Abstract

Next Generation Sequencing (NGS) technologies have a great impact both at economical and at research level, with the increasing of data production and the cost reduction and. This new kind of techniques allow the sequencing of thousands of genomes from humans to microbes and they open entirely new areas of biological inquiry, including the investigation of ancient genomes, of human disease, the characterization of ecological diversity, and the identification of unknown etiological agents. The application field could be divided into three main arguments: genomic tasks (genome assembly, SNPs and structural variations), transcriptome analysis (gene prediction and annotation, alternative splicing discovering) and epigenetic problems. The new technologies also offer challenges in experimental design, data management and analysis. In particular, it is desirable to have analysis keep pace with data production, and thus new bioinformatics tools are being developed.

Three platforms for DNA sequencing read production are in reasonably widespread use: the Roche/454, the Illumina/Solexa Genome Analyzer and the Applied Biosystems SOLiD^{TM} System. The Roche/454 is the first to achieve commercial introduction (in 2005) and it uses an innovative sequencing technology known as pyrosequencing. It produces sequences 300-400 bases, longer than Illumina/Solexa (about 70 bases) and SOLiD/Applied Biosystem (about 50 bases), but with a lower high throughput.

During my Ph.D., the Next Generation Sequencing has become a wide spread practice and the aim of my research is the development of new proper tools. I hoped to create useful programs, that would be able to transform the large amount of produced raw data into useful information for biology tasks in few time. With my research and my algorithms, I collaborated to the development and solution of two of the most challenging and studied applications: genome assembly and gene prediction.

De novo sequencing is the starting point of any possible genetic analysis with the creation of the original genomic sequence. This explain why *de novo* sequencing and genome assembly are very important and studied problems. With Next Generation Sequencing the task has began even more challenging: the reduced time and cost allows to sequence even long organisms. When I started my Ph.D., there were only few programs that were able to perform *de novo* assembly, and among these the most used were Newbler, Velvet and Cabog.

My aim was to improve the current state of the art, developing a new assembly tool that use the strengths and overcome the weaknesses of the cited programs. Moreover, the new program should be able to work with any

kind of data (Next Generation Sequencing and other available evidences), and to produce a well-defined genome assembly. Many efficient assembler have been implemented yet, but quite all of them are able to produce only unconnected fragments (contigs) of the original genome. In many cases, they were not able to realize the final scaffolding: set of well ordered and oriented contigs. Only few of them performs this task, that is useful to move toward the finishing of the assembly. The idea is to work in this direction: the development of a platform that is able to correctly order and orient a set of contigs, connected among them through mate-pairs, into scaffolds. The tool would be able to control the "consistency" of the starting assembly and correct the error of the links, to produce a genome sequence and reduce the background noise. The best strategy is to create contigs using Roche/454 reads and Newbler assembler, and mate-pair reads with Illumina or SOLiD.

Gene prediction is a well studied and known problem. Over the past, a lot of program have been developed (Jigsaw, GeneID, GeneSplicer, Genscan, Glimmer, SNAP, TigrScan, Twinscan,...), and the reached results allow to predict quite all genes with a high specificity and sensibility level. After an accurate analysis, I found that a common weak point of all the programs was the requirement of a starting training set, from which learning the rules of the organism gene structure, used for the future prediction. Unfortunately, very often this set is not available and it is necessary to create a new one, using information coming from similar organism or from other source of evidences (EST, proteins,...) . My idea was to use Next Generation Sequencing data to create a starting set of proper genes, sequencing the transcriptome, aligning the produced reads on the genome sequence and discovering the exons and introns to reconstruct the gene structure.

Sommario

La commercializzazione delle nuove tecnologie di sequenziamento (NGS, Next Generation Sequencing), ha avuto un grande impatto sia a livello economico sia biologico, grazie alla significativa riduzione dei tempri di produzione e dei costi, e all'aumento della quantità di dati ottenuti. Le nuove tecniche di sequenziamento hanno permesso di ricreare il genoma di migliaia di organismi, sia piccoli come i microbi, sia grandi come il genoma umano, aprendo nuove aree di ricerca. Ad esempio, ora è possibile studiare il DNA antico, fare ricerca su malattie genetiche, studiare caratteristiche e differenze evolutive tra organismi,... I nuovi metodi si possono applicare a tre campi principali: genomico (come l'assemblaggio dei genomi, la ricerca di SNPs e variazioni strutturali), trascrittomico (per eseguire la predizione genica, l'annotazione e lo studio di splicing alternativi) ed epigenetico.
I sequenziatori di nuova generazione hanno apportato cambiamenti anche a livello bioinformatico. Infatti, con l'acquisizione di moli di dati sempre più grandi, si è reso necessario affrontare il problema della loro gestione dal punto di vista sia di tempo computazionale per analizzarli sia di memoria richiesta per immagazzinarli. Inoltre, si è resa necessaria l'implementazione di strumenti in grado di elaborare i dati grezzi ottenuti, per trasformali in utili informazioni da applicare nelle analisi biologiche.

