing variants. Genomic samples are generally diploid, meaning that the assembler must be able to cope with processing two very similar, yet non-identical, sets of sequences. Otherwise, each SNP or small indel causes a disruption in the contigs of the graph, needlessly fragmenting the assembly.

In their original paper, Pevzner et al. (2001) proposed a preprocessing of the reads, based on the statistical representation of *k*-mers. This method was described in more detail in (Chaisson and Pevzner, 2008). The authors start by defining the spectrum G_k for genome G as the set of k-mers in G. In a *de novo* assembly, they approximate G_k by using all the k-mers which appear at least M times, M being a user defined threshold based on the average coverage of k-mers. The task, formulated as a Spectral Alignment Problem, consists of minimally modifying the reads of the dataset so that all their k-mers belong to G_k. This method inspired the error-correction method in Allpaths (Butler et al., 2008).

This approach is sensitive to the value of the cutoff M. When encountering repetitive regions or high error rates, it has been shown to lose its efficiency (Chaisson and Pevzner, 2008). For this reason, a set of methods were developed to both remove errors and simplify small biological variants based on the topology of the de Bruijn graph. Errors are corrected after graph creation, allowing simultaneous operations over the whole set of reads. A coverage threshold is only used at the end of the error-correction procedure.

This analysis of the graph's topology can be compared to previous methods for removing sequence variants from assemblies or alignments. In de Bruijn graphs, every variant creates a distinct feature. Raphael et al. (2004) therefore developed an alternative graph structre created from local pairwise alignments and not from exact k-matches. This allows non-matching nucleotides to be aligned, based on their context. This approach was shown to be useful for genomic alignments. However, its quadratic complexity makes it very costly to apply to highly redundant, very short, next-generation sequencing reads.

Within the traditional overlap-layout-consensus framework, errors generally do not affect the topology of the graph, because of the use of global alignments between long reads. This way, any minor difference can be detected, assessed and possibly ignored. However, Fasulo et al. (2002) noted that long polymorphisms could still induce structures which they called *bubbles* in the overlap graph. They defined as bubble a set of distinct paths in the directed acyclic overlap graph which emanate from the same node and end at the same node. The authors developed a method to detect polymorphisms in an assembly, smoothing them out to allow the assembly of longer contigs. Nodes are ordered greedily by a depth-first search, thus obtaining an acyclic graph by implicitly discarding arcs. Based on that ordering, the algorithm searches for pairs of potential bubble start and termination nodes that could be the end points of a bubble. Finally, the simplification of the bubbles is a straightforward re-alignment of the reads.

The method described below is conceptually similar to that found in (Fasulo et al., 2002), albeit applied to a different graph structure. However, it is based on a more general definition of bubbles and has a distinct detection and correction method.

3.2 Characterising errors

The error correction steps described in this thesis rely on topological features of the graph to detect errors reliably. Erroneous data can create three types of structures, as represented in Figure 3.1. The first type, *tips*, are generally due to errors at the ends of reads. The second, *bulges*, can either be created by errors within long reads or by nearby tips presenting spurious overlaps. Finally, the third structure, *chimeric connections*, can be the consequence of cloning errors or of a tip sharing an overlap with a random node in the graph. These three features are removed consecutively in this pipeline.

Tips form when an error occurs less than k bp from the start or end of a read. The sequence path of the read therefore follows the canonical path, before deviating into a path of its own. Given the range of micro-read lengths relative to the usual k-mer length, tips are extremely frequent when building a graph from next-generation sequencing data. Moreover, because of signal loss, next-generation sequencing reads tend to have a much higher error rate near their 3′ end.

Bubbles can arise due to several phenomena. They can be caused by a random overlap of two nearby tips. This is especially common in high coverage datasets, where the density of tips is proportional to that of reads. Bubbles can also form as a consequence of an error in the middle of a long read, which first deviates from the canonical sequence of k-mers, then re-inserts itself into it. Finally, a bubble can also be created by a polymorphism in the dataset. Instead of a single k-mer chain there are two, which mostly follow the same path through the graph, occasionally branching out and merging back together.

Chimeric connections are erroneous features which cannot be classified as either tips or bubbles. They generally connect genuine contigs in artificial ways, often breaking up contigs which otherwise would continue without branching. Such connections can occur for various reasons, such as a random overlap of two tips from different parts of the graph, a read erroneously matched to another part of the genome or an actual chimeric read in which two different parts of the genome are physically ligated.

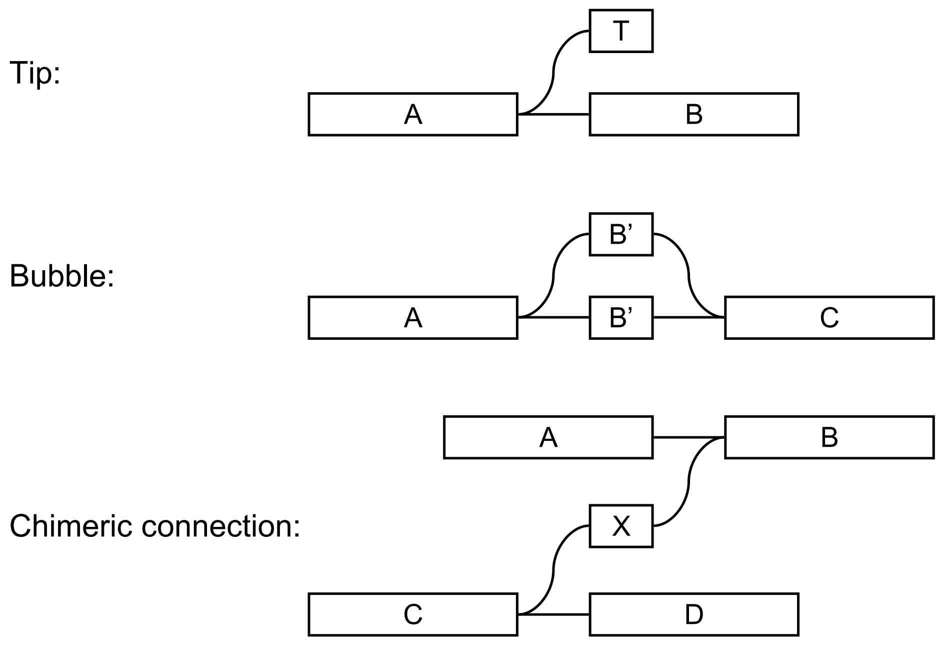


Figure 3.1: Schematic diagram of the three categories of errors in the Velvet framework

Each of these diagrams represents a type of error as classfied in this framework. In the tip, node T is considered an erroneous tip if it is shorter than $2k$ bp and if the multiplicity of the arc from A to T is lower than that of the arc from A to B. In the bubble, node B' is considered to be a minor version of B if its *time* (see below) is longer than that of B and if the two sequences are similar enough. Finally, in the third diagram, node X is flagged as a chimeric connection if its coverage is lower than a threshold. In all three cases, the removal or remapping of the erroneous node allows neighbouring nodes to be concatenated, thus simplifying the graph.

3.3 Tips

As discussed above, a tip is a chain of nodes which is disconnected on one end. The information contained in these tips is simply discarded. This results in only local effects, as no connectivity is disrupted. Nonetheless, restrictions must be applied to the process to avoid eroding genuine sequences which are merely interrupted by a coverage gap. To deal with this issue, we define two criteria: length and minority count.

A tip is be removed if it is shorter than $2k$ bp. This arbitrary cutoff length was chosen because it is greater than the length in k-mers of an individual very short read. Presumably, erroneous constructs involving entire short reads are extremely rare. In the case of long reads, this cutoff is set to be the maximum length of a tip that can be created by two nearby mistakes. A tip longer than $2k$ bp therefore represents either a genuine sequence or an accumulation of errors that is hard to distinguish from novel sequence.

A tip is considered to have minority count if, at the node where it is connected to the graph, there is at least one arc going back into the graph which has a multiplicity higher than that of the arc going into the tip, as represented in Figure 3.1. In other words, starting from that node, going through the tip is an alternative to a more common path.

Imposing the constraint of minority count ensures that, at the local level, tips are removed in an increasing order of multiplicity. The process progressively uncovers chains of high coverage nodes that are not destroyed by virtue of the previous criteria, thus preserving the graph from complete erosion.

Under these two criteria, tips are iteratively removed from the graph. When there are no more tips to remove, the graph is concatenated once again (cf. 2.5).

3.4 Bubbles and the Tour Bus algorithm

3.4.1 Definition

Two paths are considered redundant if they start and end at the same nodes (forming a bubble, see Figure 3.1) and contain similar sequences. Redundant paths can be created by errors or biological variants, such as SNPs or cloning artefacts prior to sequencing. To address the issue of erroneous bubbles, we developed an algorithm called Tour Bus. The criteria for deciding whether two paths justify simplification can be complex, taking into account error models of the sequence or (for the case of mixed haplotype samples) other features of the sequence and graph, such as coverage. In the tests that follow, simple sequence identity and length thresholds are applied.

This configuration means that bubbles created by SNPs or small indels are likely to be smoothed out. Although this removes significant information from the graph topology, it is not altogether lost. The thorough re-mapping of all the information of one branch of the bubble onto the other means that detecting these genuine variants can be done reliably by comparison of the sequences to the consensus.

At the same time, the overall assembly benefits from having these small variants removed. Instead of having contigs regularly broken up by small isolated differences, this allows the system to detect much larger homologous regions. Arguably, it is easier to analyse a long contig with small variants mapped onto it than an exhaustive graph structure.

3.4.2 Detection of bubbles

The Tour Bus algorithm is based on a Dijkstra-like breadth-first search (Dijkstra, 1959) to detect redundant paths. The algorithm starts from an arbitrary node and progresses along the graph, visiting nodes in order of increasing distance from the origin. In this application, the distance between two consecutive nodes A and B is the length of $s(B)$ (cf. 2.1) divided by the number of reads connecting A to B. This *ad hoc* metric, called *time*, gives priority to more reliable, higher coverage paths (which are “faster”).

$$time\,(A \rightarrow B) = \frac{length(B)}{multiplicity(arc(A \rightarrow B))} \tag{3.1}$$

Whenever the process encounters a previously visited node, it backtracks from both the current node and the previously visited node to find their closest common ancestor. From the two retraced paths, the sequences are extracted and aligned. If judged similar enough, the two paths are merged. The “faster” path, which reached the end node first in the search, is used as a consensus or canonical, path because of its higher coverage. The *time* metric implicitly imposes a majority vote in choosing the consensus sequence. The slower path, whether an error or a variant, is the minority path. Figure 3.2 shows how the iteration proceeds on a small example graph.

It would be possible to integrate base-calling confidence scores in this *time* metric. Phred quality scores (Ewing et al., 1998) could weight the coverage of the different arcs. When using a sufficient coverage to obtain long contigs (usually $\geq 10X$) and a normal error rate ($\leq 1\%$) majority count was found to be sufficiently reliable.

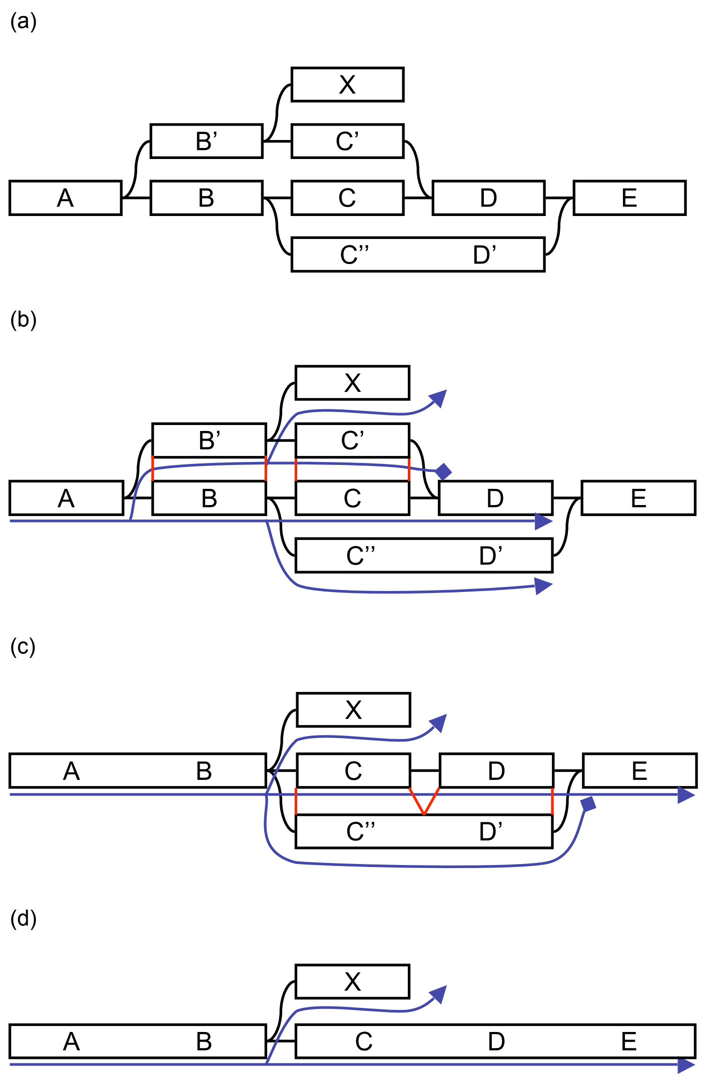


Figure 3.2: Schematic diagram of an iteration of the Tour Bus algorithm

The search starts from A and spreads towards the right along the blue arrows, as shown in the second step. The progression of the top path (through B' and C') is stopped because D was previously visited. The nucleotide sequences corresponding to the alternate paths $B'C'$ and BC are extracted from the graph and aligned. In step (c), the two paths are judged similar, so the longer one, $B'C'$, is merged into the shorter one, BC. The merging is directed by the alignment of the consensus sequences, indicated in red lines in step (b). Node X, which was connected to node B', is now connected to node B. The search progresses and the bottom path (through C' and D') arrives second in E. Once again, the corresponding paths, $C''D'$ and CD are compared. Finally, CD and $C''D'$ are judged similar enough and the longer path is merged into the shorter one.

3.4.3 Path merging

The positioning of the different elements is based on the sequence alignment of the paths. In other words, the nodes are matched according to the alignment of their sequences. If two nodes have sequences which were perfectly matched in the alignment, then they are merged. However, if the sequences were not aligned along their entire length, then it is necessary to break up the nodes and merge the corresponding fragments.

Merging two paths is a complex operation, as all the underlying graph structures must be remapped while maintaining their connections with other nodes. This means that all reads must be able to follow their sequence paths after the transformation, even if they do not follow the bubble along its entire length and the corresponding arc connections must also be preserved. Although straightforward on linear paths, i.e. when no block is visited more than once, this transformation is more complex in the presence of palindromes. Palindromes create hairpin folds, paths that go through a block one way then go through it again in the opposite direction. The need to preserve connectivity prevents projecting hairpins onto linear paths.

To merge two paths, Tour Bus creates a chain of markers along both of them, node by node. The paths are merged progressively from one end to the next. At each step, the first unmapped minority node is compared to the corresponding majority consensus node, using the local sequence alignment produced previously. All the information attached to that node, including coverage, sequence identifiers and arcs is then mapped accordingly onto the majority node. The presence of markers allows Tour Bus to dynamically modify the marked path as it corrects the graph. This can be especially useful when a path goes through node A, then later through its twin node $\tilde{A}$. After remapping A, the Tour Bus algorithm remaps $\tilde{A}$ and diverts the path markers accordingly.

Given that both paths start at the same node, at each step, the merging algorithm can only be confronted with one of the configurations described in Figure 3.3. Each of these situations is then resolved with a specific transformation. The Tour Bus algorithm transforms the graph so that both paths are parallel over the next few nodes, then goes forward to the new breakpoint between the paths.

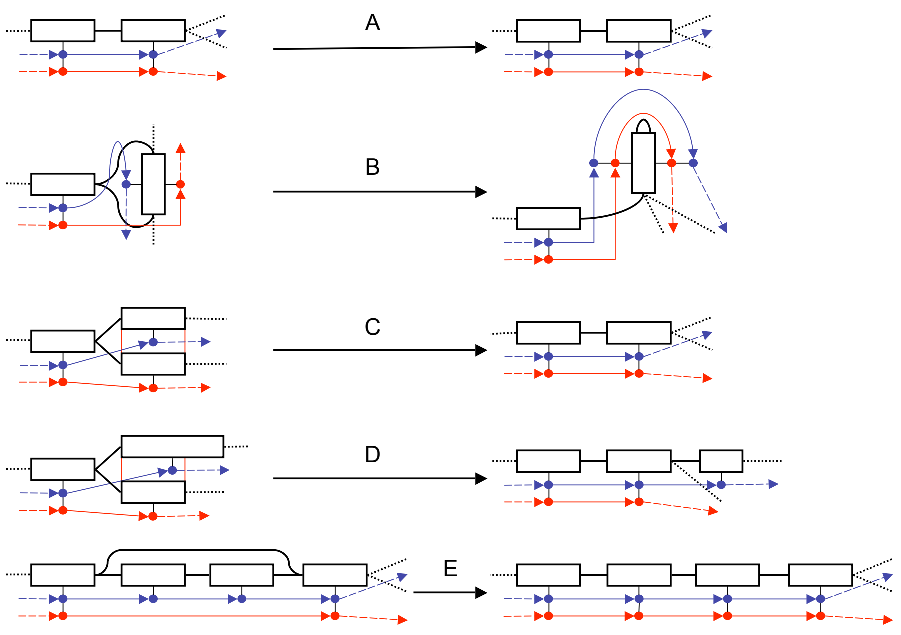


Figure 3.3: Schematic diagram of the basic Tour Bus transformations

These five different transformations are characterised by the initial topology, described on the left of the diagram. The slow and fast paths are depicted interchangeably by chains of blue or red markers. If applicable, the alignment of the nodes is indicated by parallel red lines. The nodes are then modified so that the slow and fast paths are collinear.

The merging algorithm is simply a repetition of the above transformations, depending on the topology of the paths, as described in the pseudo-code in Figure 3.4.

```
slowMarker = first marker on slow path
fastMarker = first marker in fast path

while (slowMarker in the slow path && fastMarker in the
            fast path) {
         slowNode = getNode(slowMarker)
         fastNode = getNode(fastMarker)
         fastLength = coordinate within the aligmment of the
                       end of the fast node
         slowLength = coordinate within the aligmment of the
                         end of the slow node

         if (slowNode == fastNode) {
                 slowMarker = getNextInSequence(slowMarker)
                 fastMarker = getNextInSequence(fastMarker)
         } else if (slowNode == getTwinNode(fastNode)) {
                 slowMarker = getNextInSequence(slowMarker)
                 fastMarker = getNextInSequence(fastMarker)
                 fold the common node into a hairpin
         } else if (slow path leads to fast node) {
                 remap the corresponding slow nodes onto the
                 arc leading to the fast node
         } else if (fast path leads to slow node) {
                 remap the corresponding fast nodes onto the
                 arc leading to the slow node
         } else if (slowLength == fastLength) {
                 remap slow node onto fast
                 slowMarker = getNextInSequence(slowMarker)
                 fastMarker = getNextInSequence(fastMarker)
         } else if (slowLength < fastLength) {
                 remap back of fast node onto the slow node
                 slowMarker = getNextInSequence(slowMarker)
         } else {
                 remap back of slow node onto the fast node
                 fastMarker = getNextInSequence(fastMarker)
         }
}

Concatenate nodes along the merged path when possible
Destroy guiding sequence paths
```

Figure 3.4: Path merging pseudo-code

3.4.4 Practical implementation issues

A consequence of the greedy path merging method is the emergence of nodes of length zero. These nodes appear when a node of the corrected branch cannot be re-mapped onto any node of the canonical branch because it represents an insertion (e.g. Figure 3.3, D). Because of the need to preserve connections and read path integrity, these nodes must be included into the canonical path, removing their sequence. These nodes contain no *k*-mer, yet behave exactly like ordinary nodes. They have arc connections to other nodes and are traversed by read paths.

These nodes of length zero are singular because irrespective of coverage their *time* is zero. This means that they are often part of the majority path and end up having many connections mapped onto them. Experience has shown that these nodes can behave as attractors, pulling increasing amounts of sequence markers. Correspondingly, they have more arcs than most other nodes in the graph and become "hubs".

An interesting behaviour displayed by the algorithm in early implementation was an explosion in memory consumption due to the multiplication of read markers. Given that all the basic transformations of Tour Bus can only decrease the number of nodes, this meant that some nodes were accumulating large numbers of markers. Examination of selected cases revealed that, using the rules described in Figure 3.3, clusters of length zero nodes showed a simplistic self-replicating behaviour. In short, a chain of highly connected nodes of length zero would be repeatedly folded (following rule C) and extended (according to rule D). At each step, the read paths would be correspondingly folded, thus doubling repeatedly the number of markers. This behaviour was finally eliminated by preventing Tour Bus from visiting the same node more than once.

While re-mapping one node onto another, Tour Bus is required to perform many searches for arcs, based on the origin and destination nodes within each node's unsorted list of arcs. To speed up the search the arcs are connected by simple linked lists to a basic hash table. The keys of this table are simply the index numbers of the start and finish nodes of the arcs.

This reorganisation of the data is especially useful when dealing with hub nodes, when many arcs must be located.

3.5 Chimeric features

Erroneous connections are removed after bubble smoothing. These unwanted connections do not create any recognizable loop or structure, so they cannot be readily identified from the topology of the graph as with tips and bubbles (cf. Figure 3.1). Also, they cannot be associated directly to a corresponding correct path. Therefore, they are removed with a basic coverage cutoff. Currently this cutoff is set manually, based on plots of node coverage after the removal of bubbles.

It is important to stress that this simple node removal, because it is done after Tour Bus, does not contradict the cautious approach in the design of that algorithm. The purpose of Tour Bus is to remove errors without destroying unique regions with low coverage. Once it has run, most unique regions are simplified into long straight nodes, where, by virtue of the law of large numbers, the average coverage is close to the expected value. Genuine short nodes which cannot be simplified correspond to low complexity sequences which are generally present multiple times in the genome. Their overall coverage is therefore proportionally higher. Presumably, most low coverage nodes left after Tour Bus are chimeric connections.

Nonetheless, the coverage cutoff is a sensitive parameter. If it is too high, this stage of error correction is liable to create coverage gaps in the assembly. If coverage gaps occur near junctions, they may create misassemblies, by causing the erroneous concatenation of contigs. One method to set the cutoff is based on the observation of the the coverage distribution of the contigs. Generally, two peaks are visible, one at very low coverage corresponding to stochastic errors and a Poisson-shaped peak associated to genuine sequences, as illustrated in B.5. The cutoff is then set so as to separate the two peaks.

3.6 Testing error removal on simulated data

Reads were simulated from four different reference genomes: *Saccharomyces cerevisiae*, *Caenorhabditis elegans*, *Homo sapiens*, and *Aedes aegypti*. These four genomes present different levels of repeat complexity. Their tandem repeat contents are respectively 0.3%, 6.3%, 7%, and 6% (Benson, 1999; The *C. elegans* Sequencing Consortium, 1998; Levy et al., 2007; Nene et al., 2007), and their interspersed repeat contents are respectively 3.1%, 12%, 44.83% and 62% (Kidwell, 2005; The *C. elegans* Sequencing Consortium, 1998; The International Human Genome Sequencing Consortium, 2001; Nene et al., 2007).

5Mb regions of each genome were chosen, corresponding to the approximate amount of DNA which can be sequenced with a 50X coverage depth by a single Illumina lane. 5Mb is therefore the largest amount of continuous data that could be present on current machine formats in a single lane. 35bp long reads were randomly generated at different coverage values, from 5X to 50X, then hashed by 21-mer words. Only substitution errors were considered as these are reported to be the most common class of error for Illumina reads (Dohm et al., 2008). The evolution of the N50 or median length-weighted contig length, against coverage is displayed in Figure 3.5.

In the first test, the reads do not contain errors. Initially coverage increases exponentially, as predicted by the Lander-Waterman statistic (Eric S. Lander, 1988). Then, when coverage is sufficient, the N50 abruptly stops increasing, as it is limited by the natural repetition of the reference genome. This barrier has a different level depending on the reference genome. The more repetitive and complex the genome, the lower the maximum N50.

The second test is identical to the first, but introduces a 1% substitution error rate. The results (cf. Figure 3.5) are consistent with the first test, except that the maximum N50 is lower than with error-free reads. In fact, as coverage rises to 50X, the N50 decreases slightly, due to the adjunction of errors without the closing of any coverage gap.

Finally, the third test is identical to the second, but with reads (with 1% error) generated from two copies of the reference genome: the original one and one with SNPs randomly added at a rate of 1/500bp. The N50 contig lengths obtained in this test are comparable to the previous one (cf. Figure 3.5).

Another factor of difficulty are low-complexity repeats which increase the probability of spurious repeats in a genome. To evaluate this effect, identical tests were run on four bacterial and protozoan genomes with various A+T/G+C imbalances. The genomes selected were *Escherichia coli*, *Mycobacterium tuberculosis*, *Bordetella pertussis* and *Plasmodium falciparum*. These genomes are all close to 5Mb in size, except for *P. falciparum* from which only a 5Mb region was used. Their G+C content are respectively 50.8%, 65.6%, 67.72%, and 19.4% (Blattner et al., 1997; Cole et al., 1998; Parkhill et al., 2003; Gardner et al., 2002).

As can be seen on Figure 3.6, the runs on these genomes show the degradation of the N50 score as the imbalance between (A+T) and (G+C) content increases. Low sequence complexity can clearly have significant consequences on the quality of an assembly and poses a limitation for local k-mer scale analysis. A natural solution would be to increase the hash length, but this approach is limited on the other hand by the coverage depth and quality of the dataset.

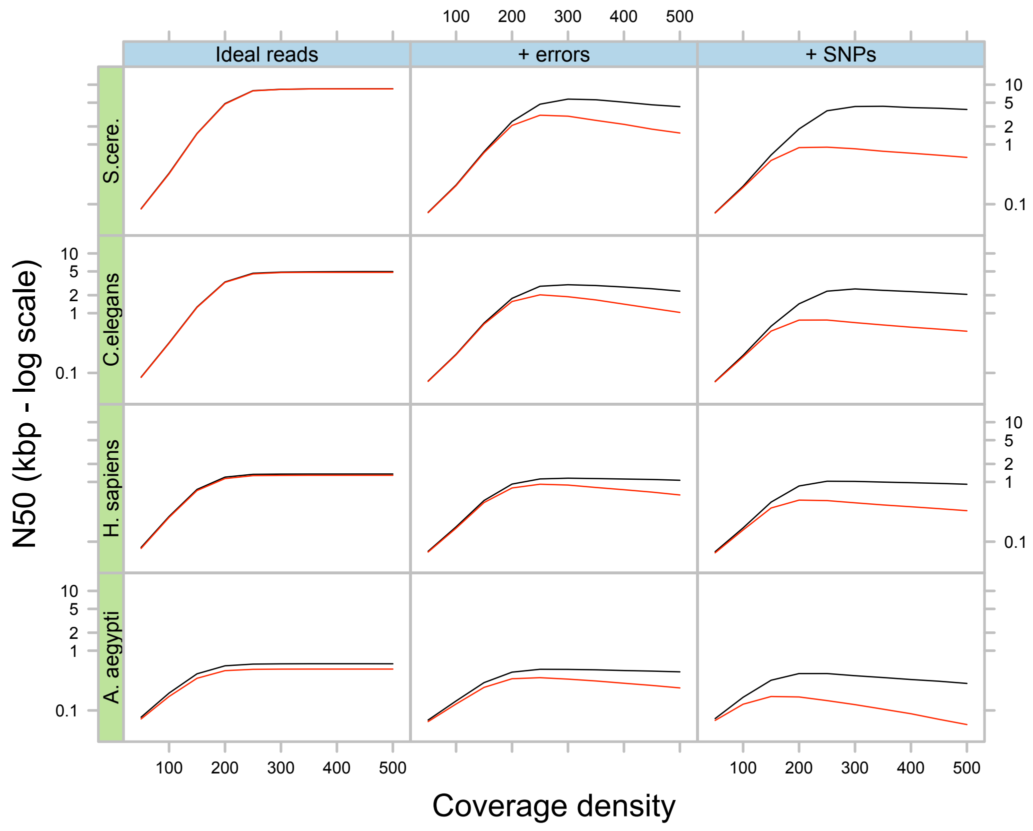


Figure 3.5: Results of simulations using various coverages
5Mb samples of DNA from 4 species (respectively *S. cerevisiae*, *C. elegans*, *H. sapiens* and *A. aegypti*) were used to generate 35bp read sets of varying read depths (x-axis of each plot). The contig length N50 (y-axis, log scale) was measured after tip-clipping (black curve) then after the subsequent bubble smoothing (red curve). In the first column are the results for perfect, error-free reads. In the second column, subsitution errors were inserted in the reads at a rate of 1%. In the third column, a slightly variant genome was generated from the original by inserting random SNPs at a rate of 1 in 500. The reads were then generated with errors from both variants, thus simulating a diploid assembly.

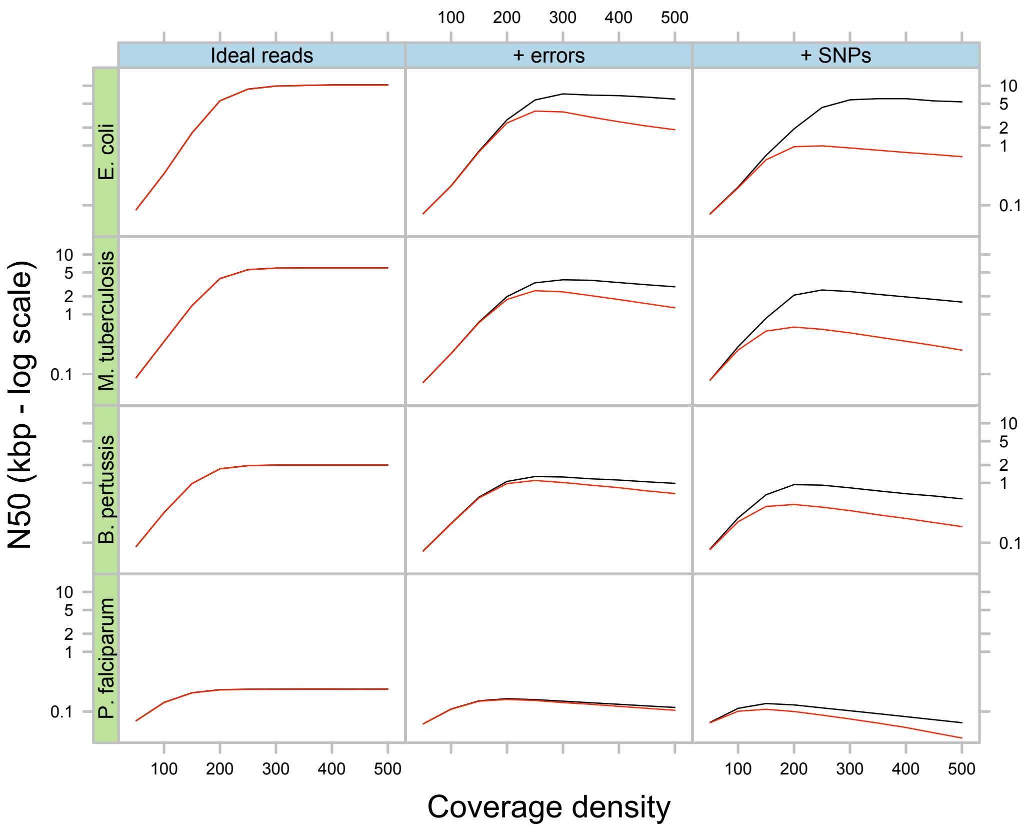


Figure 3.6: Results of simulations on various genomes of varying G+C content
The genome of *E. coli*, *M. tuberculosis* and *B. pertussis* and a 5Mb sample of DNA from *Plasmodium falciparum* were used to run the same tests as on figure 3.5.

3.7 Testing error removal on real data

A 173,428 bp human BAC was sequenced using Solexa sequencing machines, with an average coverage of 970X. The BAC was also sequenced to finished quality using conventional methods. The reads were 35bp long and the ensuing analysis was done with 31-mers.

Errors were removed using the above error correction algorithm (Table 3.1). Because of repeats, the Tour Bus method merged similar paths in the finished sequence, smoothing out similar (but not identical) repeats. This could potentially be avoided with more sophisticated detection of repeat differences, for example based on expected coverage or paired-end connections.

Step	No. of nodes	N50 (bp)	Max. length (bp)	Coverage (%> 50bp)	Coverage (%> 100bp)
Initial	1,353,791	5	7	0	0
Simplified	945,377	5	80	4.3	0.2
Tips clipped	4,898	714	5,037	93.5	78.7
Tour Bus	1,147	1,784	7,038	93.4	90.1
Cov. cutoff	685	1,958	7,038	92.0	90.0
Ideal	620	2,130	9,045	93.7	91.9

Table 3.1: Error removal on experimental human data
The properties of the de Bruijn graph were measured at different stages of the Velvet pipeline: immediately after graph construction, after concatenation of the nodes, after tip clipping, after bubble smoothing and after the removal of chimeric connections. The properties of the de Bruijn graph allow a direct comparison with the graph obtained from the reference BAC sequence.

To test the performance of Velvet against a virtual ideal assembler, we built a de Bruijn graph from the known finished sequence of the BAC. This is equivalent to an error-free, gap-free assembly. Not only did the Tour Bus method significantly increase the sensitivity and specificity of the correction, but it also preserved the integrity of the graph structure. Indeed, the median and maximum node lengths in the short read graph are comparable to those in the finished BAC graph (Table 3.1). The N50 for all nodes of the graph was 1958bp, for nodes $> 100bp$, 2041bp and for nodes $> 1000bp$, 3266bp. Direct sequence alignment showed that nodes of length 100bp or more from the short read graph covered 90.0% of the BAC with 99.989% sequence identity and no mis-assembly. This is similar to the ideal graph, in which nodes longer than 100bp represent 91.9% of the BAC. Only one indel (2bp deletion) was observed.

2.7 million 36bp reads were sequenced from the 2Mb genome of Streptococcus suis P1/7, with a mean coverage depth of 48X and assembled, as described in Table 3.2. Again, no mis-assembly was created, while 96.5% of the genome was covered with 99.996% identity. In the ideal graph, 96.8% of the genome was covered by contigs longer than 100bp. The N50 for all nodes of the graph was 8564bp, for nodes > 100bp, 8742bp and for nodes > 1000bp, 8871bp. Excluding contigs which mapped onto multiple copies of a repeat, only 6 indels, up to 2bp long, were observed.

Step	No. of nodes	N50 (bp)	Max. length (bp)	Coverage (%> 50bp)	Coverage (%> 100bp)
Initial	3,621,167	16	16	0	0
Simplified	2,222,845	16	44	0.1	0
Tips clipped	15,267	2,195	7,949	96.2	95.4
Tour Bus	3,303	4,334	17,811	96.8	96.4
Cov. cutoff	1,496	8,564	29,856	96.9	96.5
Ideal	1,305	9,609	29,856	97.0	96.8

Table 3.2: Error removal on experimental *S. suis* data
The properties of the de Bruijn graph were measured at different stages of the Velvet pipeline: immediately after graph construction, after concatenation of the nodes, after tip clipping, after bubble smoothing and after the removal of chimeric connections. The properties of the de Bruijn graph allow a direct comparison with the graph obtained from the reference *S. suis* sequence.

In conclusion, this error removal pipeline has been shown to correct erroneous data with high efficiency, on simulated dataset as well as experimental data. The removal of errors allows a significant simplification of the graph and an extension of the contigs. However, the assembly is fragmented by the inherent complexity of the repeat structure of the genome sequence, which creates tangles within the de Bruijn graph. The following chapter will discuss methods to assemble contigs beyond repeat breakpoints.

Chapter 4

Resolving repeats

Supposing sufficient read coverage, the first stage of error removal described in the previous chapter generally ends with a set of contigs that are broken up at branching points, whichever the assembly technique chosen. These are due to repeats that create ambiguity. This is especially true in de Bruijn graphs, where repeat length need only be longer than the hash length in order to create a overlap. This chapter, describes methods for resolving repeats within the de Bruijn graph framework. Two types of information are used: the complete sequences of long reads and paired-end reads.

4.1 The repeat resolution problem

4.1.1 Description

A repeated node is a node whose sequence is present several times in the genome. As explained in Chapter 2, any repeated *k*-mer creates a valid overlap. If a repeat is longer, then it is a path through the graph, through which the genome's sequence path goes through several times. In the simplest of scenarios, a repeat is represented as a simple crossing point between two paths, as shown on Figure 4.1

However, experience has shown that actual repeat sub-graphs are

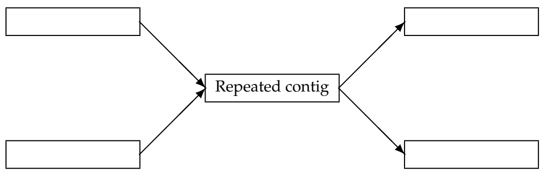


Figure 4.1: Schematic diagram of the simplest possible repeat structure

much more complex as shown in Figure 2.9, sometime because of internal sub-repeats, but especially because of variation between the copies themselves, thus creating breakpoints within the repeat. As an example, the de Bruijn graph of 8 copies of the ALU repeat contains 24 nodes. Also, biological repeats generally have more than two copies, so they have a corresponding number of incoming and outgoing arcs. Repeats should therefore not be considered as isolated junction nodes, but as connected sub-graphs, often referred to as *tangles*.

The objective of repeat resolution is to determine the genome's sequence path through the graph as it goes through repeated nodes. Obviously, the difficulty lies in finding the correct outgoing path, based on the sequence of previously visited nodes.

4.1.2 Formalisation

Pevzner et al. (2001) considered this issue and formalised it as the Eulerian Superpath Problem: finding the shortest path through the graph which is a super-path of the known read paths, in other words that contains all of the read paths as subpaths. However, the complexity of this problem was proven to be NP-hard by Medvedev et al. (2007).

4.1.3 Previous approaches

Nonetheless, Pevzner *et al* did provide some heuristics to start resolving this problem using graph transformations that reduce the number of nodes in the graph without affecting the space of possible superpaths: detachements and cuts.

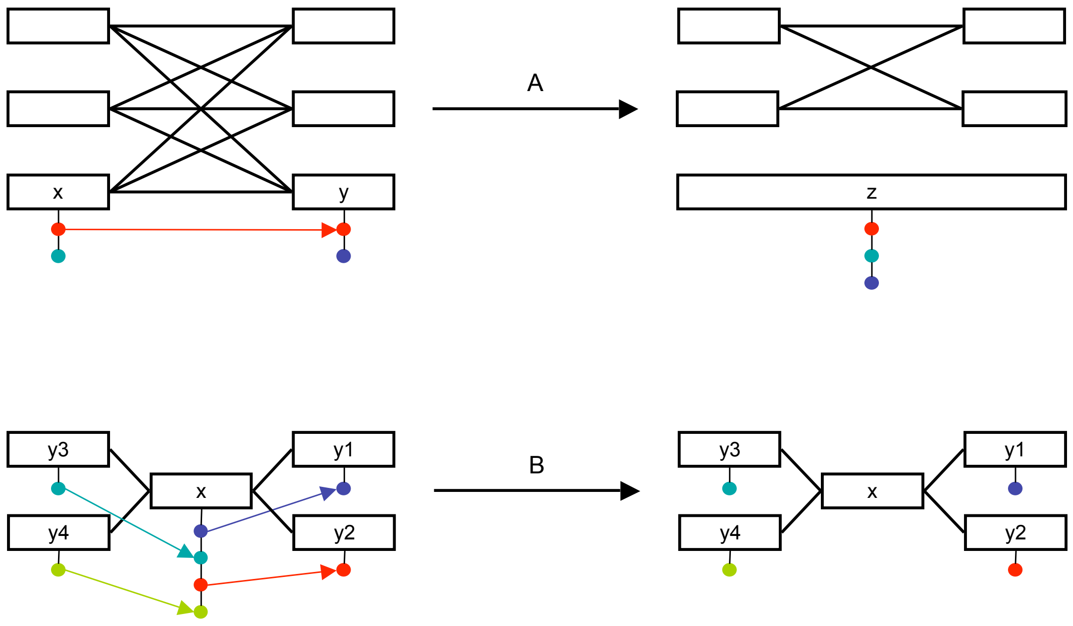


Figure 4.2: Schematic diagram of $x - y$ detachments and x cuts (adapted from (Pevzner et al., 2001))

In (A) all the reads which extend contig x go into contig y. Contigs x and y are therefore merged into z, which is *detached* from the previous tangle. In (B) none of the reads going through x connect one of the left hand contigs y_3 or y_4 to either of the right hand contigs, y_1 or y_2. These reads are therefore *cut* as they enter x.

Using the Velvet representation of de Bruijn graphs, *x-y detachments* consist of simplifying tangles of the graph where all the reads arriving by node x either come to an end or exit by a single node y and vice versa, as shown on diagram 4.2. Both nodes are then merged, avoiding the alternative connections. *Cuts* correspond to simplifications of the sequence paths where detachments cannot be operated.

However, as discussed above, repeats can actually comprise many nodes. Therefore, one may ask whether all repeats can be resolved by iterations of cuts and detachments. Is it possible to extend detachments to more complex tangles? The Rock Band algorithm described below attempts to answer this question.

Paired-end reads have been widely considered as a promising solution. This approach has been studied in projects such as ARACHNE (Batzoglou et al., 2002) and BAMBUS (Pop et al., 2004). In all of these examples paired-end information was used to order and orient or scaffold, contigs and test the validity of these contig assemblies.

In EULER-DB (Pevzner and Tang, 2001) the idea of extracting the sequence between contigs was presented, even though it belonged to collapsed repeated regions. Read-pairs were tested separately to check if they could constrain the scaffolding problem by defining a unique path between two contigs. ALLPATHS (Butler et al., 2008) extended this idea by bundling all the read-pairs connecting two contigs to reduce calculations. The SHORTY algorithm (Chen and Skiena, 2007) additionally used sets of mate-pairs which were all anchored on one end to a unique *k*-mer. This allowed the algorithm to obtain ordering information at a scale far finer than the insert length itself.

A simple scaffolding algorithm, called Breadcrumb (Zerbino and Birney, 2008), was developed. It was inspired by SHORTY, but used long contigs instead of *k*-mers to anchor groups of mate-pairs. Breadcrumb could resolve simpler repeats but was quickly limited in the case of mammalian genomes. The algorithm was then significantly improved and renamed Pebble. It now exploits the knowledge of insert lengths to resolve more complex situations. This allows the algorithm to localise mate-pairs much more efficiently using only short unique contigs.

4.2 The Rock Band algorithm

4.2.1 Identifying unique nodes

In order to resolve repeats with confidence it is necessary to determine which contigs in the assembly are unique. Various methods based on topology have been designed (Pevzner et al., 2001; Medvedev and Brudno, 2008), but in its current implementation, Pebble only relies on contig coverage values, in a similar way to the A-statistic presented in (Myers, 2005). This formula was adapted to take advantage of the high density of read start positions offered by next-generation sequencing data.

X_i denotes the number of read starts on position i. It is assumed that the X_i are independent random variables with a Poisson distribution of parameter equal to the expected density ρ. The value of ρ is determined experimentally using the distribution of node coverages. It is also assumed that the multiplicity within each contig's k-mers is constant, because of the properties of the de Bruijn graph. In other words, a node is either unique or repeated r times, but it should not be a mixture of unique and repeated k-mers.

According to the Central Limit Theorem, the mean of all the X_i inside a node of length n, follows a distribution which converges towards a normal distribution of mean ρ and standard deviation $\sqrt{\rho/n}$. A node is flagged as unique based on the log-odds ratio of the likelihood of the average coverage $\bar{X}$ for the node being unique over that if it is represented twice in the genome. Given an observed mean density of $\bar{X}$, this ratio is:

$$F(\bar{X}, n, \rho) = \frac{log2}{2} + n\frac{\rho^2 - \frac{\bar{X}^2}{2}}{2\rho} \tag{4.1}$$

To ensure specificity, a uniqueness cutoff of $F \geq 5$ is imposed. For practical purposes, because next-generation sequencing technologies generally produce reads of very similar length, the coverage of a node is used as a proxy for the number of reads. Even with this simplification, this test has proven to be sufficiently reliable.