Attualmente le tre piattaforme di sequenziamento più utilizzate sono Roche/454 , Illumina/Solexa Genome Analyzer, e Applied Biosystems SOLiD$^{\text{TM}}$. Il primo sequenziatore ad essere commercializzato nel 2005 fu il 454. Si basa su tecniche di sequenziamento innovative (pyrosequencing) ed è in grado di produrre sequenze lunghe 300-400 basi, con una buona qualità media. Tuttavia il 454 non raggiunge i livelli di produzione di altri sequenziatori, come SOLiD ed Illumina, che in poco tempo sono in grado di produrre milioni di sequenze, anche se di dimensioni minori (circa 50 e 70 basi rispettivamente per SOLiD e Illumina).

L'idea del mio dottorato è di applicare le conoscenze matematiche ed informatiche allo studio di nuove tecniche per l'utilizzo dei dati di nuova generazione in problemi biologici. Lo scopo è di sviluppare dei programmi in grado di elaborare grandi quantità di dati in poco tempo. Con la mia ricerca ho collaborato all'implementazione di metodi per la risoluzione di problemi di assemblaggio e di predizione genica.

Il sequenziamento *de novo* e successivamente l'assemblaggio sono un punto fondamentale per l'analisi del genoma di un organismo. Attualmente il problema dell'assemblaggio è un problema ancora aperto e ampiamente studiato: non esistono ancora programmi in grado di ricostruire un genoma completo

partendo da reads prodotte con un sequenziamento di nuova generazione. Esistono software come Newbler, Velvet e Cabog che producono lunghi frammenti di sequenza (contigs), ma tra loro disgiunti e di cui non si conosce la corretta posizione occupata all'interno del genoma d'origine. Alla maggior parte dei programmi manca una fase di "scaffolding" e "finishing", in cui tutti i frammenti prodotti con l'assemblaggio vengono ordinati e orientati, creando gli scaffolds. Il mio scopo era di realizzare un metodo di scaffolding, Consort, e di analisi per il miglioramento dell'assemblaggio ottenuto. Il programma richiede come dati di input un insieme di contigs prodotti assemblando le reads 454 con il Newbler, e un insieme di mate-pairs generati con il SOLiD.

La predizione genica è stata la mia seconda area di ricerca. E' un problema ben studiato e negli anni moltissimi programmi sono stati sviluppati per predire efficientemente i geni contenuti in un genoma. Tra questi i più utilizzati e conosciuti sono: Jigsaw, GeneID, GeneSplice, Genscan, Glimmer, SNAP, TigrScan, Twinscan. La maggior parte dei software richiede un insieme di dati di allenamento dal quale apprendere le caratteristiche per eseguire la successiva predizione, che molto spesso non sono disponibili. Pertanto, si devono creare a partire da genomi simili. Tuttavia, questa soluzione non è sempre applicabile, anche se molto spesso lavora bene e permette di ottenere buon risultati. Infatti, se l'organismo studiato è nuovo e non se ne conoscono altri di abbastanza vicini, si rischia di non avere i i dati richiesti. La mia ricerca in quest'area si applica allo sviluppo di un metodo di creazione di un insieme di dati di allenamento a partire da sequenze di trascriptoma dello stesso organismo. L'idea è di allineare le reads prodotte sul genoma e di estrarre tutte le regioni individuate, che sono potenziali geni. L'algoritmo implementato ha mostrato la possibilità di ricavare insieme di dati sicuri con questa tecnica. Tuttavia, il metodo è soggetto alla predizione di molti falsi positivi a causa dell'elevato rumore di fondo. Per evitare di creare un training set poco affidabile, è preferibile essere molto stringenti nei criteri di selezione dei geni.

Contents

1 Sequencing milestone

DNA sequencing is the process by which the nucleotide sequence of DNA molecule is determined. This represents the first crucial step for further genomic analysis. Why DNA sequencing is so important? Why the sequencing technology improvements are so critical? And why it is necessary to continue the sequencing improvements both at biochimical and at bioinformatics level?