4.2.2 Iterative joining of contigs

The advantage of the de Bruijn graph over the overlap graph is to allow the mixture of read-lengths. If long reads are available, they can be easily used to connect the nodes of the graph after error correction, using a simple procedure called Rock Band. The general idea is that if all the long reads which go out of one unique node go consistently to another unique node and vice-versa, then both nodes can be confidently merged.

Examining every unique node in an arbitrary order, Rock Band iteratively tries to extend that node to another one. It enumerates the long reads which go out of that node and follows them to the next unique node. If all the long reads go to the same destination, then Rock Band merges the two unique nodes, using the long read sequences to fill the gap between the two. At the next iteration, some of the reads used previously may come to an end. However, merging the nodes involves also merging the sets of reads, possibly incorporating new reads which allows the process to continue.

In order to reduce calculations, long reads are clipped at their ends so that all their tips are mapped onto unique nodes. This clipping can potentially completely erase long reads which are apparently included in repeated regions. Nonetheless, this was found to be quite rare even in eukaryotic sequences. In order to reduce noise, a path between two unique nodes is neglected if it is not validated by two reads or more. When confronted with discrepancies between long reads, Rock Band interrupts its progression and flags the corresponding node as non-unique.

4.3 The Pebble algorithm

The Pebble algorithm was also designed to connect the unique nodes identified previously but using paired-end information. For each unique node, chosen in an arbitrary order, it iteratively estimates distances from that node, extends it to the next unique node, then merges the distance information provided by both nodes.

4.3.1 Building a primary scaffold

Before resolving repeats, it is necessary to organise read-pair information in a condensed and convenient structure. For any two nodes in the graph, Velvet enumerates the reads or mate-pairs which connect them. Using the maximum likelihood estimator (MLE) described below, it estimates the distance between them. The complete set of estimated inter-node distances is called the *primary scaffold*. Because each contig is represented by two nodes, one for each strand, orientation is retained.

Assuming a series of independent observations $\{X_i\}_{i=1..n}$ that follow a normal distribution of mean μ and of known variance σ^2, the MLE of the mean μ is given by:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{4.2}$$

It has the following properties:

$$E(\hat{\mu}) = \mu \quad \text{(unbiased)} \tag{4.3}$$

$$\sigma_{\hat{\mu}}^2 = \frac{\sigma^2}{n} \tag{4.4}$$

If the variables being measured have a normal probability distribution function centred on μ, but with different variances $\{\sigma_i^2\}_{i=1..n}$ then the formulas become:

$$\hat{\mu} = \frac{\sum_{i=1}^{n} \frac{X_i}{\sigma_i^2}}{\sum_{i=1}^{n} \frac{1}{\sigma_i^2}} \tag{4.5}$$

It has the following properties:

$$E(\hat{\mu}) = \mu \quad \text{(unbiased)} \tag{4.6}$$

$$\sigma_{\hat{\mu}}^2 = \frac{1}{\sum_{i=1}^{n} \frac{1}{\sigma^2}} \tag{4.7}$$

Therefore it is possible to merge the data from multiple insert libraries despite their different properties. It is computationally convenient, as it allows the calculation of the estimator and its variance in linear time, as the information is extracted.

Thi approach makes a number of simplifying assumptions. Firstly, it supposes that each insert library has a normal length distribution. Secondly, it only accounts for read-pairs which successfully connect two given nodes, thus biasing the insert length distribution by interval censoring. To be exhaustive, the MLE would have to consider the likelihood of the read-pairs which fail to connect the two nodes. However, this formalization would be much more expensive to calculate, because of the quadratic number of operations, as the information of each read-pair would have to be integrated in many MLE calculations simultaneously.

The precision of the estimator was tested on experimental Illumina reads produced from the genome of *Pseudomonas syringae* (Farrer et al., 2009). The expected insert length length was 400bp, so connections between unique nodes which were farther apart than 1000bp were discarded as noise. The results of this test on Figure 4.3 show that, as expected, the variance of the estimator decreases as the number of measurements increases. There is a bias towards underestimating the distance between contigs presumably due to interval censoring.

After establishing for each unique node a list of its primary neighbours, Pebble removes connections between unique nodes which are not supported by enough evidence. To start with, it discards distance estimates which were derived from fewer mate-pairs than a given cutoff (by default 10). Assuming that the distance estimate is correct, it then estimates the expected number of mate-pairs which should connect the two contigs (cf. below). If the observed number is below 10% of that expected values, then the distance estimate is deleted.

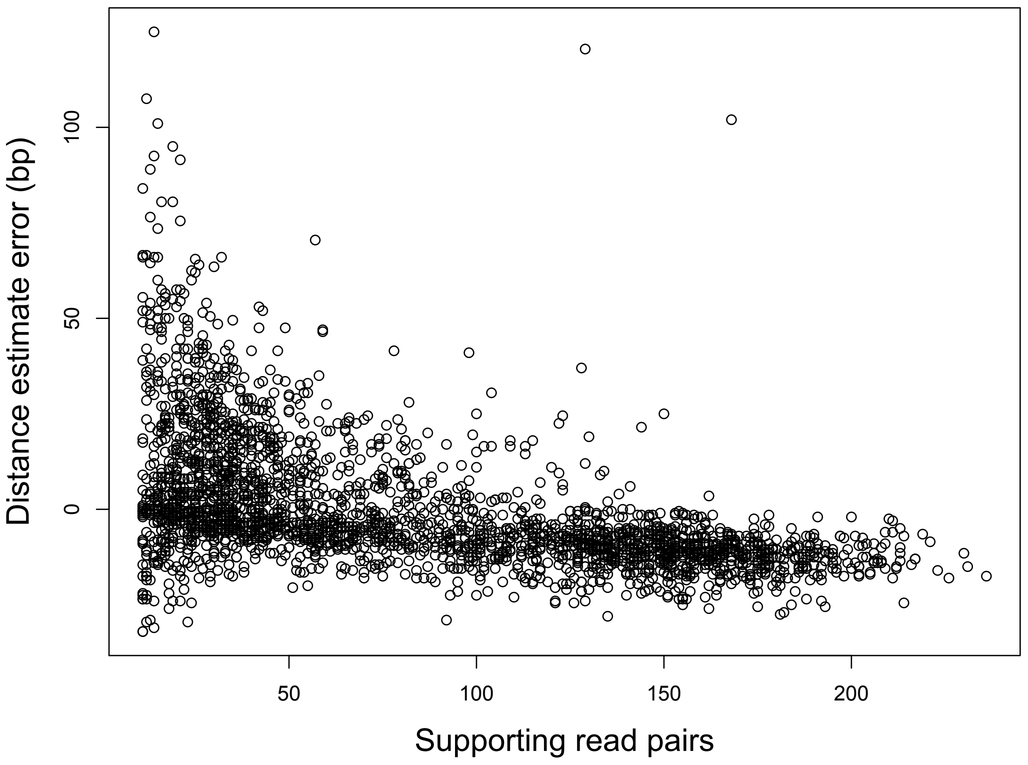


Figure 4.3: Test of the contig distance MLE on experimental data

Unique contigs from the *P. syringae* assembly were queried for paired-end read connections. If the contigs in a connection were mapped more than 1kb apart, it was discarded as incorrect.

Estimating the distance between contigs

Although this calculation is quite straightfoward, no example was found in manuals. The details are therefore provided below.

It is assumed a property is observed with different measuring methods, which have different standard deviations (SD).

Each observation is assumed an independent random variable defined by a normal distribution centered on μ and of known variance (provided, for example, by previous tests). This produces n results of the kind:

$$X_1 \pm \sigma_1;\ X_2 \pm \sigma_2;\ X_3 \pm \sigma_3;\ ...X_n \pm \sigma_n$$

The maximum likelihood estimator, $\hat{\mu}$, is the value for which μ maximises the function:

$$L(\mu) = f(X_1, X_2..., X_n \mid \mu, \sigma_1^2, \sigma_2^2...\sigma_n^2)$$

The observations are assumed independent, therefore:

$$L(\mu) = \prod_{i=1}^{n} f(X_i \mid \mu, \sigma_i^2)$$

The measurements are assumed to have a normal distribution of errors, therefore:

$$L(\mu) = \prod_{i=1}^{n} \mathcal{N}(X_i \mid \mu, \sigma_i^2) = \prod_{i=1}^{n} \frac{1}{\sigma_i\sqrt{2\pi}} e^{-(X_i-\mu)^2/2\sigma_i^2}$$

$$l(\mu) = log(L(\mu)) = \sum_{i=1}^{n} -log(\sigma_i) - \frac{1}{2}log(2\pi) - \frac{(X_i-\mu)^2}{2\sigma_i^2}$$

$$\frac{\partial}{\partial\mu} l(\mu) = \sum_{i=1}^{n} \frac{X_i - \mu}{\sigma_i^2}$$

$$\frac{\partial}{\partial\mu} l(\mu) = \sum_{i=1}^{n} \frac{X_i}{\sigma_i^2} - \mu \sum_{i=1}^{n} \frac{1}{\sigma_i^2}$$

From this affine function it is obvious that $l(\mu)$ and therefore $L(\mu)$, are both maximal when:

$$\mu = \hat{\mu} = \frac{\sum_{i=1}^{n} \frac{X_i}{\sigma_i^2}}{\sum_{i=1}^{n} \frac{1}{\sigma_i^2}}$$

Note: when returning to the standard case where all the σ_i are equal, the standard result can be obtained:

$$\hat{\mu} = \frac{\sum_{i=1}^{n} X_i}{n}$$

The variance of the estimator $\hat{\mu}$ can now be calculated:
If $K = \frac{1}{\sum_{i=1}^{n} \frac{1}{\sigma^2}}$, then:

$$\hat{\mu} = K \sum_{i=1}^{n} \frac{X_i}{\sigma_i^2}$$

Hence:

$$\sigma_{\hat{\mu}}^2 = K^2 \sigma^2 (\sum_{i=1}^{n} \frac{X_i}{\sigma_i^2})$$

Because of the independence of the X_i:

$$\sigma_{\hat{\mu}}^2 = K^2 \sum_{i=1}^{n} \frac{\sigma_{X_i}^2}{\sigma_i^{2^2}}$$

$$\sigma_{\hat{\mu}}^2 = K^2 \sum_{i=1}^{n} \frac{1}{\sigma_i^2} = K$$

Note: returning to the standard case where all the σ_i are equal, the standard result is obtained:

$$\sigma_{\hat{\mu}}^2 = \frac{1}{\sum_{i=1}^{n} \frac{1}{\sigma^2}} = \frac{\sigma^2}{n}$$

Note: if the σ_i are ordered by increasing value:

$$\frac{1}{\sigma_{\hat{\mu}}^2} - \frac{1}{min(\sigma_i^2)} = \sum_{i=1}^{n} \frac{1}{\sigma_i^2} - \frac{1}{\sigma_1^2} = \sum_{i=2}^{n} \frac{1}{\sigma_i^2} \geq 0$$

Hence:

$$\sigma_{\hat{\mu}}^2 \leq min(\sigma_i^2)$$

Estimating the expected number of mate-pairs between contigs

The objective is to estimate to how many mate-pairs would connect two nodes A and B, assuming that the distance D between the two is known.

This calculation will only use the lengths of A and B (which will also be written A and B for simplicity), the distance D separating them and the number of reads on each of them. Renaming the nodes if necessary, it is assumed that A is longer than B. The expected number of connecting mate-pairs will be obtained by calculating the probability $P = P(r \rightarrow B \mid r \in A)$ for a read r on A to connect to B, then multiply it by the number of reads on A.

Assuming that the distribution of reads is uniform over the length of A, P can be expressed as a sum of probabilities for each possible start coordinate p_r of r on node A:

$$P = \sum_{i \in A} P(r \rightarrow B \mid p_r = i) P(p_r = i) = \sum_{i=1}^{A} \frac{P(r \rightarrow B \mid p_r = i)}{A} \tag{4.8}$$

P can then be transformed into a sum of probabilities on the insert length of r, $l(r)$:

$$P = \frac{1}{A} \sum_{i=1}^{A} P(i + D \leq l(r) \leq i + D + B - 1) \tag{4.9}$$

Assuming that $l(r)$ folows a normal distribution of mean μ and variance σ^2, its probability distribution function (pdf) is noted $\varphi_{\mu,\sigma}$:

$$P = \frac{1}{A}\sum_{i=1}^{A}\int_{x=i+D}^{i+D+B-1}\varphi_{\mu,\sigma}(x)dx \tag{4.10}$$

$$P = \frac{1}{A}\sum_{i=1}^{A}\int_{\Re} S(i+D, i+D+B-1, x)\varphi_{\mu,\sigma}(x)dx \tag{4.11}$$

where $S(a,b,x)$ returns 1 if $a \leq x \leq b$ and 0 otherwise. Summing the terms under the integration sign, an easily computable approximation is found:

$$P = \frac{1}{A}\int_{\Re}\sum_{i=1}^{A} S(i+D, i+D+B-1, x)\varphi_{\mu,\sigma}(x)dx \tag{4.12}$$

$$P \approx \frac{1}{A}\int_{\Re} F(A,B,D,x)\varphi_{\mu,\sigma}(x)dx \tag{4.13}$$

where F(A,B,D,x) is the continuous function, piecewise linear, which returns (cf. Figure 4.4):

$$F(A,B,D,x) = \begin{cases} 0 & \text{if } x \leq D \\ x - D & \text{if } D \leq x \leq D+B \\ B & \text{if } D+B \leq x \leq D+A \\ A+B+D-x & \text{if } D+A \leq x \leq D+A+B \\ 0 & \text{if } D+A+B \leq x \end{cases} \tag{4.14}$$

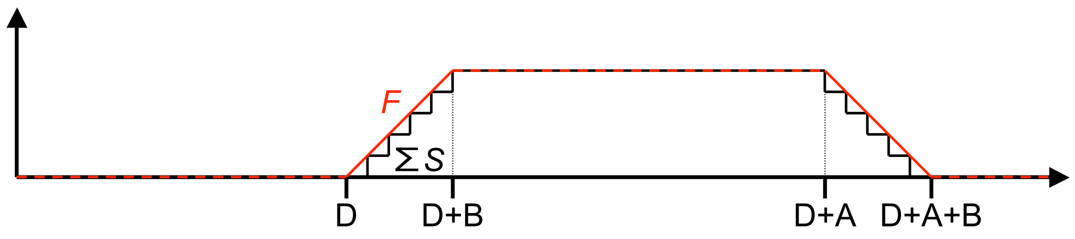


Figure 4.4: Schematic diagram of S and F functions

Changing variables with $u = \frac{x-\mu}{\sigma}$ the expression becomes, using φ

as pdf associated to $N(0,1)$:

$$P \approx \frac{1}{A} \int_{\Re} F(A, B, D, u\sigma + \mu)\varphi(u)du \tag{4.15}$$

Defining new parameters:

$$\begin{array}{rcl} \alpha & = & \frac{D-\mu}{\sigma} \\ \beta & = & \frac{D+B-\mu}{\sigma} \\ \gamma & = & \frac{D+A-\mu}{\sigma} \\ \delta & = & \frac{D+A+B-\mu}{\sigma} \end{array}$$

the integral becomes:

$$\begin{array}{rcl} P & \approx & \frac{1}{A}\Big[\int_{\alpha}^{\beta} \sigma(u-\alpha)\varphi(u)du \\ & + & \int_{\beta}^{\gamma} B\varphi(u)du \\ & + & \int_{\gamma}^{\delta} -\sigma(u-\delta)\varphi(u)du\Big] \end{array} \tag{4.16}$$

Using the integrative properties of φ, the integral is calculated explicitly:

$$\begin{array}{rcl} P & \approx & \frac{1}{A}\left[\sigma\left(\varphi(\alpha) - \varphi(\beta) - \frac{\alpha}{2}\left(erf\left(\frac{\beta}{\sqrt{2}}\right) - erf\left(\frac{\alpha}{\sqrt{2}}\right)\right)\right)\right. \\ & + & \frac{B}{2}\left(erf\left(\frac{\gamma}{\sqrt{2}}\right) - erf\left(\frac{\beta}{\sqrt{2}}\right)\right) \\ & - & \left.\sigma\left(\varphi(\gamma) - \varphi(\delta) - \frac{\delta}{2}\left(erf\left(\frac{\delta}{\sqrt{2}}\right) - erf\left(\frac{\gamma}{\sqrt{2}}\right)\right)\right)\right] \end{array} \tag{4.17}$$

where erf is the standard error function associated to the normal distribution. P is then multipled by the number of reads on A to obtain an approximate expected number of mate-pair connections between the two nodes.

The advantage of this formula is that a result can be quickly calculated from just seven numerical parameters. Unfortunately, the error of the approximation could not be bounded. Instead, tests were run on experimental Illumina data from the sequencing of *Pseudomonas syringae* (Farrer et al., 2009). The expected insert length length was 400bp, so connections between unique nodes were judged correct or incorrect depending on whether the distance between the contigs on the reference was smaller or greater than 1000bp. Connections which were flagged as correct but with fewer than 10 supporting read pairs, were discarded as they would be in the pipeline. This filter was not applied to the incorrect connections because none would have passed this second selection. The results of this test are displayed in Figure 4.5. The estimated value was very close to the observed count in the case of the correct connections.

This function is an efficient filter against noise. In this example, after removing connections for which the number of read pairs was below 10% of the expected value, only 0.7% of the retained connections were erroneous. In this example these false positive hits would not have been selected because the number of connecting read pairs was below the default cutoff ($= 10$). On the other hand, all the correct connections passed the filter. The addition of this filter was seen to significantly reduced the number of misassemblies created by the Pebble algorithm.

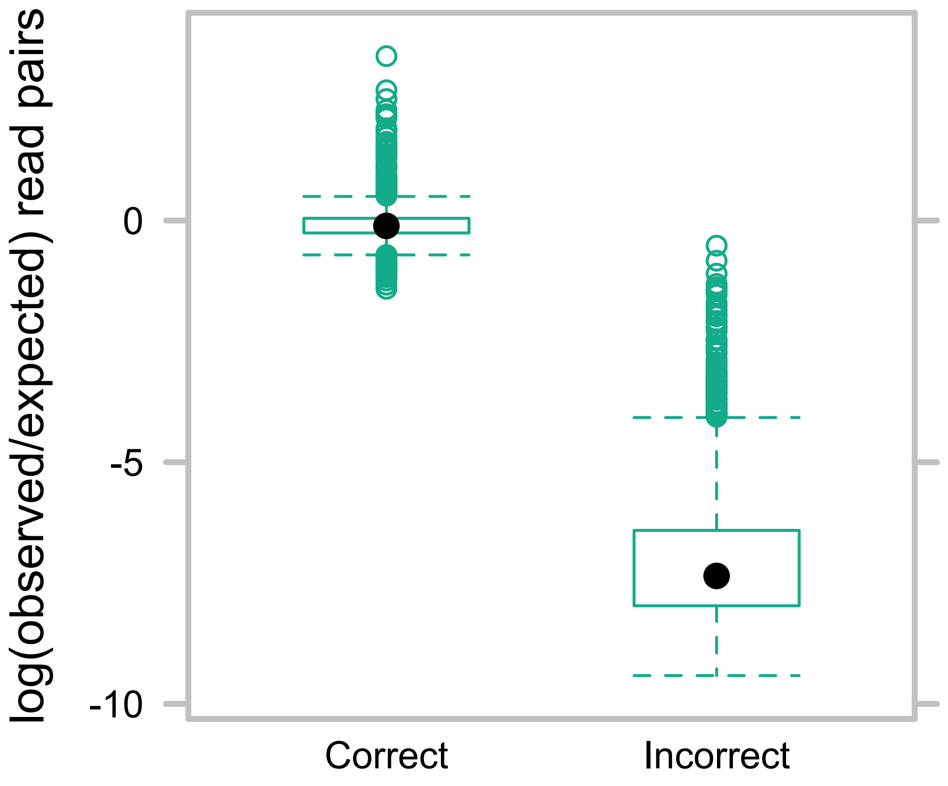


Figure 4.5: Test of the paired-end multiplicity estimator on experimental data

Unique contigs from the *P. syringae* assembly were queried for paired-end read connections. If the contigs in a connection were mapped less than 1kb apart, it was flagged as correct, otherwise it was flagged as incorrect. This box plot represents the log ratio of the number of observed divided by the number of expected connecting read pairs.

4.3.2 An exhaustive approach to scaffolding

A comprehensive method which was briefly considered was to use spectral graph theory to order the nodes. Spectral graph theory is generally used as a means of clustering nodes in a graph, however it can also be used to order them. Saerens et al. (2004) describe and demonstrate how Principal Component Analysis (PCA) can be applied to graphs and its relationship with spectral graph theory.

From a physical point of view this is analogous to resolving eigenfunctions of the parabolic equation of heat convection on the graph, hence the use of a Laplacian matrix. The closer nodes are to one another, the closer their temperatures are. The lowest energy wave is presumably the one that flows monotonically from one end of the genome to the other.

From a statistical point of view, this is analogous to resolving a PCA of the nodes. If neighbouring nodes are correlated, then presumably the first eigenvector, which explains most of the dissimilarites, is mainly due to genomic coordinates.

Using the mate pair information, it is possible to build a symmetric weighted Laplacian matrix of the graph. This matrix has a row and a column for each node, filled with the following values for each pair of nodes (u, v) based on the inter-node distance function d (as calculated above):

$$L(u,v) = \begin{cases} \sum_{w \neq u} \frac{1}{d(u,w)} & \text{if } u = v; \\ \frac{-1}{d(u,v)} & \text{if } u \neq v \text{ and u and v are adjacent;} \\ 0 & \text{otherwise.} \end{cases} \tag{4.18}$$

Once this real symmetric matrix is built, an eigenvector corresponding to the lowest eigenvalue can be used to order the nodes of the graph.

However, this method was not implemented. It does not take into account the double-strandedness of the contigs and can be disturbed if a repeated contig is inserted in the analysis.

4.3.3 Schematic diagram of the construction of a secondary scaffold

Before trying to extend a unique node A, it is necessary to establish which other nodes are in its vicinity. The first available information is the primary scaffold around A. However, this set of distances is generally insufficient to extend it. If the insert length is longer than the length of A, then there is no primary information on its immediate neighbourhood. For this reason it is necessary to compute a set of local distances or *secondary scaffold*, relative to A.

The SHORTY (Chen and Skiena, 2007) and Breadcrumb (Zerbino and Birney, 2008) algorithms were based on the concept of projecting the image of a unique k-mer or node onto its neighbourhood, using paired-end reads. In Pebble, the approach is slightly different: for a given unique node the algorithm enumerates the neighbouring unique nodes which project onto it, then uses those to estimate distances around it.

Figure 4.6 describes the process used to construct the secondary scaffold. *Primary neighbours* are the unique nodes which share a connection with a given node A in the primary scaffold. All the primary connections of these nodes are extracted and the nodes to which they lead are flagged as *secondary neighbours*. Using the known primary distance estimates, it is possible, by a simple subtraction, to derive an estimate of the distance between A and the secondary neighbours.

The secondary scaffold takes into account orientation by using algebraic distances. Contigs which are upstream of the reference point are assigned negative distances, whereas contigs downstream are assigned positive ones. This notation allows the subtraction of distances without any constraint. However, derived distance estimates have to be handled with caution, because secondary neighbours are not necessarily unique. It is possible that the distance estimate from a unique contig to a repeated one is actually based on the distances from the unique node to separate copies of the repeat. This means that distances in the secondary scaffold

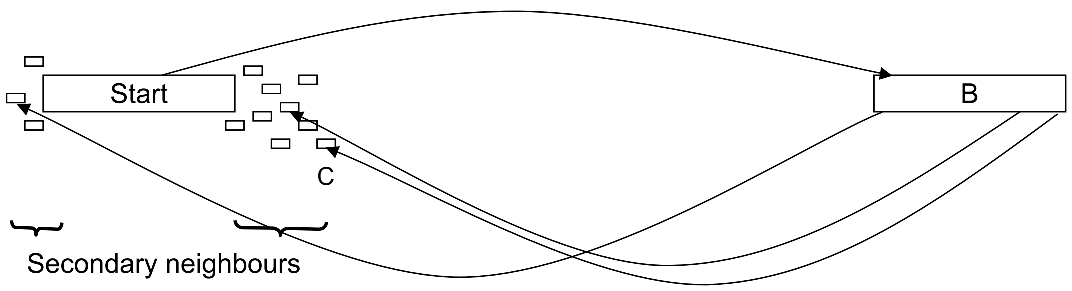


Figure 4.6: Construction of the secondary scaffold

From a unique node *A*, the Pebble algorithm incorporates all the distances relative to this node found in the primary scaffold. For each unique node *B* which is connected to *A*, Pebble follows the primary connections associated to *B*, thus flagging secondary neighbours of *A*, such as *C*. Assuming that all the nodes are laid out on linearly on the genome, the distance from *A* to *C* is estimated to be the difference between the distance from *A* to *B* and that from *B* to *A*.

tend to be fuzzy.

4.3.4 Heuristic search through the local scaffold

Once a set of distance estimates from node *A* have been determined, a plausible path connecting it to the nearest unique node is searched for. The strategy employed is a heuristic depth-first search. The algorithm advances through the graph, from node to node, through the existing arcs.

At each iteration, it searches through the outgoing arcs and chooses the direct neighbour which is putatively closest to the starting point. In order to avoid loops, priority is systematically given to nodes which have not yet been visited or have been visited the fewest amount of times.

If a path is found to the next unique node, then the sequence associated to the path and the sequence of the destination node are appended to the starting node. All the read information from the destination is transposed onto the starting node. The destination node can then be deleted.

Reads belonging to the path are left in place. By default the nodes along the path are considered to be potential repeats. This is reads are not tracked in repeated regions.

As two unique nodes are being merged into one, it is vital to also merge the corresponding scaffolding information, to keep the process going. This operation is directly based on the original construction of the local scaffold, since the formula allows incrementally addition of information, both from primary and derived distances. Admittedly this use of the formula ignores the fact that the derived distances around one node are not necessarily independent from those around a neighbouring node. Nonetheless, this bias is neglected for the sake of computational speed.

It is not always possible to extend a node to another unique contig. This can be due to coverage gaps or to anomalies detected as described below. However, the search for a path between two contigs is asymmetric, depending on which of the two local scaffold was built around. Whenever two contigs, using read-pair information, are estimated to be neighbours, but a path cannot be found from both ends, the two are scaffolded together. They are merged as if a path were found, but separated by a buffer sequence of undetermined nucleotides (marked as Ns). The length of this buffer is equal to the distance estimate between the nodes.

Because Pebble is a greedy algorithm, special care must be taken to avoid creating misassemblies. These can be detected through inconsistencies in contig distances. When extending a unique contig, the algorithm checks that it terminates on the nearest unique contig in the local scaffold and did not find a path directly to a farther one. Such inconsistencies can be caused by the variance in the distance estimator, especially when using mate-pair libraries with large insert length variance. However, for the sake of precaution, the process stops whenever it meets such an occurrence.

Another pitfall of this heuristic depth-first search is looping within highly repetitive regions of the graph before eventually finding the correct path out. These regions are not resolved but simply jumped over. In order to prevent looping, the search is interrupted whenever a node was visited twice, without having visited a new node in between.

4.4 Simulations

The two algorithms, RockBand and Pebble, were tested on simulated datasets generated from the same genomic regions as in the previous chapter. Randomly placed reads were generated according to three different scenarios. In the first case, perfect reads were generated from the reference sequence. In the second, single base-pair errors were inserted at a rate of 1%. Finally, a diploid sequencing project was simulated. An alternate version of the reference was created where artificial SNPs were introduced randomly at a rate of 1 per 500bp. Half of the reads were generated from the original reference, half from the modified reference. All reads contained random errors as above.

4.4.1 Mixed-length assemblies

In the first experiment, various mixtures of long and short reads were tested. The results, shown in Figures 4.7 and 4.8, are qualitatively as expected. The more long reads are available, the more repeats are covered, the more contigs can be merged together. Similarly, as long reads get longer, then the number of repeats which can be bridged goes up, so the resulting contigs are longer.

The *Plasmodium falciparum* datasets proved difficult to assemble, because of the high fragmentation of the de Bruijn graph. This in turn required more memory, which prevented from running many runs in parallel. Also, the results on *Bordetella pertussis* are extremely uniform. This is presumably because large parts of that genome are easy to assemble, but it broken by the numerous copies of insertion sequence element IS*481* (Parkhill et al., 2003), which is roughly 1kb long (Stibitz, 1998). Therefore practically all assemblies obtain similar results. This example shows how the use of longer read can be blocked by particular repeat patterns.

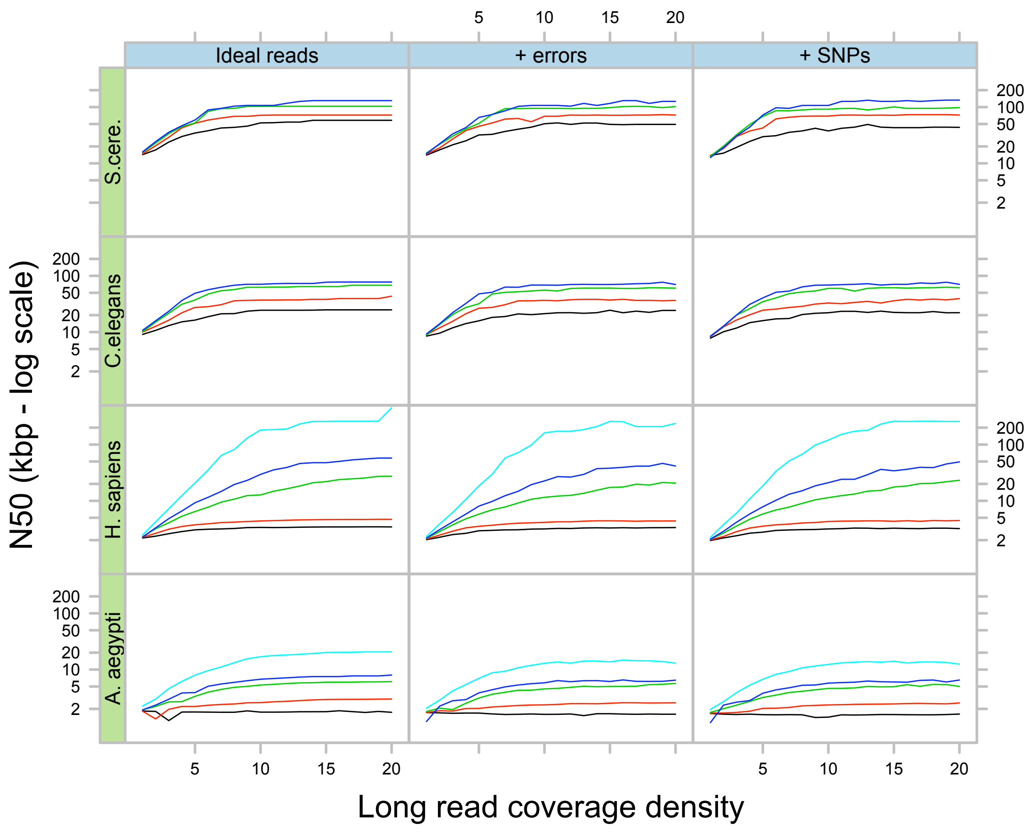


Figure 4.7: Results of simulations using various long / short read mixtures

Final contig N50 as a function of long read concentration, in four different eukaryotic species and three different simulation scenarios. The length of the long reads is represented by the colour of the curves: 100 (black) 200 (red) 400 (green) 500 (blue) and 1000bp (light blue). In this set of simulations, random 35 bp short reads were generated at a constant coverage of 50X. Random long reads were generated at varying coverage values, namely all integer values from 1 to 20X. The long reads were either 100, 200, 400, 500 or 1000bp long.

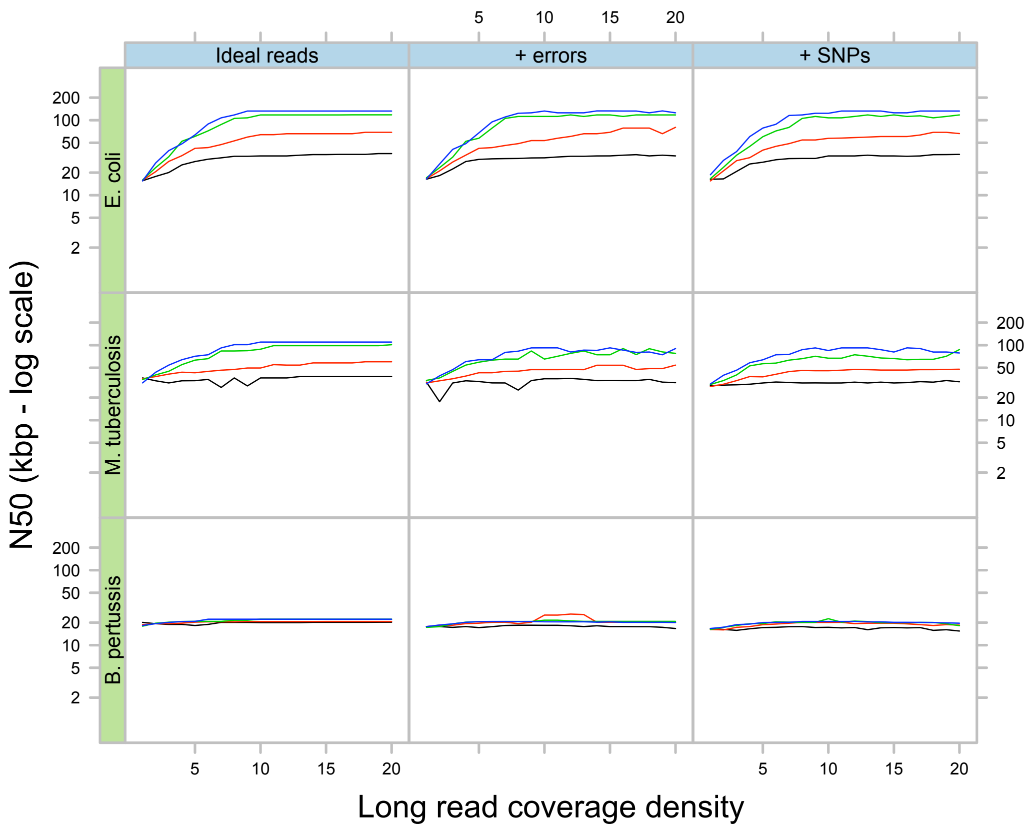


Figure 4.8: Results of simulations using various long / short read mixtures

Final contig N50 as a function of long read concentration, in three different bacterial species and three different simulation scenarios. The length of the long reads is represented by the colour of the curves: 100 (black) 200 (red) 400 (green) 500 (blue) and 1000bp (light blue). In this set of simulations, random 35 bp short reads were generated at a constant coverage of 50X. Random long reads were generated at varying coverage values, namely all integer values from 1 to 20X. The long reads were either 100, 200, 400, 500 or 1000bp long.

4.4.2 Paired-end assemblies

In the second simulation, several insert lengths were tested, from 100bp to 12kb. The results shown in Figures 4.9 and 4.10 show that as the insert length increases, the N50 also increases, until it stabilises and eventually decreases. The initial rise is simply due to the increased insert lengths, thus increasing the probability for nearby unique regions to be connected by mate-pairs. The final degradation is presumably linked to the higher variance of the insert lengths, which can cause Pebble to detect possible missassemblies and interrupt its progression.

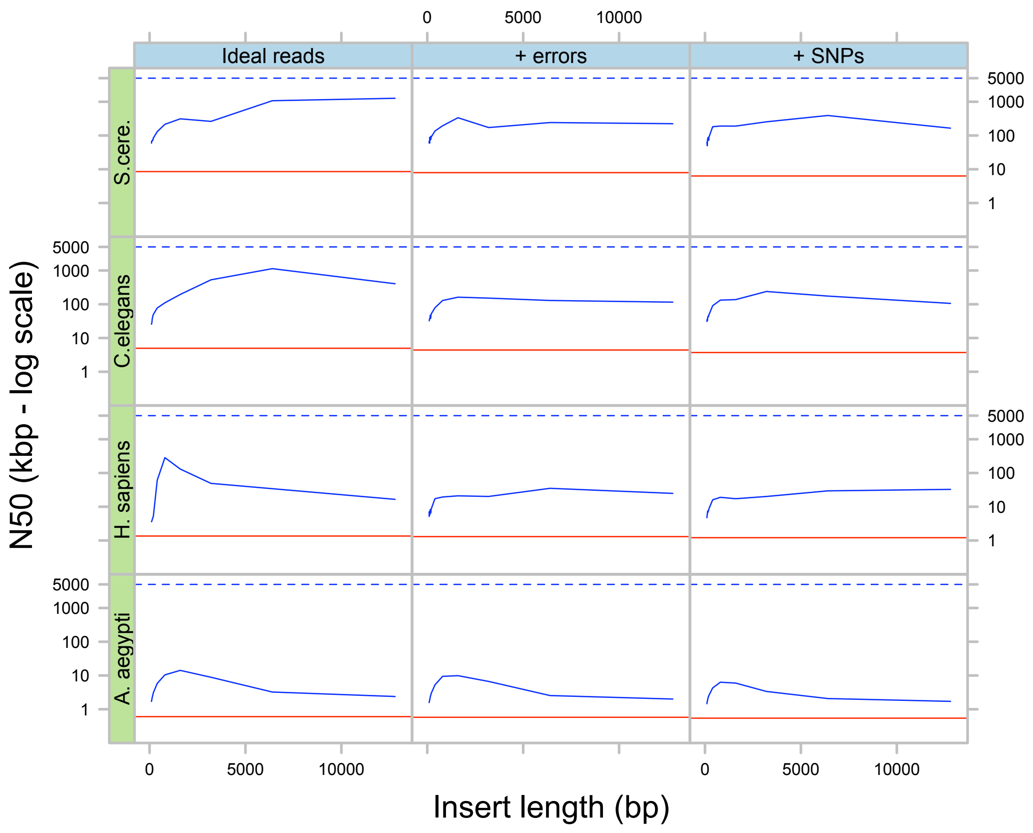


Figure 4.9: Results of simulations using various insert lengths

Final scaffold N50 as a function of insert length, in four different species and three different simulations scenarios. The horizontal red lines represent the initial N50 after error removal and before repeat resolution. The dashed blue lines represent the highest possible N50, namely the length of the sequence being sampled. Random 35bp paired-end short reads were generated at a constant coverage of 50X. The simulated insert lengths were either: 100, 200, 300, 400, 500, 600, 700, 800, 1600, 3200, 6400 or 12800bp long. The actual insert lengths were randomly varied around the expected value with a standard deviation of 10% of that length.

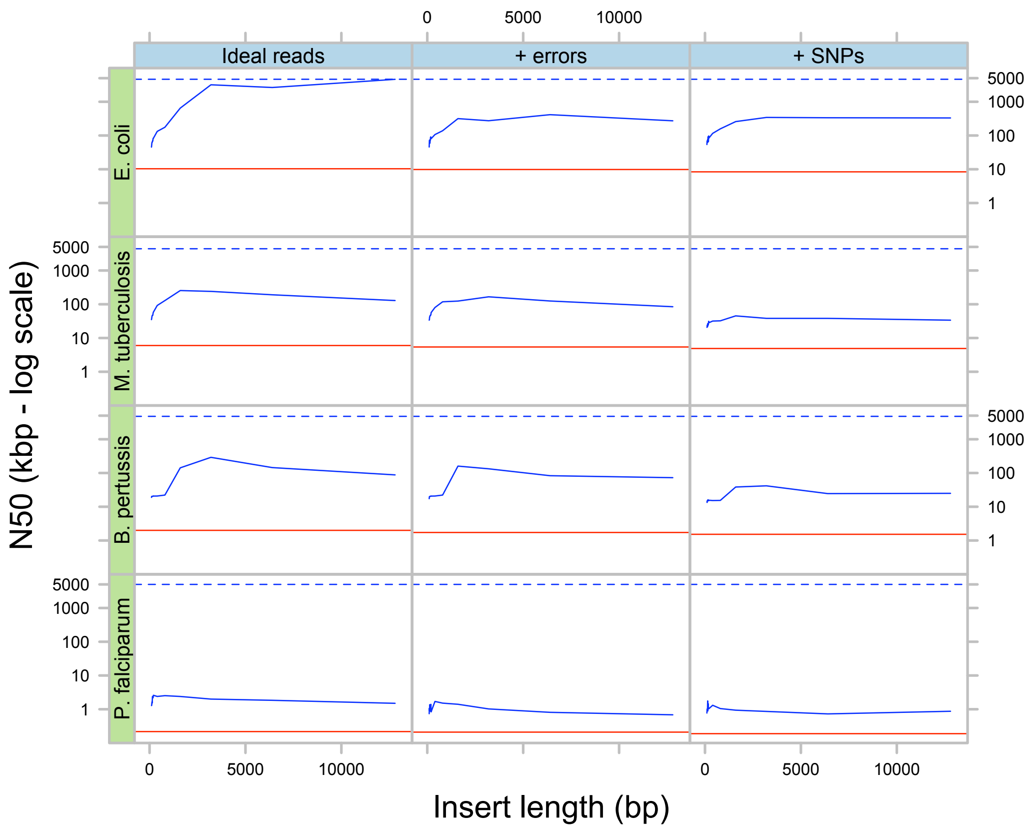


Figure 4.10: Results of simulations using various insert lengths

Final scaffold N50 as a function of insert length, in four different species and three different simulations scenarios. The horizontal red lines represent the initial N50 after error removal and before repeat resolution. The dashed blue lines represent the highest possible N50, namely the length of the sequence being sampled. Random 35bp paired-end short reads were generated at a constant coverage of 50X. The simulated insert lengths were either: 100, 200, 300, 400, 500, 600, 700, 800, 1600, 3200, 6400 or 12800bp long. The actual insert lengths were randomly varied around the expected value with a standard deviation of 10% of that length.

4.5 Experimental assays

4.5.1 *Pseudomonas syringae*

Pebble was tested on actual Illumina data, generated from *Pseudomonas syringae*. After running Velvet, 274 contigs longer than 100bp where obtained, with an N50 contig length of 104kb and a maximum contig length of 314kb. The run took in total 5min 38s and required 1.2GB of RAM. Aligning the contigs to the known reference, only 15 short misassembled contigs were found, representing a total of 26kb. As expected, these were found to map onto highly repetitive regions.

If scaffolding is turned off and the contigs are broken up at every buffer, then the contig N50 goes down to 24kb. However, many of the buffers are very short. If the contigs are only broken up at buffers which are estimated to be strictly longer than 1bp, then the N50 of the "near-contigs" becomes 48kb

As way of comparison these results were compared to those obtained with EULER-USR (Chaisson et al., 2009) with the help of Mark Chaisson. EULER was found to be the leading paired-end micro-read assembler (Simpson et al., 2009). The ALLPATHS assembler (Butler et al., 2008) could not be used because it is designed to function with two different insert lengths, unlike Velvet which can handle one or several libraries indifferently.

The details of this comparison are in Table 4.1. Whereas Euler does produce longer contigs, Velvet's near-contigs are comparable to Euler's. Moreover, Velvet's scaffolds are significantly longer.

4.5.2 *Drosophila melanogaster* BACs

Finally, Pebble was tested on Illumina data from a *D. melanogaster* BAC. This assembly produced a single contig 147.362bp long, which aligned with a 0.02% global error rate. Figure 4.11 shows the alignment of this bac onto the known reference, as calculated with BLASTZ (Schwartz et al.,

Table 4.1: Comparison of Velvet and Euler assemblies

Assembly	N50 length (kb)	Maximum length (kb)	Contig or scaffold count	Coverage (%)
EULER contigs	40	215	598	103.0
Velvet contigs	24	134	595	97.9
Velvet near-contigs	48	157	430	97.9
EULER scaffolds	51	215	620	107.8
Velvet scaffolds	96	314	274	98.4

Comparison of Velvet and Euler assemblies. The contig or scaffold count corresponds to the number of contigs or scaffolds longer than 100bp. Near-contigs are defined as scaffolds which are broken up only if the distance between two contigs is estimated to be strictly greater than 1bp.

2003).

4.5.3 *Plasmodium falciparum*

Thomas Dan Otto from the Sanger Institute ran tests of Velvet on experimental Illumina runs of *Plasmodium falciparum* IT. The DNA preparation was PCR free to reduce amplification bias, and the reads were paired with an insert length of 171bp. The final assembly had an N50 of 3.7kb and the largest contig was 40kb long. The total length of the assembly was 22.7Mb. The run required just above 3 hours and needed up to 74GB of RAM.

This experiment shows that Velvet can run on extremely AT-rich genomes and produce a near complete assembly. However, it required substantial resources, and the final contigs are significantly fragmented. Once again, this is to be imputed to the low complexity of the *Plasmodium* genome. Because the sequence is so AT-rich, spurious overlaps are very frequent, and create a dense network of tangles in the de Bruijn graph.

These results show that microreads can be used to obtain contigs of significant length even on complex genomes. However, the Velvet algorithms are designed to be applied in a *de novo* WGS genome assembly. The final chapter of this thesis described how the concepts discussed in previous chapters could be used for another application, structural variant analysis.

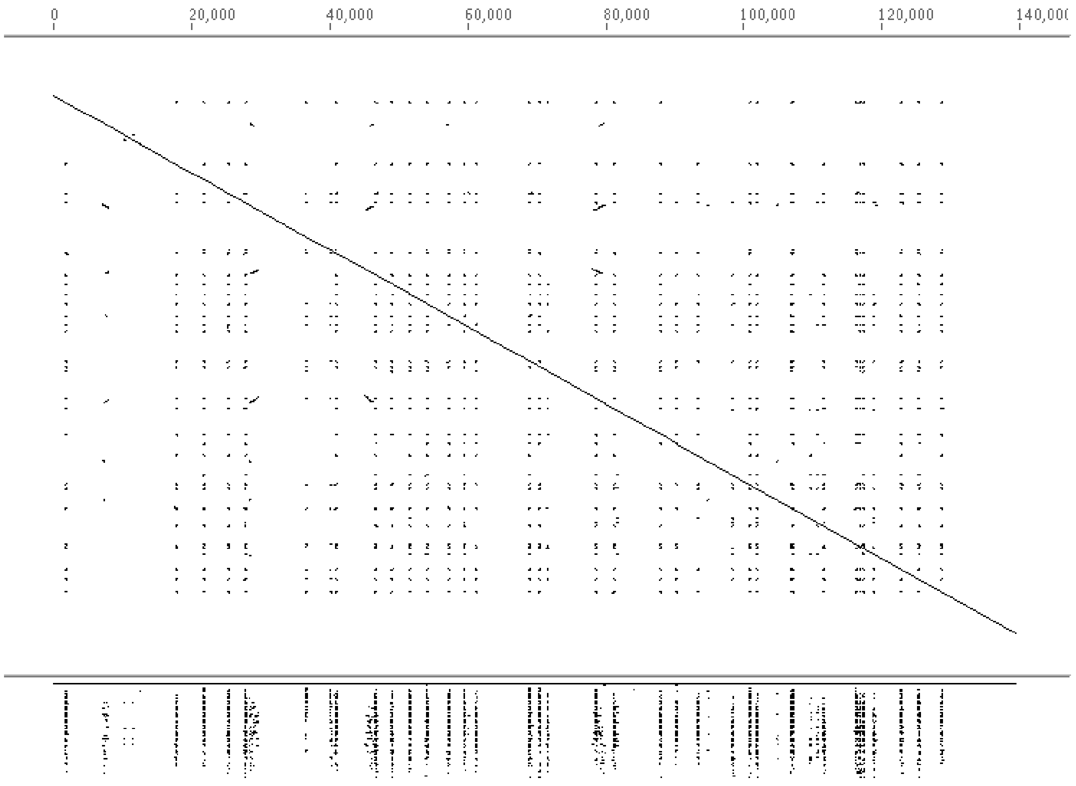


Figure 4.11: Alignment of the Velvet contig onto the reference
In the dotplot of the top frame, the horizontal axis represents reference coordinates, whereas the vertical axis represents coordinates in the contig. A clear reverse complementary alignment is visible. The lower frame represents in its vertical axis the quality of the alignmnent of all the hits along the reference. The contig aligns with uniform quality along it whole length.

Chapter 5

Comparing sequence variants

Genomic variation has been observed to occur at different scales, from single nucleotide variation to large-scale structural variants. In recent years, much attention has been focused on single nucleotide polymorphisms (SNPs). Major data collection projects such as the HapMap Project (The International HapMap Consortium, 2003) allowed *genome-wide association studies* (GWAS)(Hirschhorn and Daly, 2005) which could detect relevant variants for medical or evolutionary research. Such polymorphisms are commonly detected using oligonucleotide arrays (Matsuzaki et al., 2004). Thanks to next-generation sequencing, they can now be easily found by re-mapping re-sequencing reads onto a reference genome (Hillier et al., 2008). These technologies are sufficiently mature to support commercial direct-to-customer (DTC) genotyping services (Guttmacher and Collins, 2003).

Small insertions and deletions in the genome can be efficiently detected by analysing re-sequencing data as shown for example by Ossowski et al. (2008) at the Max Planck Institute for Developmental Biology. They developed a hybrid method that combines the computational efficiency of re-mapping with the flexibility of *de novo* assembly. Generally, short read alignment programs do not allow a single read to align with a reference if there are more than a given number (usually 3) of mismatching bases. This means that no read is mapped over features longer than that cutoff.

They observed such dips and gaps in the coverage of various strains of *Arabidopsis thaliana*. They used Velvet for the *de novo* assembly of all the unmapped reads and those mapped within 100bp of these loci. This way, they obtained contigs that revealed local variation compared to the reference. All the contigs which were verified experimentally proved to be entirely correct. *De novo* assembly can therefore be used to extend the sensitivity of re-mappers at the regions where the latter are constrained by their models.

Much larger structural variants (SVs) can significantly alter the architecture of the genome. I shall briefly describe how they are currently analysed and two projects which we undertook to visualise and detect them.

5.1 Structural variants

Whole regions of the genome can be inverted, duplicated or cut out and reinserted in the same chromosome (*transposition*) or in a different chromosome (*translocation*) (Brown, 2007). Occasionally, the whole genome is duplicated (Zhang, 2003). Such large scale alterations to the genome can have correspondingly large phenotypic effects. Rearrangements are often associated with tumors and disease (Kidd et al., 2008), but can also spur evolution by creating paralogy (Zhang, 2003). They are correspondingly less frequent than point mutations and can be used as reliable markers to date evolutionary events (Ehrlich et al., 1997).

SVs were first analysed by hand in simple model organisms (Palmer et al., 1987). It was suggested that evolutionary distances could be measured in terms of SV occurrences. The *edit distance* (Sankoff and El-Mabrouk, 2002) was defined as the minimal number of SV necessary to explain the differences between two genomes. Later, mathematical representations were adopted to formalise the problem. Genomes were represented as signed permutations of numbers, representing the order and orientation of genes. SVs could be formalised as specific permutations which could be applied to the genomes, thus creating a group structure (Kececioglu and Sankoff, 1995).

Computer scientists then worked on the problem of determining the edit distance of sequences based on this model. Hannenhalli and Pevzner (1999) developed a very powerful method to determine the *inversion distance* using a new structure called the *breakpoint graph* introduced previously by Bafna and Pevzner (1993). In this graph structure, genes are represented as nodes in a graph. Two nodes are connected by an edge if the corresponding genes are consecutive in one sequence but not in the other. The edge is colored depending on which sequence the genes are consecutive in. Several years later, Alekseyev and Pevzner (2007) realised the homology between the breakpoint and the de Bruijn graphs, combining them into the *colored de Bruijn graph*.

In parallel, the de Bruijn graph was being used for multiple sequence alignments. It became apparent that standard matrix alignments were inadequate for global whole-genome alignments. Because the system of columns implicitly requires the preservation of synteny, traditional alignments are incapable of representing common SVs. For this reason, graph-based alignment representations were developed, such as the Threaded Blockset Aligner (TBA)(Blanchette et al., 2004). However, TBA did not allow for inversions or duplications. In this context, the de Bruijn graph provided a convenient structure to extend traditional alignments, allowing for all common SVs. To compensate for the rigidity of exact *k*-mer matching, Raphael et al. (2004) defined the A-Bruijn graph, which was instead built on standard local pairwise alignments.

It has recently become possible to detect SVs using dense paired-end short read data (Kidd et al., 2008; Lee et al., 2008). Many methods were inherited from previous studies in assembly validation (Phillippy et al., 2008) where the mappings of paired end reads onto contigs were used to detect inconsistencies in an assembly. For example, coverage density can be used to detect overcollapsed repeats, whereas read-pair distances can be used to detect insertions or deletions. Assimilating the reference sequence to an erroneous contig, all rearrangements are detected in the same way as misassemblies.

However, this approach can only detect and place SV breakpoints. Other teams have attempted and sometimes succeeded in actually assembling the breakpoint sequences using Velvet. For example, Cheetham *et al* (pers. comm.) studied variation in humans. They mapped Illumina reads from a single individual onto the reference genome and detected SVs using the approach described above. They then assembled with Velvet read-pairs which were mapped near one of these breakpoints. In a number of cases they obtained contigs which clearly spanned over the breakpoint.

Similarly, Pleasance *et al* studied somatic mutations in tumourous cell lines (Bignell et al., 2007). They sequenced cells from both the tumour and healthy cells and mapped the reads onto a reference to discover breakpoints, much like the previous project. Comparing both datasets, they were able to distinguish common germline rearrangements from tumor specific somatic breakpoints. They then assembled the reads corresponding to the somatic mutations with Velvet, obtaining some bridging contigs (Pleasance - pers. comm.).

However, systematic *de novo* sequencing of rearrangement breakpoints is difficult. Many of rearrangement events occur next to interspersed repeats, which significantly complicate the process. When mapping the reads onto the reference, the reads that come from the connecting sequence are generally unmappable. If their mate pair maps onto a repeat, the alignment software has two options: either refuse to map the read at all or place it at random on one of the possible sites.

In the light of all these advances, the chapter presents the development of tools to visualise and assemble SV breakpoints from unassembled short read datasets.

5.1.1 Visualising structural variants

A method to visualise portions of the de Bruijn graph and observe SVs in unassembled data was first developed. The alignment is done using the basic Velvet functions: first building the graph using the reference sequence and the reads, then smoothing SNPs and other small variants with Tour Bus. Given sequence coordinates, a script then extracts all the information on the relevant nodes of the graph, along with their direct graph neighbours. It then creates an image representing this subgraph.

This extraction pipeline was tested on experimental Sanger reads provided by Dr. Julian Parkhill's team at the Sanger Institute.These reads came from a previously studied *Salmonella typhi* sample, characterised by a site of frequent inversion. The graph of the known locus built from this dataset is shown in Figure 5.1.

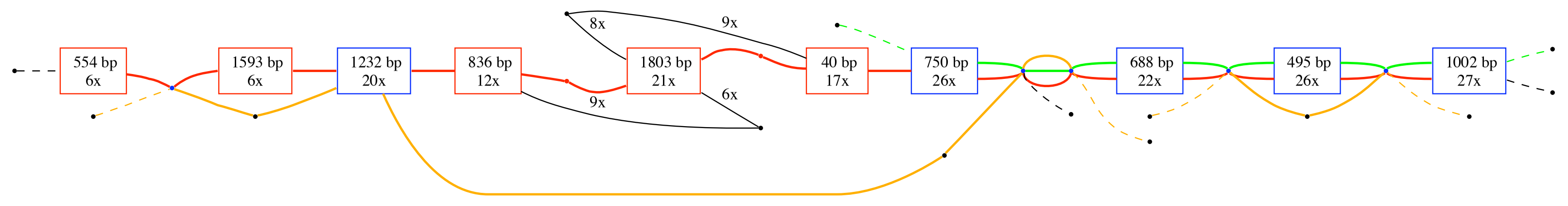


Figure 5.1: Graph neighbourhood of an inversion site

The queried sequence is represented by a thick red line going from left to right through various nodes. Nodes longer than 40bp are represented by a rectangle and shorter nodes are represented by a dot. Connected regions of the reference sequence are represented by thick colored lines. Dotted lines indicate the arcs where these connected regions are about to leave the graph neighbourhood of the query. Thin black lines correspond to novel sequences connected to the query, with their multiplicity indicated next to them.

The graph neighbourhood of the query sequence contains two types of features. Most of the nodes and arcs connected to the query also belong to the reference, but at different loci. The extractor script detects these connected regions and represents them alongside the reference, occasionally overlapping. It is visible from Figure 5.1 that the query sequence displays significant homologies around this locus; however, there are also some short nodes connected to the query by novel sequence. These structures correspond to sequences which appeared in the reads but not in the reference. The configuration of these arcs clearly indicates that an inversion took place at that region. The multiplicity of these arcs shows that they are probably not spurious chimeric reads.

This extractor script therefore allows the observation and efficient represent of a known SV. The de Bruijn graph representation condenses the information from the unassembled dataset, making the SV immediately recognisable on visual inspection of regions of interest.

5.1.2 Systematic *de novo* assembly of rearrangement breakpoints

In collaboration with Zamin Iqbal, Velvet was specially modified and tested for the *de novo* assembly of SV breakpoints. The pipeline, outlined in Figure 5.2, consisted in taking the output from a re-mapper and selecting the read pairs which do not align consistently to the reference. These reads were then processed, along with the reference, using Velvet. Once the graph was built and simplified with Tour Bus, all nodes which did not contain any read were removed in order to obtain contigs which spanned the sample-specific region and into both flanking regions.

This principle was first tested on simulated reads. A 5Mb section of chromosome 20 was selected and edited with the insertion of several fragments from chromosome X: two 100bp long, two 1kb long and two 10kb long. Random perfect readswere generated from this modified reference, aligned to the genuine reference and processed using the pipeline

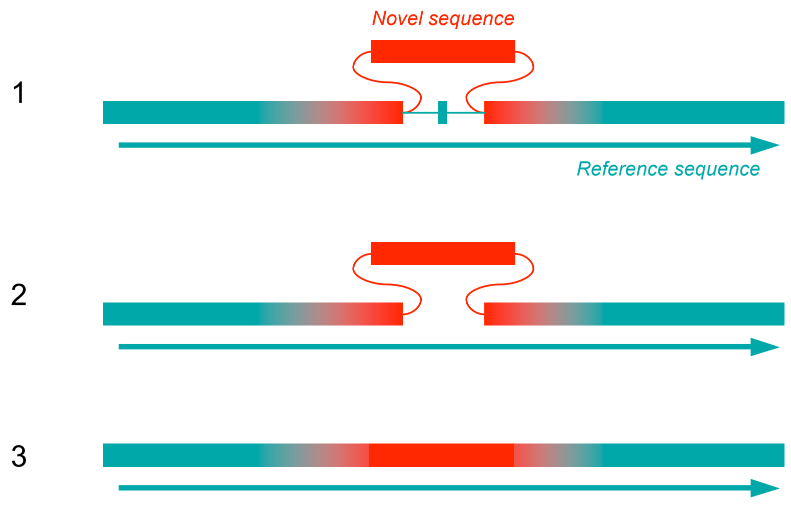


Figure 5.2: Analysis pipeline for comparative short read assembly

In the proposed pipeline, non-aligned read-pairs are assembled simultaneously to the reference. Some contigs are specific to the reference (green), other to the reads (red) and other a combination of both (red-green gradient). Reference specific nodes are removed (second step), allowing the concatenation of the other nodes into hybrid sample-reference contigs (third step).

described above. Initially, gaps appeared in the contigs because the reads corresponding to those loci were not present in the read selection and had presumably been mapped elsewhere on the short reference. This problem was resolved using the Pebble algorithm (cf. Chapter 4), allowing for the construction of scaffolds. A scaffold was obtained for each insertion.

The same test was run with reads with random errors. Coverage cutoffs had to be instroduced to reduce the signal noise. A different cutoff was applied to non-reference contigs, where reads are least likely to be re-mapped onto the reference, as opposed to contigs which belong to the reference, where the pre-selection of reads is likely to reduce the coverage. Once again, it was possible to obtain the desired contigs.

The approach was then tested with data from the 1000 Genomes Project (www.1000genomes.org). This project is focused on re-sequencing a large human population to obtain a summary of commonly found variants. Whereas most individuals were sequenced at a low coverage (2X), two parent-child trios were sequenced at higher depths (20X). This test was run on the chromosome 20 of one of the trio children (HapMap identifier NA12878) only.

The data was provided by David Carter and Li Heng at the Sanger Institute as output alignments of Illumina reads using the MAQ aligner (Li et al., 2008a). Each read pair was flagged according to the relative positions of the reads as mapped onto the reference. The reads could thus be flagged as either consistently orientated at appropriate distances, incorrectly orientated, at an unusual distance from one another or on different chromosomes. In extreme cases, one or both reads in a pair were not mapped at all onto the reference.

The pairs in which neither read mapped at all to the reference were rejected. They represented a very large set (160 million reads) which was difficult to manipulate and k-mer representation in this dataset was extremely sparse. For example, an estimated 99.36% of the k-mers were present ≤ 3 times. An attempt was made to assemble the remaining 0.64% of the reads, but resulted in extremely short contigs (N50=3bp). The use of these unmapped reads would have probably introduced much more stochastic noise than information to the process.

The set of read pairs which mapped partially or inconsistently to chromosome 20 were processed through the pipeline. 467 contigs longer than 100bp were produced, with an N50 of 187bp and a maximum length of $2,414$bp. To validate these contigs, they were aligned to 454 reads obtained from the same individual by the Baylor College of Medicine. More than half of the contigs (270/467) aligned along 90% of their length to one of the 454 read.

However, other groups, which were working on the same data within the 1000 Genomes Structural Variation group, were able to detect many more events, using spanning paired-end information. The sensitivity of the method was therefore tested by selecting 28 breakpoints found by Evan Eichler and his team (pers. comm.). The pipeline was re-run selecting only reads with anomalous mappings within 1kb of each breakpoint. Unfortunately most of these assemblies produced no contigs (19/28) and the remaining ones produced contigs with extremely low coverage. A probable explanation for this is that very few reads were selected from each locus. All but four of them were placed on an annotated repeat. MAQ randomly assigns ambiguous reads to one of the possible hits. Presumably, the reads which corresponded to the breakpoints and therefore did not map onto the reference, were matched to reads on repeats. The latter reads were then randomly mapped onto any of the repeat copies in the genome.

In conclusion, this attempt at detecting SVs lacked sensitivity, possibly because of the read selection step. The use of a standard aligner introduced an implicit bias in the experiment. Indeed, the function of the aligner is to map the reads as best it can onto a reference, minimising any variation. This reference bias is especially strong with repeated regions, which provide much ambiguity. The random assignment strategy adopted by MAQ is justified in many applications based on coverage statistics. However, in the case of *de novo* assembly, where utmost sensitivity is necessary to obtain sufficient coverage, a specific assignment process is required. It is therefore necessary to improve the mapping of the reads. Recent work, for example by Lee *et al* (Lee et al., 2008), aims at placing read alignments based on a probabilistic model of SVs. It is hoped that such methods will allow for the *de novo* assembly of SVs.

5.1.3 Scaled analysis of structural variants

Most attempts at analysing SVs were built on the assumption that large scale information was available, either gene ordering (Hannenhalli and Pevzner, 1999) or whole genome alignments (Dubchak et al., 2009; Ma et al., 2008). However, when starting from unassembled re-sequencing data, the analysis must be started from local information. It is therefore necessary to develop tools to simplify the de Bruijn graph as obtained from a short read dataset.

Conceptually, methods describe below aim to obtain the functionality of geographical maps found on the Internet: as one zooms out, small features are removed to make the large scale SVs apparent. In reverse, when zooming in towards a specific regions, minute details are progressively made apparent. This transformation can be applied to a single assembled genome, to analyse repeat patterns. Thanks to the properties of the de Bruijn graph, it can also be used to detect SVs between two or more genomes, whether assembled or not. This scaled representation of de Bruijn graphs was first used to develop an actual browser, but could conceivably be extended to a computational analysis of breakpoint graphs.

Graph crystallisation

The first step consists of developing a graph simplification process, which removes features from a complete de Bruijn graph to obtain more synthetic breakpoint graphs. This process is named crystallisation, by analogy to the simulated annealing (SA) algorithm (Kirkpatrick et al., 1983). As an arbitrary "temperature" function is lowered, "defects" (in this case small scale features) are removed to allow the creation of large "crystals" (in this case syntenic regions).

In this trial, the chosen implementation is much simpler than the stochastic approach of the true SA algorithm. An "importance" score is assigned to each node, equal to its length multiplied by its coverage. The nodes from the graph are then pruned in increasing order of importance, progressively reconnecting and concatenating the neighbouring nodes.

This process was able to gradually simplify the de Bruijn graph of the *E. coli* genome, gradually revealing conserved features of greater scale. Figure 5.3 represents the progressive crystallisation of the finished *E. coli* genome. It gradually developed from a very complex graph, full of tangles, to a single string. Ideally, the genome should have finished as a circular ring, as the *E. coli* chromosome is itself circular, but the sequence did not offer any overlap at the ends to ensure the circularity of the de Bruijn graph. Interestingly, the tangles which appear in the simpler graphs (lev-

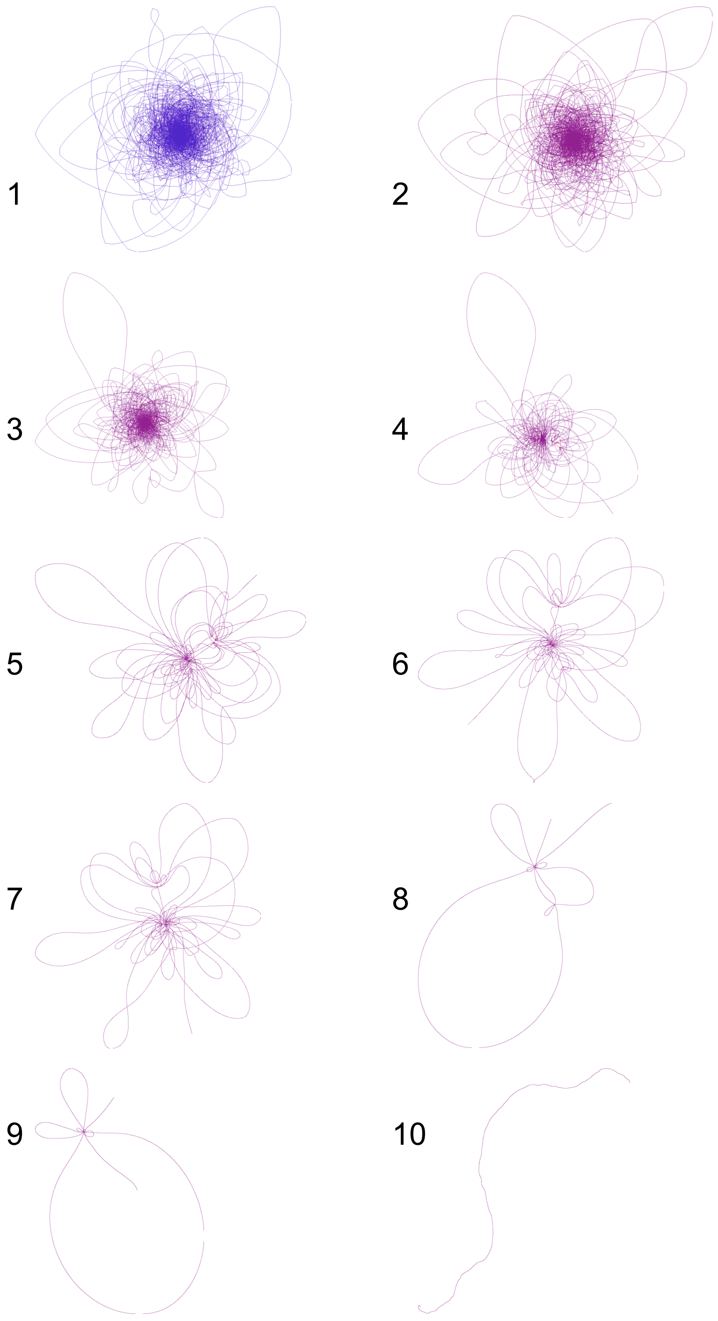


Figure 5.3: Crystallisation of the *E. coli* genome

Topology of the de Bruijn graph at different stages (arbitrary unit cutoffs) of the crystallisation process.

els 9 to 7) correspond to biologically annotated features such as ribosomal genes and transposable elements. The crystallisation process could be improved in several ways. A more appropriate importance metric could ensure the detection of more biologically relevant features. From an implementation point of view, a more stochastic search, as done in the SA algorithm, would make the system more robust to random connections.

Scaled visualisation of structural variants

In order to explore the graphs produced by the above crystallisation process, a de Bruijn graph browser was implemented, following the analogy to interactive internet maps. The extraction script described in 5.1.1 was integrated into an interactive pipeline. Using a standard internet browser, the user specifies a region of interest and an image is sent back. The user can inspect the image and click directly on the nodes to request either different views or the corresponding sequence annotation available on the Ensembl genome browser (Hubbard et al., 2009).

This graph browser is currently functional and new images are generated within several seconds. The web client/server architecture was chosen because the library of scaled de Bruijn graphs can represent very large files, which would be costly to copy over to different locations. The storage of de Bruijn graphs in dedicated databases is currently being implemented (Rasko Leinonen, Siamak Sobhany, Guy Cochrane - pers. comm.).

5.2 Future directions

5.2.1 Automatic identification of structural variants

Beyond the creation of a human interface, the layered approach to de Bruijn graphs described above could also be used for computational analysis. For example, a program can easily scan a graph to detect clear-cut features such as the one in Figure 5.1 even without prior knowledge of the locus. However, to obtain such a result the parameters of the graph had to be adjusted (namely the *k*-mer length had to be increased to 31bp) to remove breakpoints within the inverted block.

Moreover, once the breakpoints between two sequences have been found, analysing the patterns and explaining the evolution between two genomes in terms of independent rearrangements can be very difficult. As mentioned above, efficient algorithms were developed but for specific scenarios, such as inversion distances on simple genomes (Hannenhalli and Pevzner, 1999) or perfectly duplicated genomes (Alekseyev and Pevzner, 2007). It would therefore be useful to develop heuristics to approach such a problem on experimental sequencing data, where all kinds of SVs are to be expected. Crystallisation offers a scaled exploration of the graph, progressively removing fine details so that large features are themselves clearly apparent, before being removed themselves. This heuristic approach could therefore identify individual rearrangements, which would then need to be ordered in time.

5.2.2 *De novo* assembly of transcriptomes

An extension of the *de novo* genome assembly problem is *de novo* transcriptome assembly. It differs mainly in two respects. Because the initial concentrations of mRNA are associated to different expression levels, it is experimentally impossible to ensure a uniform coverage over the complete set of transcripts. Also, in eukaryotic genomes, alternative splicing (AS) events create combinations of sequences which have to be processed simultaneously. This is especially difficult in the case of microreads, where most reads are shorter than the exons they correspond to.

The problem of unequal concentration can be avoided with the help of prior knowledge. For example, if an annotated genome is available, then it is possible to map the reads onto the contigs and features such as coverage (Mortazavi et al., 2008) or paired-end connections (Lacroix et al., 2008a). However, in the case of a *de novo* assembly, this aid is not available.

De Bruijn graphs can be useful to handle the diversity of alternative splicing events. In fact, the field of transcriptomics is already used to analysing splicing graphs, as designed by Pevzner *et al*, with many similarities with the de Bruijn graph (Steffen Heber et al., 2002).

Because of the great interest in these phenomena, researchers have attempted to assemble *de novo* transcriptomes using microreads, sometimes using Velvet. For example, Wakaguri et al. (2008) analysed EST data from insects and Collins et al. (2008) used Velvet on plant (*Pachycladon enysii*) RNA. Trapnell et al. (2008) compared such assemblies with spliced realignments of individual reads. They found that their method was more sensitive, postulating that Velvet probably broke up contigs at alternative splice sites.

The first stages of Velvet, namely construction and error correction, do not use any notion of uniform coverage, except at the final user controlled coverage cutoff step, so this use is justified. However, all of the repeat resolution methods which were developed (cf. Chapter 4) are based on coverage uniformity and therefore cannot be used directly in these experiments. In collaboration with Marcel Schulz, from the Max Planck Institute for Molecular Genetics, solutions for this particular problem have just been designed. For example, the use of paired-end reads to detect clusters of nodes could help isolate the sub-graphs corresponding to single genes and simplify the analysis. This work is still in progress, but, if successful, would fill a significant gap in the array of tools available for transcriptome analysis.

Conclusions

The arrival of ”second-generation” sequencing technologies has therefore brought along many changes in the way laboratories attempt DNA based assays, from genome sequencing to transcriptome analysis or the study of epigenetic features. Within a few years, these techniques established themselves as standards.

These disruptive technologies forced all the steps in the sequencing pipeline to be continuously adapted to rapidly evolving conditions. New protocols such as ChIP-seq (Johnson et al., 2007) or RNA-seq (Mortazavi et al., 2008) extended the application spectrum of sequencing. Sample preparation techniques were developed to ensure less cloning bias (Kozarewa et al., 2009) or introduce new features such as paired-end reads, multiplexing (Meyer et al., 2007) and the capture of specific regions (Olson, 2007). The performance of the sequencing machines themselves improved exponentially, through breakthroughs in chemistry and processes. Sequencing centers have had to address storage issues of a new order with huge investments and careful planning (Doctorow, 2008). Quality control metrics have been gradually updated (Erlich et al., 2008; Quinlan et al., 2008) (Tim Massingham, pers. comm.). Numerous re-mapping (Li et al., 2008a,b; Lin et al., 2008; Prüfer et al., 2008; Bona et al., 2008; Jiang and Wong, 2008; Ondov et al., 2008; Langmead et al., 2009) (as well as Brudno *et al* - compbio.cs.toronto.edu/shrimp/; Strömberg *at al* - bioinformatics.bc.edu/marthlab/Mosaik) and peak-calling (Fejes et al., 2008; Boyle et al., 2008) (as well as Steven Wilder - pers. comm.) algorithms were specifically developed.

Although slower to evolve because of technical difficulties, *de novo* genome assembly also went through profound changes. Within a matter of months, assembly software has shifted from an overlap centric approach (Batzoglou et al., 2002; Mullikin and Ning, 2003; Hernandez et al., 2008) to a search based methods (Warren et al., 2007; Jeck et al., 2007; Dohm et al., 2007; Bryant Jr. et al., 2009) and generally converged towards a de Bruijn graph approach (Chaisson et al., 2009; Chen and Skiena, 2007; Zerbino and Birney, 2008; Butler et al., 2008). As the algorithms became more mature, their implementation was perfected, through parallelization (Simpson et al., 2009; Jackson et al., 2009).

The Velvet project was surprisingly timely. As I started working on de Bruijn graphs, demonstration microread datasets were hardly available. At the time of writing this thesis, their widespread use has become all but routine for many laboratories. Worldwide projects such as the 1000 Genomes (www.1000genomes.org) and the 1001 Genomes (1001genomes.org) projects now use next-generation technologies at an industrial scale.

I have had the privilege of receiving a lot of useful feedback from Velvet users. 250 readers from dozens of academic institutions as well as several biotech companies are currently registered to the Velvet mailing list, regularly exchanging views on next-generation sequencing and *de novo* assembly. These numerous interactions have enriched the project by helping defining the priorities and requirements of the end-user. Although technically I was not working in a wetlab environment, much of the project was directed by experimental constraints or requirements. Velvet has come to be used in many groups and draw attention (MacLean et al., 2009; Appels, 2009; Reinhardt et al., 2009; Chaisson et al., 2009; Denoeud et al., 2008; Nusbaum et al., 2009; Farrer et al., 2009; Wakaguri et al., 2008; Almeida et al., 2009; Ossowski et al., 2008; Miller et al., 2008; Cronn et al., 2008; Paten et al., 2008; Li et al., 2008a; Lacroix et al., 2008b; Bozdag et al., 2008; Quitzau and Stoye, 2008; Salzberg et al., 2008; Wang et al., 2008; Schloss, 2008; Shendure and Ji, 2008; Bona et al., 2008; Graveley, 2008)

There are still a few directions we could have wished to push the project, for example metagenomics or transcriptome analysis. Although some teams have tried using Velvet in such extreme scenarios (Collins et al., 2008; Wakaguri et al., 2008) it is clear that only specifically designed methods will produce robust and reliable results. Velvet, like most genomic assemblers, cannot be used to the fullest in these cases.

However, the field still has a lot of momentum. "Third generation" sequencing machines are soon going to be operational, boasting much longer reads (Eid et al., 2008; Clarke et al., 2009). The paradigms of genome assembly are likely to revert back to the overlap based techniques, which are more efficient with long reads (Miller et al., 2008) and are more easily parallelized on commodity hardware (Mullikin and Ning, 2003).

Nonetheless, algorithms for microread still have their use. In practice, it will take years to replace the current sequencing machines. More fundamentally, there are likely to be applications where sequence coverage will be more important than read-length. For example, in metagenomics or transcriptomics, deeper sequencing provides a finer image of variation within a population. Given sufficient coverage, paired-end reads of a given insert length resolve practically as many repeats as reads of the same length. However, many questions remain to be resolved before microreads can be used for the *de novo* analysis of these datasets.

Appendix A

Publications

- Zerbino DR, Birney, E (2008) Velvet: algorithms for short read *de novo* assembly using de Bruijn graphs. *Genome Research* **18**:821-829.
- Zerbino DR, McEwen G, Margulies, E, Birney E Pebble and Rock Band: Heuristic resolution of repeats and scaffolding in the Velvet short-read *de novo* assembler. *submitted*

Appendix B

Velvet Manual - version 0.7

B.1 For impatient people

```
> make
> ./velveth
> ./velvetg

> ./velveth sillyDirectory 21 -shortPaired data/test_reads.fa
> ./velvetg sillyDirectory
(Final graph has 16 nodes and n50 of 24184 max 44966)
> less sillyDirectory/stats.txt

> ./velvetg sillyDirectory -cov_cutoff 5 -read_trkg yes -amos_file yes
(Final graph has 1 nodes and n50 of 99975 max 99975)
> less sillyDirectory/velvet_asm.afg

> ./velvetg sillyDirectory -exp_cov 19 -ins_length 100
(Final graph has 12 nodes and n50 of 99975 max 99975)

> ./velveth sillyDirectory 21 -short data/test_reads.fa -long data/
test_long.fa
> ./velvetg sillyDirectory -exp_cov 19
```

```
(Final graph has 2 nodes and n50 of 99893 max 99893)
```

B.2 Installation

Requirements

Velvet should function on any standard 64bit Linux environment with gcc. A good amount of physical memory (12GB to start with, more is no luxury) is recommended.

It can in theory function on a 32bit environment, but such systems have memory limitations which might ultimately be a constraint for assembly.

Compiling instructions

From a GNU environment, simply type:

```
> make
```

Colorspace Velvet

To produce the colorspace version of Velvet, compile with the instruction:

```
> make color
```

All the rest of the manual remains valid, except that the executables are now called velveth_de and velveth_de .

Beware that color- and sequence space are incompatible, hence separate sets of executables. In other words, don't try to hash sequence files with colorspace velvet or vice-versa, under penalty of meaningless results!

B.3 Running instructions

Running velveth

Velveth helps you construct the dataset for the following program, velvetg, and indicate to the system what each sequence file represents.

If, on the command line, you forget the syntax, you can print out a short help message:

```
> ./velveth
```

Velveth takes in a number of sequence files, produces a hashtable, then outputs two files in an output directory (creating it if necessary), Sequences and Roadmaps, which are necessary to velvetg. The syntax is as follows:

```
> ./velveth output_directory hash_length
            [[-file_format][-read_type] filename]
```

The hash length, also known as *k*-mer length, corresponds to the length, in base pairs, of the words being hashed. See B.5 for a detailed explanation of how to choose the hash length.

Supported file formats are:

fasta (default)

fastq

fasta.gz

fastq.gz

eland

gerald

Read categories are:

short (default)

shortPaired

short2 (same as short, but for a separate insert-size library)

shortPaired2 (see above)

long (for Sanger, 454 or even reference sequences)

longPaired

For concision, options are stable. In other words, they are true until contradicted by another operator. This allows you to write as many filenames as you wish without having to re-type identical descriptors. For example:

```
> ./velveth output_directory/ 21 -fasta -short solexa1.fa solexa2.fa
        solexa3.fa -long capillary.fa
```

In this example, all the files are considered to be in FASTA format, only the read category changes. However, the default options are "fasta" and "short", so the previous example can also be written as:

```
> ./velveth output_directory/ 21 solexa*.fa -long capillary.fa
```

Running velvetg

Velvetg is the core of Velvet where the de Bruijn graph is built then manipulated. Note that although velvetg saves some files during the process to avoid useless recalculations, the parameters are **not** saved from one run to the next. Therefore:

```
> ./velvetg output_directory/ -cov_cutoff 4
> ./velvetg output_directory/ -min_contig_lgth 100
```

. . . is different from:

```
> ./velvetg output_directory/ -cov_cutoff 4 -min_contig_lgth 100
```

This means you can freely play around with parameters, without re-doing most of the calculations:

```
> ./velvetg output_directory/ -cov_cutoff 4
> ./velvetg output_directory/ -cov_cutoff 3.8
> ./velvetg output_directory/ -cov_cutoff 7
> ./velvetg output_directory/ -cov_cutoff 10
> ./velvetg output_directory/ -cov_cutoff 2
```

On the other hand, within a single velvetg command, the order of parameters is not important.

Finally, if you have any doubt at the command line, you can obtain a short help message by typing:

```
> ./velvetg
```

Single reads

Initally, you simply run:

```
> ./velvetg output_directory/
```

This will produce a fasta file of contigs and output some statistics. Experience shows that there are many short, low-coverage nodes left over from the intial correction. Determine as you wish a coverage cutoff value (cf. B.5), say 5.2X, then type:

```
> ./velvetg output_directory/ -cov_cutoff 5.2
```

On the other hand, if you want to exclude highly covered data from your assembly (e.g. plasmid, mitochondrial, and chloroplast sequences) you can use a maximum coverage cutoff:

```
> ./velvetg output_directory/ -max_coverage 300
        (... other parameters ...)
```

Adding long reads

Reminder: you must have flagged your long reads as such when running velveth (cf. B.3).

If you have a sufficient coverage of short reads, and any quantity of long reads (obviously the more the coverage and the longer the reads, the better), you can use the long reads to resolve repeats in a greedy fashion.

To do this, Velvet needs to have a reasonable estimate of the expected coverage *in short reads* of unique sequence (see B.5 for a definition of k-mer coverage). The simplest way to obtain this value is simply to observe the distribution of contig coverages (as described in B.5), and see around which value the coverages of nodes seem to cluster (especially the longer nodes in your dataset). Supposing the expected coverage is 19X, them you indicate it with the *exp_cov* marker:

```
> ./velvetg output_directory/ -exp_cov 19 (... other parameters ...)
```

Paired-ends reads

Reminder: you must have flagged your reads as being paired-ends when running velveth (cf. B.3).

To activate the use of read pairs, you must specify two parameters: the *expected* (i.e. average) insert length (or at least a rough estimate), and the expected *short-read k-mer coverage* (see B.5 for more information). If you expect your insert length to be around 400bp, and your coverage to be around 21.3X, you would type:

```
> ./velvetg output_directory/ -ins_length 400 -exp_cov 21.3
        (... other parameters ...)
```

If you happen to have hashed paired long reads and you ordered them as explained in B.4 you can also tell Velvet to use this information for scaffolding by indicating the corresponding insert length (remember that you still need to indicate the short-read k-mer coverage):

```
> ./velvetg output_directory/ -exp_cov 21 -ins_length_long 40000
        (... other parameters ...)
```

Standard deviations This is a more subtle point which you can ignore if you have only one dataset of paired-end reads or if the standard deviation (SD) of the insert lengths is roughly proportional to the expected length (e.g. if the insert-lengths are described as $length \pm p\%$).

Velvet does not use the absolute values of the insert-length SDs, but their relative values. Therefore, you do not need to spend too much time on the estimation of the SDs, as long as you are consistent. You can then enter your own *a priori* SD's. To do so simply indicate them as follows:

```
> ./velvetg output_directory/ -exp_cov 21
                -ins_length 200 -ins_length_sd 20
                -ins_length2 20000 -ins_length2_sd 5000
                -ins_length_long 40000 -ins_length_long_sd 1000
                (... other parameters ...)
```

Controlling Velvet's output

Selecting contigs for output By default, Velvet will print out as many contigs as possible. This has the drawback of potentially flooding the output with lots of unwanted very short contigs, which are hardly useable in a significant way. If you wish, you can request that the contigs in the contigs.fa file be longer than a certain length, say 100bp:

```
> ./velvetg -min_contig_lgth 100 (... other parameters ...)
```

Using read tracking Velvet's read tracking can be turned on with the read-tracking option. This will cost slightly more memory and calculation time, but will have the advantage of producing in the end a more detailed description of the assembly:

```
> ./velvetg output_directory/ -read_trkg yes
        (... other parameters ...)
```

Producing an .afg file If you turn on the read tracking, you might also want to have all the assembly information in one datastructure. For this purpose Velvet can produce AMOS files (cf B.4). Because the .afg files tend to be very large, they are only produced on demand:

```
> ./velvetg output_directory/ -amos_file yes
        (... other parameters ...)
```

Using multiple categories You can be interested in keeping several kinds of short read sets separate. For example, if you have two paired-end experiments, with different insert lengths, mixing the two together would be a loss of information. This is why Velvet allows for the use of 2 short read channels (plus the long reads, which are yet another category).

To do so, you simply need to use the appropriate options when hashing the reads (see B.3). Put the shorter inserts in the first category. Supposing your first readset has an insert length around 400bp and the second one a insert length around 10,000bp, you should type:

```
> ./velvetg output_directory/ -ins_length 400 -ins_length2 10000
        (... other parameters ...)
```

Note: Increasing the amount of categories is possible. It's simply a bit more expensive memory-wise.

Note: In the stats.txt file, you will find all three categories (long, short1 and short2) treated separately.

Advanced parameters: Tour Bus

Caveat Emptor

The following parameters are probably best left untouched. If set unwisely, Velvet's behaviour may be unpredictable.

Nonetheless, some users are curious to control the way in which Tour Bus (cf. B.6) decides whether to merge polymorphisms or not.

Before we go into the actual details, it is worth discussing the pros and cons of bubble smoothing. The original idea is that a few SNPs, in the case of diploid assembly, should not prevent the construction of an overall contig. Detecting them post assembly is just a matter of scanning the assembly files and detecting discrepancies between the consensus sequence and the reads.

On the other hand, if you have two copies of a repeat in a haploid genome, you want to reduce the merging to a minimum, so that later analysis with paired-end reads or long reads may allow you to retrieve both individual copies, instead of just one artificial "consensus" sequence.

Hopefully, these issues will eventually be resolved by further thought and experiment. In the mean time, Velvet allows direct access to these parameters for those who want to play around, or maybe tailor Velvet to specific needs (e.g. multi-strain sequencing).

Maximum branch length Partly for engineering issues and partly to avoid aberrant transformations, there is a limit as to how long two paths must before simplification. By default, it is set a 100bp. This means that Velvet will not merge together two sequences which are sufficiently divergent so as not to have any common *k*-mer over 100bp. If you want to allow greater simplifications, then you can set this length to, say, 200bp:

```
> ./velvetg output_directory/ -max_branch_length 200
        (...other parameters...)
```

Maximum indel count Before comparing two sequences, Velvet compares their lengths, and, if they are too different, aborts the inspection. By default, this parameter is set to 3bp, but you can change it to, for example, 6bp:

```
> ./velvetg output_directory/ -max_indel_count 6
```

```
        (...other parameters...)
```

Maximum divergence rate After aligning the two sequences with a standard dynamic alignment, Velvet compares the number of aligned pairs of nucleotides to the length of the longest of the two sequences. By default, Velvet will not simplify two sequences if they are more than 20% diverged. If you want to change that limit to 33%:

```
> ./velvetg output_directory/ -max_divergence 0.33
        (...other parameters...)
```

Maximum gap count After aligning the two sequences with a standard dynamic alignment, Velvet compares the number of aligned pairs of nucleotides to the length of the longest of the two sequences. By default, Velvet will not simplify to sequences if more than 3bp of the longest sequence are unaligned.

```
> ./velvetg output_directory/ -max_gap_count 5
        (...other parameters...)
```

Advanced parameters: Tour Bus

Minimum read-pair validation Velvet will by default assume that paired end reads are perfectly placed. With experimental data this assumption can be contradicted by occasional mis-pairings or by incorrect mappings of the reads because of errors. To avoid being misled by random noise, and therefore avoid missassemblies, Velvet requires that a connection between two contigs be corroborated by at least 10 mate pairs. If you want to change this cutoff to, say, 20, simply type:

```
> ./velvetg output_directory/ -min_pair_count 20
        (...other parameters...)
```

B.4 File formats

Input sequence files

Velvet works mainly with fasta and fastq formats.

For paired-end reads, the assumption is that each read is next to its mate read. In other words, if the reads are indexed from 0, then reads 0 and 1 are paired, 2 and 3, 4 and 5, etc.

If for some reason you have forward and reverse reads in two different FASTA files but in corresponding order, the bundled Perl script shuffleSequences.pl will merge the two files into one as appropriate. To use it, type:

```
> ./shuffleSequences.pl forward_reads.fa reverse_reads.fa output.fa
```

Concerning read orientation, Velvet expects paired-end reads to come from opposite strands facing each other, as in the traditional Sanger format. If you have paired-end reads produced from circularisation (i.e. from the same strand), it will be necessary to replace the first read in each pair by its reverse complement before running velveth.

Output files

After running Velvet you will find a number of files in the output directory:

The contigs.fa file

This fasta file contains the sequences of the contigs longer than $2k$, where k is the word-length used in velveth. If you have specified a *min_contig_lgth* threshold, then the contigs shorter than that value are omitted.

Note that the length and coverage information provided in the header of each contig should therefore be understood in k-mers and in k-mer coverage (cf. B.5) respectively.

The stats.txt file

This file is a simple tabbed-delimited description of the nodes. The column names are pretty much self-explanatory. Note however that node lengths are given in *k-mers*. To obtain the length in nucleotides of each node you simply need to add $k-1$, where k is the word-length used in velveth.

The *in* and *out* columns correspond to the number of arcs on the 5′ and 3′ ends of the contig respectively.

The coverages in columns *short1_cov*, *short1_Ocov*, *short2_cov*, and *short2_Ocov* are provided in k-mer coverage (B.5).

Also, the difference between **_cov* and **_Ocov* is the way these values are computed. In the first count, slightly divergent sequences are added to the coverage tally. However, in the second, stricter count, only the sequences which map perfectly onto the consensus sequence are taken into account.

The velvet_asm.afg file

This file is mainly designed to be read by the open-source AMOS genome assembly package. Nonetheless, a number of programs are available to transform this kind of file into other assembly file formats (namely ACE, TIGR, Arachne and Celera). See http://amos.sourceforge.net/ for more information.

The file describes all the contigs contained in the contigs.fa file (cf B.4).

If you are overwhelmed by the size of the file, two bundled scripts provided by Simon Gladman can help you out:

- asmbly_splitter.pl breaks down the original .afg file into individual files for each contig,
- snp_view.pl allows you to print out a simple ASCII alignment of reads around a given position on a contig.

The LastGraph file

This file describes in its entirety the graph produced by Velvet, in an idiosyncratic format which evolved with my PhD project. The format of this file is briefly as follows:

- One header line for the graph:

  ```
  $NUMBER_OF_NODES    $NUMBER_OF_SEQUENCES  \\
          $HASH_LENGTH
  ```

- One block for each node:

  ```
  NODE    $NODE_ID    $COV_SHORT1   $O_COV_SHORT1  \\
          $COV_SHORT2  $O_COV_SHORT2
  $ENDS_OF_KMERS_OF_NODE
  $ENDS_OF_KMERS_OF_TWIN_NODE
  ```

  Note that the ends of k-mers correspond to the last nucleotides of the k-mers in the node. This means that the two sequences given above are not reverse-complements of each other but reverse complements shifted by k nucleotides. The common length of these sequences is equal to the length of the corresponding contig *minus* $k - 1$.

  See B.4 for an explanation of O_COV values.

- One line for each arc:

  ```
  ARC $START_NODE    $END_NODE    $MULTIPLICITY
  ```

  Note: this one line implicitly represents an arc from node A to B and another, with same multiplicity, from -B to -A.

- For each long sequence, a block containing its path:

  ```
  SEQ $SEQ_ID
  $NODE_ID $OFFSET_FROM_START $START_COORD $END_COORD
  $OFFSET_FROM_END
  $NODE_ID2 etc.
  ```

The offset variables are distances from the edges of the nodes whereas the start and end coordinates are correspond to coordinates within the read sequence.

- If short reads are tracked, for every node a block of read identifiers:

```
NR $NODE_ID $NUMBER_OF_SHORT_READS
$READ_ID $OFFSET_FROM_START_OF_NODE \\
        $START_COORD
$READ_ID2 etc.
```

B.5 Practical considerations / Frequently asked questions

Choice of hash length k

The hash length is the length of the k-mers being entered in the hash table.

Firstly, you must observe three technical constraints:

- it must be an odd number, to avoid palindromes. If you put in an even number, Velvet will just decrement it and proceed.
- it must be below or equal to 31, because it is stored on 64 bits
- it must be strictly inferior to read length, otherwise you simply will not observe any overlaps between reads, for obvious reasons.

Now you still have quite a lot of possibilities. As is often the case, it's a trade-off between specificity and sensitivity. Longer kmers bring you more specificity (i.e. less spurious overlaps) but lowers coverage (cf. below)... so there's a sweet spot to be found with time and experience.

We like to think in terms of "k-mer coverage", i.e. how many times has a k-mer been seen among the reads. The relation between k-mer coverage C_k and standard (nucleotide-wise) coverage C is $C_k = C * (L - k + 1)/L$ where k is your hash length, and L you read length.

Experience shows that this kmer coverage should be above 10 to start getting decent results. If C_k is above 20, you might be "wasting" coverage. Experience also shows that empirical tests with different values for k are not that costly to run!

Choice of a coverage cutoff

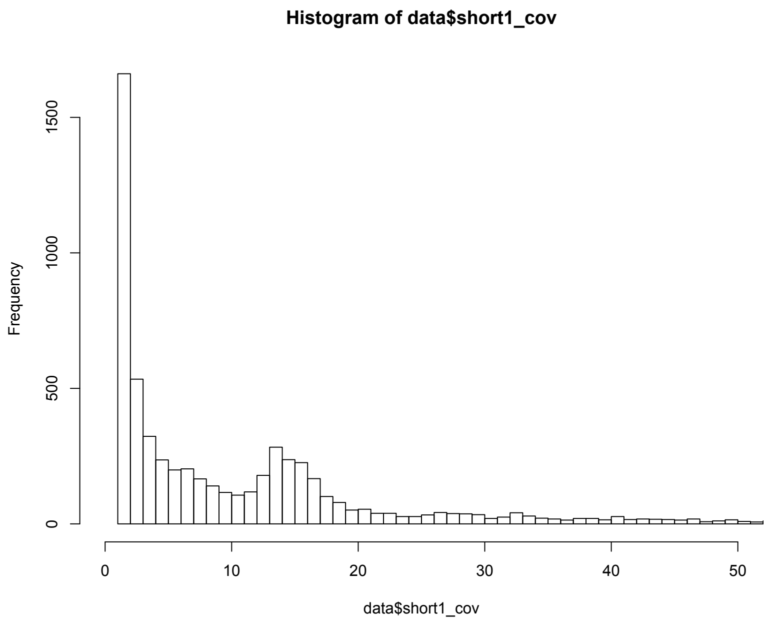
Velvet was designed to be explicitly cautious when correcting the assembly, to lose as little information as possible. This consequently will leave some obvious errors lying behind after the Tour Bus algorithm (cf. B.6) was run. To detect them, you can plot out the distribution of k-mer coverages (B.5), using plotting software (I use R).

The examples below are produced using the S. suis P1/7 data available from the Sanger Institute (www.sanger.ac.uk/Projects/S_suis/) (Note: a simple script is necessary to convert the sequence files to FastA). I used a k-mer length of 21 and no cutoff.

With the R instruction:

```
(R) > data = read.table("stats.txt", header=TRUE)
(R) > hist(data$short1_cov, xlim=range(0,50), breaks=1000000)
```

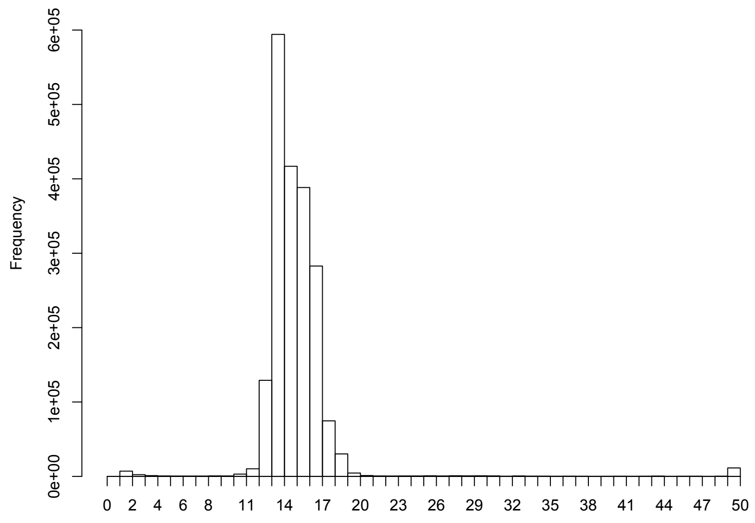
. . . you can obtain:



However, if you weight the results with the node lengths (you need to install the *plotrix* package for R to do this):

```
(R) > library(plotrix)
(R) > weighted.hist(data$short1_cov, data$lgth, breaks=0:50)
```

. . . you obtain:



The comparison of these two plots should convince you that below 7 or 8X you find mainly short, low coverage nodes, which are likely to be errors. Set the exact cutoff at your discretion.

However beware that there is such a thing as an over-aggressive cutoff, which could create mis-assemblies, and destroy lots of useful data.

If you have read-pair information, or long reads, it may be profitable to set a low coverage cutoff and to use the supplementary information resolve the more ambiguous cases.

Determining the expected coverage

From the previous weighted histogram it must be pretty clear that the expected coverage of contigs is near 14X.

Visualising contigs and assemblies

This section will be quite vague, as there are a number of solutions currently available, and presumably new ones under development. The following indications are just hints, as I have not done any exhaustive shopping nor benchmarking.

Most assembly viewers require an assembly format, which come in a variety of shapes and colours: ACE, AMOS, CELERA, TIGR, etc. Velvet only ouputs AMOS .afg files, but these can easily be converted with open-source software (amos.sourceforge.net).

What's long and what's short?

Velvet was pretty much designed with micro-reads (e.g. Illumina) as short and short to long reads (e.g. 454 and capillary) as long. Reference sequences can also be thrown in as long.

That being said, there is no necessary distinction between the types of reads. The only constraint is that a short read be shorter than 32kb. The real difference is the amount of data Velvet keeps on each read. Short reads are presumably too short to resolve many repeats, so only a minimal amount of information is kept. On the contrary, long reads are tracked in detail through the graph.

This means that whatever you call your reads, you should be able to obtain the same initial assembly. The differences will appear as you are trying to resolve repeats, as long reads can be followed through the graph. On the other hand, long reads cost more memory. It is therefore perfectly fine to store Sanger reads as "short" if necessary.

B.6 For more information

Publication: For more information on the theory behind Velvet, you can turn to:

> D.R. Zerbino and E. Birney. 2008. Velvet: algorithms for *de novo* short read assembly using de Bruijn graphs. *Genome Research*, **18**: 821-829

Please use the above reference when citing Velvet.

Webpage: For general information and FAQ, you can first take a look at www.ebi.ac.uk/∼zerbino/velvet.

Mailing list: For questions/requests/etc. you can subscribe to the users' mailing list: velvet-users@ebi.ac.uk.

To do so, see listserver.ebi.ac.uk/mailman/listinfo/velvet-users .

Contact emails: For specific questions/requests you can contact us at the following addresses:

- Daniel Zerbino <zerbino@ebi.ac.uk>
- Ewan Birney: <birney@ebi.ac.uk>

Reporting bugs: We are very grateful to all the people who send us bugs. However, to speed up the process and avoid useless delays, please:

1. ensure that you have the very last version of Velvet, to the last digit, as displayed on the website.
2. attach to your e-mail the Log file from within the Velvet directory.
3. if the program crashed and created a core dump file could you please:
   (a) destroy the core.* file
   (b) recompile Velvet with the instruction "make debug"
   (c) re-run Velvet and let it crash (therefore creating a new core file)
   (d) launch the GNU debugger with the instructions:
   ```
   > gdb ./velvetg core.*
   ```
   (e) within gdb, request a backtrace:

```
(gdb) bt
```

(f) send the listing with the entire gdb session.

Appendix C

Source code

All of the Velvet source code can be downloaded freely under Gnu Public License 2.0 at the following address:

www.ebi.ac.uk/ zerbino/velvet

Velvet was developed in C, and designed for a 64bit Linux-compatible environment, with the Gnu C Compiler (gcc). However, it also compiles and runs on other systems. Currently, Velvet has been tested on Linux 64bit, Mac OS X, and Cygwin platforms, as well as on 32bit environments. Because of Sparc/Solaris compatibility issues, a special Sparc version of Velvet was created (available on demand).

Below is a description of the overall structure of the Velvet source code.

C.1 Global definition files

globals.h	global constants, variable size definition, and custom names.
run.h	list of header files used in the higher level program files, **run.c** and **run2.c**

C.2 Data structures

In line with object oriented programming, the variable and function definitions for each data structure are stored statically in the corresponding ***.c** code file (by convention the first letter of each file is not capitalised), whereas all the callable functions are declared as an API in the corresponding ***.h** header file.

TightString	bit encoded nucleotide sequences
ReadSet	annotated sets of sequences
SplayTable	hash structure used to compute overlaps between sequences
RoadMap	short hand storage of overlaps within a set of reads
PreGraph	reduced version of the de Bruijn graphs, also contains PreNode and PreArc
Graph	de Bruijn graph, also contains Node, Arc, and ShortMarker
PassageMarker	long read marker

C.3 Graph construction functions

preGraphConstruction	Construction of memory efficient graph
concatenatedPreGraph	Concatenation and tip clipping on PreGraph
graphReConstruction	Construction of complete graph from the PreGraph
concatenatedGraph	Concatenation of the Graph

C.4 Graph transformation algorithms

correctedGraph	Tour Bus algorithm
readCoherentGraph	Rock Band algorithm
shortReadPairs	Pebble algorithm

C.5 Miscellaneous

graphStats	Simple routines to output information on the graph
recycleBin	Memory management routine (originally by Guy Slater)
splay	Splay tree structure used within SplayTable
crc	CRC32 hashing function used in SplayTable (originally by Ewan Birney)
fib and dfib	Fibonacci heap structure used in Tour Bus (originally by John-Mark Gurney)

C.6 High-level code

run	velveth: hashing of reads and computation of RoadMap
run2	velvetg: PreGraph then Graph construction, followed by Graph simplification

Bibliography

Adams, M. D., Kelley, J. M., Gocayne, J. D., Dubnick, M., Polymeropoulos, M. H., Xiao, H., Merril, C. R., Wu, A., Olde, B., Moreno, R. F., et al. (1991). Complementary DNA sequencing: expressed sequence tags and human genome project. *Science* **252**:1651–1656.

Alekseyev, M. A. and Pevzner, P. A. (2007). Colored de Bruijn graphs and the genome halving problem. *IEEE/ACM Transactions On Computational Biology And Bioinformatics* **4**:98–107.

Almeida, N. F., Yan, S., Lindeberg, M., Studholme, D. J., Schneider, D. J., Condon, B., Liu, H., Viana, C. J., Warren, A., Evans, C., et al. (2009). A draft genome sequence of *Pseudomonas syringae* pv. tomato t1 reveals a type III effector repertoire significantly divergent from that of *Pseudomonas syringae* pv. tomato DC3000. *Molecular Plant-Microbe Interactions* **22**:52–62.

Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. (1990). Basic local alignment search tool. *Journal of Molecular Biology* **215**:403–410.

Anderson, S. (1981). Shotgun dna sequencing using cloned dnase i-generated fragments. *Nucleic Acids Research* **9**:3015–3027.

Appels, R. (2009). Diversity of genome research at the 2009 plant and animal genome conference. *Functional and Integrative Genomics* **9**:1–6.

Bafna, V. and Pevzner, P. A. (1993). Genome rearrangements and sorting by reversals. In *Proceedings of the 34th Annual Symposium on the Foundations of Computer Science*, pages 148–157.

Batzoglou, S., Jaffe, D. B., Stanley, K., Butler, J., Gnerre, S., Mauceli, E., Berger, B., Mesirov, J. P., and Lander, E. S. (2002). ARACHNE: A whole-genome shotgun assembler. *Genome Research* **12**:177–189.

Benson, G. (1999). Tandem repeats finder: a program to analyze dna sequences. *Nucleic Acids Research* **27**:573–580.

Bentley, D. R. (2006). Whole-genome re-sequencing. *Current Opinion in Genetics and Development* **16**:545–552.

Bentley, S. (2008). Genomic 'valleys of death'. *Nature Reviews in Microbiology* **6**:260–261.

Bignell, G. R., Santarius, T., Pole, J. C., Butler, A. P., Perry, J., Pleasance, E., Greenman, C., Menzies, A., Taylor, S., Edkins, S., et al. (2007). Architectures of somatic genomic rearrangement in human cancer amplicons at sequence-level resolution. *Genome Research* **17**:1296–1303.

Birney, E., Clamp, M., and Durbin, R. (2004). GeneWise and Genomewise. *Genome Research* **14**:988–995.

Blanchette, M., Kent, W. J., Riemer, C., Elnitski, L., Smit, A. F., Roskin, K. M., Baertsch, R., Rosenbloom, K., Clawson, H., Green, E. D., et al. (2004). Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Research* **14**:708–715.

Blattner, F. R., Plunkett, G., Bloch, C. A., Perna, N. T., Burland, V., Riley, M., Collado-Vides, J., Glasner, J. D., Rode, C. K., Mayhew, G. F., et al. (1997). The complete genome sequence of *Escherichia coli* K-12. *Science* **277**:1453–1462.

Bona, F. D., Ossowski, S., Schneeberger, K., and Rätsch, G. (2008). Optimal spliced alignments of short sequence reads. *Bioinformatics* **24**:i174–i180.

Boyle, A. P., Guinney, J., Crawford, G. E., and Furey, T. S. (2008). F-Seq: a feature density estimator for high-throughput sequence tags. *Bioinformatics* **24**:2537–2538.

Bozdag, S., Close, T., and Lonardi, S. (2008). Computing the minimal tiling path from a physical map by integer linear programming. In J. Crandall, KA; Lagergren, ed., *8th International Workshop on Algorithms in Bioinformatics (WABI 2008)*, volume 5251, pages 148–161.

Brown, T. (2007). *Genomes 3*. Garland Science.

Bryant Jr., D. W., Wong, W.-K., and Mockler, T. C. (2009). QSRA - a quality-value guided de novo short read assembler. *BMC Bioinformatics* **In press**.

Butler, J., MacCallum, I., Kleber, M., Shlyakhter, I. A., Belmonte, M. K., Lander, E. S., Nusbaum, C., and Jaffe, D. B. (2008). ALLPATHS: De novo assembly of whole-genome shotgun microreads. *Genome Research* **18**:810–820.

Carninci, P., Kasukawa, T., Katayama, S., Gough, J., Frith, M. C., Maeda, N., Oyama, R., Ravasi, T., Lenhard, B., Wells, C., et al. (2005). The transcriptional landscape of the mammalian genome. *Science* **309**:1559–1563.

Chaisson, M. J., Brinza, D., and Pevzner, P. A. (2009). De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Research* **19**:336–346.

Chaisson, M. J. and Pevzner, P. A. (2008). Short read fragment assembly of bacterial genomes. *Genome Research* **18**:324–330.

Chen, J. and Skiena, S. (2007). *Advances in Genome Sequencing Technology and Algorithms*, chapter Assembly For Double-Ended Short-Read Sequencing Technologies, pages 123–141. Artech House Publishers.

Chevreux, B. (2005). *MIRA: An Automated Genome and EST Assembler*. Ph.D. thesis, The Medical Faculty of Heidelberg of the Ruprecht-Karls-University.

Clarke, J., Wu, H.-C., Jayasinghe, L., Patel, A., Reid, S., and Bayley, H. (2009). Continuous base identification for single-molecule nanopore DNA sequencing. *Nature Nanotechnology* **In press**.

Cole, C. G., McCann, O. T., Oliver, J. E. C. K., Willey, D., Gribble, S. M., Yang, F., McLaren, K., Rogers, J., Ning, Z., Beare, D. M., et al. (2008). Finishing the finished human chromosome 22 sequence. *Genome Biology* **9**:R78.

Cole, S. T., Brosch, R., Parkhill, J., Garnier, T., Churcher, C., Harris, D., Gordon, S. V., Eiglmeier, K., Gas, S., Barry, C. E., et al. (1998). Deciphering the biology of *Mycobacterium tuberculosis* from the complete genome sequence. *Nature* **393**:537–544.

Collins, F. S., Morgan, M., and Patrinos, A. (2003). The Human Genome Project: Lessons from large-scale biology. *Science* **300**:286–290.

Collins, L. J., Biggs, P. J., Voelckel, C., and Joly, S. (2008). An approach to transcriptome analysis of non-model organisms using short-read sequences. *Genome Informatics* **21**:3–14.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2002). *Introduction to Algorithms*. Higher Education. McGraw-Hill.

Cronn, R., Liston, A., Parks, M., Gernandt, D. S., Shen, R., and Mockler, T. (2008). Multiplex sequencing of plant chloroplast genomes using solexa sequencing-by-synthesis technology. *Nucleic Acids Research* **36**:e122.

Denoeud, F., Aury, J.-M., Da Silva, C., Noel, B., Rogier, O., Delledonne, M., Morgante, M., Valle, G., Wincker, P., Scarpelli, C., et al. (2008). Annotating genomes with massive-scale rna sequencing. *Genome Biology* **9**:R175.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik* **1**:269–271.

Doctorow, C. (2008). Big data: Welcome to the petacentre. *Nature* **455**:16–21.

Dohm, J. C., Lottaz, C., Borodina, T., and Himmelbauer, H. (2007). SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Research* **17**:1697–1706.

Dohm, J. C., Lottaz, C., Borodina, T., and Himmelbauer, H. (2008). Substantial biases in ultra-short read data sets from high-throughput DNA sequencing. *Nucleic Acids Research* **36**:e105.

Dubchak, I., Poliakov, A., Kislyuk, A., and Brudno, M. (2009). Multiple whole genome alignments without a reference organism. *Genome Research* **In press**.

Edwards, A., Voss, H., Rice, P., Civitello, A., Stegemann, J., Schwager, C., Zimmermann, J., Erfle, H., Caskey, T., and Ansorge, W. (1990). Automated DNA sequencing of the human HPRT locus. *Genomics* **6**:593–608.

Ehrlich, J., Sankof, D., and Nadeau, J. H. (1997). Synteny conservation and chromosome rearrangements during mammalian evolutionconservation and chromosome rearrangements during mammalian evolution. *Genetics* **147**:289–296.

Eid, J., Fehr, A., Gray, J., Luong, K., Lyle, J., Otto, G., Peluso, P., Rank, D., Baybayan, P., Bettman, B., et al. (2008). Real-time dna sequencing from single polymerase molecules. *Science* **323**:133–138.

Eric S. Lander, M. W. (1988). entated on the genome. genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics* **2**:231–239.

Erlich, Y., Mitra, P. P., delaBastide, M., McCombie, R., and Hannon, G. J. (2008). Alta-Cyclic: a self-optimizing base caller for next generation sequencing. *Nature Methods* **5**:679–682.

Ewing, B., Hillier, L., Wendl, M. C., and Green, P. (1998). Base-calling of automated sequencer traces using Phred. I. accuracy assessment. *Genome Research* **8**:175–185.

Fagin, B., Watt, J., and Gross, R. (1993). A special-purpose processor for gene sequence analysis. *Bioinformatics* **9**:221–226.

Farrer, R. A., Kemen, E., Jones, J. D., and Studholme, D. J. (2009). *De novo* assembly of the *Pseudomonas syringae pv. syringae* b728a genome using illumina/solexa short sequence reads. *FEMS Microbiology Letters* **291**:103–111.

Fasulo, D., Halpern, A., Dew, I., and Mobarry, C. (2002). Efficiently detecting polymorphisms durring the fragment assembly process. *Bioinformatics* **18**:S294–S302.

Fejes, A. P., Robertson, G., Bilenky, M., andMatthew Bainbridge, R. V., and Jones, S. J. M. (2008). FindPeaks 3.1: a tool for identifying areas of enrichment from massively parallel short-read sequencing technology. *Bioinformatics* **24**:1729–1730.

Gansner, E. R. and North, S. C. (1999). An open graph visualization system and its applications. *Software - Practice and Experience* **30**:1203–1233.

Gardner, M. J., Hall, N., Fung, E., White, O., Berriman, M., Hyman, R. W., Carlton, J. M., Pain, A., Nelson, K. E., Bowman, S., et al. (2002). Genome sequence of the human malaria parasite *Plasmodium falciparum*. *Nature* **419**:498–511.

Giegerich, R. and Kurtz, S. (1997). From Ukkonen to McCreight and Weiner: A unifying view of linear-time suffix tree construction. *Algorithmica* **19**:331–353.

Graveley, B. R. (2008). Molecular biology - power sequencing. *Nature* **453**:1197–1198.

Guttmacher, A. E. and Collins, F. S. (2003). Welcome to the genomic era. *The New England Journal of Medicine* **349**:996–998.

Hannenhalli, S. and Pevzner, P. A. (1999). Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *Journal of the Association for Computing Machinery* **46**:1–27.

Harris, T. D., Buzby, P. R., Babcock, H., Beer, E., Bowers, J., Braslavsky, I., Causey, M., Colonell, J., DiMeo, J., Efcavitch, J. W., et al. (2008). Single-molecule DNA sequencing of a viral genome. *Science* **320**:106–109.

Havlak, P., Chen, R., Durbin, K. J., Amy Egan, Y. R., Song, X.-Z., Weinstock, G. M., and Gibbs, R. A. (2004). The Atlas genome assembly system. *Genome Research* **14**:721–732.

Hernandez, D., François, P., Farinelli, L., Østerås, M., and Schrenzel, J. (2008). De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. *Genome Research* **18**:802–809.

Higgins, D. and Sharp, P. (1988). Clustal: a package for performing multiple sequence alignment on a microcomputer. *Gene* **73**:237–244.

Hillier, L. W., Marth, G. T., Quinlan, A. R., Dooling, D., Fewell, G., Barnett, D., Fox, P., Glasscock, J. I., Hickenbotham, M., Huang, W., et al. (2008). Whole-genome sequencing and variant discovery in *C. elegans*. *Nature Methods* **5**:183–188.

Hirschhorn, J. N. and Daly, M. J. (2005). Genome-wide association studies for common diseases and complex traits. *Nature Reviews Genetics* **6**:95–108.

Huang, X., Wang, J., Aluru, S., Yang, S.-P., and Hillier, L. (2003). PCAP: A whole-genome assembly program. *Genome Research* **13**:2164–2170.

Hubbard, T. J. P., Aken, B. L., Ayling, S., Ballester, B., Beal, K., Bragin, E., Brent, S., Chen, Y., Clapham, P., Clarke, L., et al. (2009). Ensembl 2009. *Nucleic Acids Research* **37**:D690–D697.

Idury, R. M. and Waterman, M. S. (1995). A new algorithm for DNA sequence assembly. *Journal of Computational Biology* **2**:291–306.

Jackson, B. G., Schnable, P. S., and Aluru, S. (2009). Parallel short sequence assembly of transcriptomes. *BMC Bioinformatics* **10**:S14.

Jeck, W. R., Reinhardt, J. A., Baltrus, D. A., Hickenbotham, M. T., Magrini, V., Mardis, E. R., Dangl, J. L., and Jones, C. D. (2007). Extending assembly of short DNA sequences to handle error. *Bioinformatics* **23**:2942–2944.

Jiang, H. and Wong, W. H. (2008). SeqMap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics* **24**:2395–2396.

Johnson, D. S., Mortazavi, A., Myers, R. M., and Wold, B. (2007). Genome-wide mapping of in vivo protein-DNA interactions. *Science* **316**:1497–1502.

Kececioglu, J. and Sankoff, D. (1995). Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica* **13**:180–210.

Kidd, J. M., Cooper, G. M., Donahue, W. F., Hayden, H. S., Sampas, N., Graves, T., Hansen, N., Teague, B., Alkan, C., Antonacci, F., et al. (2008). Mapping and sequencing of structural variation from eight human genomes. *Nature* **453**:56–64.

Kidwell, M. G. (2005). *The Evolution of the Genome*, chapter Transposable elements. Elsevier/Academic Press.

Kim, J., Bhinge, A. A., Morgan, X. C., and Iyer, V. R. (2005). Mapping DNA-protein interactions in large genomes by sequence tag analysis of genomic enrichment. *Nature Methods* **2**:47–53.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science* **220**:671–680.

Klein, C. A. and Hölzel, D. (2006). Systemic cancer progression and tumor dormancy. *Cell Cycle* **5**:1788–1798.

Kozarewa, I., Ning, Z., Quail, M. A., Sanders, M. J., Berriman, M., and Turner, D. J. (2009). Amplification-free illumina sequencing-library preparation facilitates improved mapping and assembly of (g+c)-biased genomes. *Nature Methods* **In press**.

Lacroix, V., Sammeth, M., Guigo, R., and Bergeron, A. (2008a). Exact transcriptome reconstruction from short sequence reads. In *Proceedings of the 8th international workshop on Algorithms in Bioinformatics*, pages 50–63. Springer-Verlag.

Lacroix, V., Sammeth, M., Guigo, R., and Bergeron, A. (2008b). Exact transcriptome reconstruction from short sequence reads. In *8th International Workshop on Algorithms in Bioinformatics (WABI 2008)*, pages 50–63.

Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. L. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology* **10**:R25.

Lasken, R. S. (2007). Single-cell genomic sequencing using multiple displacement amplification. *Current Opinion in Microbiology* **10**:510–516.

Lee, S., Cheran, E., and Brudno, M. (2008). A robust framework for detecting structural variations in a genome. *Bioinformatics* **24**:i59–i67.

Levy, S., Sutton, G., Ng, P. C., Feuk, L., Halpern, A. L., Walenz, B. P., Axelrod, N., Huang, J., Kirkness, E. F., Denisov, G., et al. (2007). The diploid genome sequence of an individual human. *PLoS Biology* **5**:e254.

Li, H., Ruan, J., and Durbin, R. (2008a). Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research* **18**:1851–1858.

Li, R., Li, Y., Kristiansen, K., and Wang, J. (2008b). SOAP: short oligonucleotide alignment program. *Bioinformatics* **24**:713–714.

Lin, H., Zhang, Z., Zhang, M. Q., Ma, B., and Li, M. (2008). Zoom! zillions of oligos mapped. *Bioinformatics* **24**:2431–2437.

Ma, J., Ratan, A., Raney, B. J., Suh, B. B., Miller, W., and Haussler, D. (2008). The infinite sites model of genome evolution. *Proc. Nati. Acad. Sci. USA* **105**:14254–14261.

Maaß, M. G. (2008). Linear bidirectional on-line construction of affix trees. *Algorithmica* **37**:43–74.

MacLean, D., Jones, J. D. G., and Studholme, D. J. (2009). Application of 'next-generation' sequencing technologies to microbial genetics. *Nature Reviews Microbiology* **7**:287–296.

Mardis, E. R. (2008). The impact of next-generation sequencing technology on genetics. *Trends in Genetics* **24**:133–141.

Margulies, M., , Egholm, M., , Altman, W. E., Attiya, S., Bader, J. S., Bemben, L. A., Berka, J., Braverman, M. S., et al. (2005). Genome sequencing in microfabricated high-density picolitre reactors. *Nature* **437**:376–380.

Matsuzaki, H., Dong, S., Loi, H., Di, X., Liu, G., Hubbell, E., Law, J., Berntsen, T., Chadha, M., Hui, H., et al. (2004). Genotyping over 100,000 SNPs on a pair of oligonucleotide arrays. *Nature Methods* **1**:109–111.

Medvedev, P. and Brudno, M. (2008). *Research in Computational Molecular Biology*, chapter Ab Initio Whole Genome Shotgun Assembly With Mated Short Reads. Springer.

Medvedev, P., Georgiou, K., Myers, G., and Brudno, M. (2007). Computability of models for sequence assembly. In *Proceedings of Workshop on Algorithms in Bioinformatics (WABI)*, pages 289–301.

Meldrum, D. R. and Holl, M. R. (2002). Microscale bioanalytical systems. *Science* **297**:1197–1198.

Metzker, M. L. (2005). Emerging technologies in DNA sequencing. *Genome Research* **15**:1767–1776.

Meyer, M., Stenzel, U., Myles, S., Prüfer, K., and Hofreiter, M. (2007). Targeted high-throughput sequencing of tagged nucleic acid samples. *Nucleic Acids Research* **35**:e97.

Miller, J. R., Delcher, A. L., Koren, S., Venter, E., Walenz, B. P., Brownley, A., Johnson, J., Li, K., Mobarry, C., and Sutton, G. (2008). Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics* **24**:2818–2824.

Mortazavi, A., Williams, B. A., McCue, K., Schaeffer, L., and Wold, B. (2008). Mapping and quantifying mammalian transcriptomes by RNA-Seq. *Nature Methods* **5**:621–628.

Mott, R. (1998). Trace alignment and some of its applications. *Bioinformatics* **14**:92–97.

Mullikin, J. C. and Ning, Z. (2003). The Phusion assembler. *Genome Research* **13**:81–90.

Myers, E. W. (1995). Towards simplifying and accurately formulating fragment assembly. *Journal of Computational Biology* **2**.

Myers, E. W. (2005). The fragment assembly string graph. *Bioinformatics* **21**:ii79–ii85.

Myers, E. W., Sutton, G. G., Delcher, A. L., Dew, I. M., Fasulo, D. P., Flanigan, M. J., Kravitz, S. A., Mobarry, C. M., Reinert, K. H. J., Remington, K. A., et al. (2000). A whole-genome assembly of drosophila. *Science* **287**:2196–2204.

Needleman, S. and Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* **48**:443–453.

Nene, V., Wortman, J. R., Lawson, D., Haas, B., Kodira, C., Tu, Z. J., Loftus, B., Xi, Z., Megy, K., Grabherr, M., et al. (2007). Genome sequence of *Aedes aegypti*, a major Arbovirus vector. *Science* **316**:1718–1723.

Nusbaum, C., Ohsumi, T. K., Gomez, J., Aquadro, J., Victor, T. C., Warren, R. M., Hung, D. T., Birren, B. W., Lander, E. S., and Jaffe, D. B. (2009). Sensitive, specific polymorphism discovery in bacteria using massively parallel sequencing. *Nature Methods* **6**:67–69.

Olson, M. (2007). Enrichment of super-sized resequencing targets from the human genome. *Nature Methods* **4**:891–892.

Ondov, B. D., Varadarajan, A., Passalacqua, K. D., and Bergman, N. H. (2008). Efficient mapping of Applied Biosystems SOLiD sequence data to a reference genome for functional genomic applications. *Bioinformatics* **24**:2776–2777.

Ossowski, S., Schneeberger, K., and Clark, R. M. (2008). Sequencing of natural strains of *Arabidopsis thaliana* with short reads. *Genome Research* **18**:2024–2033.

Palmer, J. D., Nugent, J. M., and Herbon, L. A. (1987). Unusual structure of geranium chloroplast DNA: A triple-sized inverted repeat, extensive gene duplications, multiple inversions, and two repeat families. *Proc. Nati. Acad. Sci. USA* **84**:769–773.

Parkhill, J., Sebaihia, M., Preston, A., Murphy, L. D., Thomson, N., Harris, D. E., Holden, M. T. G., Churcher, C. M., Bentley, S. D., Mungall, K. L., et al. (2003). Comparative analysis of the genome sequences of *Bordetella pertussis*, *Bordetella parapertussis* and *Bordetella bronchiseptica*. *Nature Genetics* **35**:32–40.

Paten, B., Herrero, J., Beal, K., Fitzgerald, S., and Birney, E. (2008). Enredo and Pecan: Genome-wide mammalian consistency-based multiple alignment with paralogs. *Genome Research* **18**:1814–1828.

Pearson, W. and D.J.Lipman (1988). Improved tools for biological sequence comparison. *Proc. Nati. Acad. Sci. USA* **4**:2444–2448.

Pevzner, P. A. and Tang, H. (2001). Fragment assembly with double-barreled data. *Bioinformatics* **17**:S225–S233.

Pevzner, P. A., Tang, H., and Waterman, M. S. (2001). An eulerian path approach to DNA fragment assembly. *Proc. Nati. Acad. Sci. USA* **98**.

Phillippy, A. M., Schatz, M. C., and Pop, M. (2008). Genome assembly forensics: finding the elusive mis-assembly. *Genome Biology* **9**:R55.

Pop, M., Kosack, D. S., and Salzberg, S. L. (2004). Hierarchical scaffolding with bambus. *Genome Research* **14**:149–159.

Prüfer, K., Stenzel, U., Dannemann, M., Green, R. E., Lachmann, M., and Kelso, J. (2008). PatMaN: rapid alignment of short sequences to large databases. *Bioinformatics* **24**:1530–1532.

Quinlan, A. R., Stewart, D. A., Strömberg, M. P., and Marth, G. T. (2008). Pyrobayes: and improved base caller for SNP discovery in pyrosequencing. *Nature Methods* **5**:179–181.

Quitzau, J. and Stoye, J. (2008). Detecting repeat families in incompletely sequenced genomes. In *8th International Workshop on Algorithms in Bioinformatics (WABI 2008)*, pages 342–353.

Raphael, B., Zhi, D., Tang, H., and Pevzner, P. (2004). A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Research* **14**:2336–2346.

Rasmussen, K., Stoye, J., and Myers, E. (2005). Efficient q-gram filters for finding all-matches over a given length. In *9th Conf. on Computational Molecular Biology*, pages 189–203.

Reinhardt, J. A., Baltrus, D. A., Nishimura, M. T., Jeck, W. R., Jones, C. D., and Dangl, J. L. (2009). De novo assembly using low-coverage short read sequence data from the rice pathogen Pseudomonas syringae pv. oryzae. *Genome Research* **19**:294–305.

Roach, J. C., Boysen, C., Wang, K., and Hood, L. (1995). Pairwise end sequencing: A unified approach to genomic mapping and sequencing. *Genomics* **26**:345–353.

Ronaghi, M., Karamohamed, S., Pettersson, B., Uhlén, M., and Nyrén, P. (1996). Real-time DNA sequencing using detection of pyrophosphate release. *Analytical Biochemistry* **242**:84–89.

Saerens, M., Fouss, F., Yen, L., and Dupont, P. (2004). The principal components analysis of a graph, and its relationships to spectral clustering. In S. B. . Heidelberg, ed., *Machine Learning: ECML 2004*, volume 3201/2004, pages 371–383.

Salzberg, S. L., Sommer, D. D., Puiu, D., and Lee, V. T. (2008). Gene-boosted assembly of a novel bacterial genome from very short reads. *PLoS Computational Biology* **4**:e100186.

Sanger, F., G.M. Air, B. B., Brown, N., Coulson, A., Fiddes, J., III, C. H., Slocombe, P., and Smith, M. (1977a). Nucleotide sequence of bacteriophage ϕx174 DNA. *Nature* **265**:687–695.

Sanger, F., Nicklen, S., and Coulson, A. R. (1977b). DNA sequencing with chain-terminating inhibitors. *Proc. Nati. Acad. Sci. USA* **74**:5463–5467.

Sankoff, D. and El-Mabrouk, N. (2002). *Current Topics in Computational Molecular Biology*, chapter Genome Rearrangement. MIT Press.

Schloss, J. A. (2008). How to get genomes at one ten-thousandth the cost. *Nature Biotechnology* **26**:1113–1115.

Schwartz, S., Kent, W. J., Smit, A., Zhang, Z., Baertsch, R., Hardison, R. C., Haussler, D., and Miller, W. (2003). Human–mouse alignments with BLASTZ. *Genome Research* **12**:103–107.

Service, R. F. (2006). The race for the $1000 genome. *Science* **311**:1544–1546.

Shendure, J. and Ji, H. (2008). Next-generation DNA sequencing. *Nature Biotechnology* **26**:1135–1145.

Simpson, J. T., Wong, K., Jackman, S. D., Schein, J. E., Jones, S. J., and Birol, I. (2009). ABySS: A parallel assembler for short read sequence data. *Genome Research* **In press**.

Sleator, D. D. and Tarjan, R. E. (1985). Self-adjusting binary search trees. *Journal of the Association for Computing Machinery* **32**:652–686.

Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology* **147**:195–197.

Steffen Heber, M. A., Sze, S.-H., Tang, H., and Pevzner, P. A. (2002). Splicing graphs and EST assembly problem. *Bioinformatics* **18**:S181–S188.

Stibitz, S. (1998). IS*481* and IS*1002* of *Bordetella pertussis* create a 6-base-pair duplication upon insertion at a consensus target site. *Journal of Bacteriology* **180**:4963–4966.

The *C. elegans* Sequencing Consortium (1998). Genome sequence of the nematode *C. elegans*: A platform for investigating biology. *Science* **282**:2012–2018.

The International HapMap Consortium (2003). The International HapMap Project. *Nature* **426**:789–796.

The International Human Genome Sequencing Consortium (2001). Initial sequencing and analysis of the human genome. *Nature* **409**:860–921.

The International Human Genome Sequencing Consortium (2004). Finishing the euchromatic sequence of the human genome. *Nature* **431**:931–945.

The Mouse Genome Sequencing Consortium (2002). Initial sequencing and comparative analysis of the mouse genome. *Nature* **420**:520–562.

Trapnell, C., Pachter, L., and Salzberg, S. L. (2008). TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics* **In press**.

Velculescu, V. E., Zhang, L., Vogelstein, B., and Kinzler, K. W. (1995). Serial analysis of gene expression. *Science* **270**:484–487.

Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., Smith, H. O., Yandell, M., Evans, C. A., Holt, R. A., et al. (2001). The sequence of the human genome. *Science* **291**:1304–1351.

Wakaguri, H., Suzuki, Y., Katayama, T., Kawashima, S., Kibukawa, E., Hiranuka, K., Sasaki, M., Sugano, S., and Watanabe, J. (2008). Full-malaria/parasites and full-arthropods: databases of full-length cDNAs of parasites and arthropods, update 2009. *Nucleic Acids Research* **37**:D520–D525.

Walker, A. and Parkhill, J. (2008). Single-cell genomics. *Nature Reviews in Microbiology* **6**:176–177.

Wang, J., Wang, W., Li, R., Li, Y., Tian, G., Goodman, L., Fan, W., Zhang, J., Li, J., Zhang, J., et al. (2008). The diploid genome sequence of an asian individual. *Nature* **456**:60–65.

Warren, R. L., Sutton, G. G., Jones, S. J. M., and Holt, R. A. (2007). Assembling millions of short DNA sequences using SSAKE. *Bioinformatics* **23**:500–501.

Whiteford, N., Haslam, N., Weber, G., Prügel-Bennett, A., Jonathan W. Essex, P. L. R., Bradley, M., and Neylon, C. (2005). An analysis of the feasibility of short read sequencing. *Nucleic Acids Research* **33**:e171.

Williams, R., Peisajovich, S. G., Miller, O. J., , Magdassi, S., Tawfik, D. S., and Griffiths, A. D. (2006). Amplification of complex gene libraries by emulsion PCR. *Nature Methods* **3**:545–550.

Zerbino, D. R. and Birney, E. (2008). Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research* **18**:821–829.

Zhang, J. (2003). Evolution by gene duplication: an update. *Trends in Ecology and Evolution* **18**:292–298.

Zhang, K., Martiny, A. C., Reppas, N. B., Barry, K. W., Malek, J., Chisholm, S. W., and Church, G. M. (2006). Sequencing genomes from single cells by polymerase cloning. *Nature Biotechnology* **24**:680–686.