Genome sequencing tools allow the decipheration of whole genomes, opening entirely new areas of biological inquiries which include the investigation of ancient genomes, of comparative genomics, of human disease, the characterization of ecological diversity, and the identification of unknown etiological agents. Among all the possible applications, the primary justification for the improvement efforts of the last years is the biomedical field research: the idea that the technology could become so affordable that sequencing the full genomes of individual patients would be warranted from a health-care perspective.

Unfortunately, sequencing is not yet an easy task, and it requires a lot of money and time. The great importance of these new genomic tasks justifies all studies and researches that have been performed to improve the sequencing process, both at economical and production level, from the Sanger DNA sequencing technology to the current Next Generation Sequencing (NGS) era. The principal aim is to reduce the sequencing cost of several orders of magnitude. At the same time, the development of new *ad hoc* bioinformatics tools for sequencing data allows to overcome and partially solves the biological limitations.

The next sections presents at first an overview of the existing sequencing technologies, from the beginning, with Sanger DNA sequencing, until the recent next generation sequencers; then it will follow a brief comparative discussion about the present available sequencers, their advantages and disadvantages, their points of strength, weaknesses and limitations, and their contribution in genomic project and research.

1.1 The beginning: Sanger sequencing

The genome sequencing era began in the mid-1970s when Frederick Sanger, a biochemist at Cambridge University in England, theorized an ingenious method for sequencing chains of DNA. Few years later, the technique was performed, and in 1977 Frederick Sanger and Alan R. Coulson published two methodological papers on the rapid determination of DNA sequence with chain-terminating inhibitors, known as Sanger Sequencing [1]. At the same time in the United States, Walter Gilbert and Allan Maxam were developing a somewhat similar sequencing method known as Maxam and Gilbert Sequencing [2]. The first method dramatically improved the earlier DNA sequencing techniques developed by Maxam and Gilbert, and it began the new sequencer, becoming the only used DNA sequencing method for over 30 years, until the coming of Next Generation Sequencing era. Over these years, the refinements to reduce sequencing cost and production time brought to semi-automatic Sanger-biochemistry.

In the original Sanger method, the DNA strand is used as a template, and a DNA polymerase moves over it (from 3' to 5') to generate the complementary strand. To determine the DNA sequence, four different PCR reactions are prepared, each containing a certain amount of radioactive labeled dideoxynucleoside triphosphate (ddNTP) analogs to one of the four nucleotides (ATP, CTP, GTP or TTP). This allows to prematurely truncate the synthesis each time an analog is incorporated in the reaction. The result is a pool of DNA fragment of different length, each of them ending with the nucleotide that is specific for that reaction. With a gel electrophoresis, the four reactions are separated into four lanes, and the sequence of the original template is determined, evaluating the position of the different spots in the gel. Later on, four base-specific fluo-

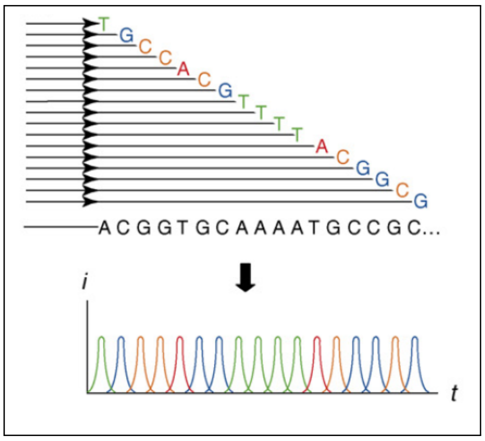


Figure 1.1: *Synthetic chain-terminator chemistry. Each chain terminates when a chain terminator (dideoxynucleoside triphosphate) with a base-specific fluorescent dye attached to it is incorporated (colored letters). The signal is detected electrophoretically, and it can be viewed as a trace that displays intensity (i) with respect to time of detection (t) (modified by Bentley D. [3]).*

rescent dyes were developed allowing to sequence in a unique PCR reaction a DNA fragment (see Figure 1.1). In Figure 1.2, there is a schematic workflow of Sanger sequencing method.

The main project made on Sanger sequencing method was the Human Genome Project (HGP). This was a biological milestone, that, after many years of hard work, brought to the development of high-throughput, high capacity production of DNA sequencing and associated finishing (gap closure and sequencing quality improvement) pipeline. The experience gained and the problem encountered during the project suggested that important improvements and innovative technologies should be introduced in the sequencing approach.
Two approaches were used for genome sequencing: BAC-based sequencing (BAC) and whole-genome sequencing (WGS). In the first approach almost 30,000 large bacterial artificial chromosome (BAC) clones were used. Each BAC contained approximately 100 kb human genome fragment, and all together they provided an overlapping set of path through each human chromosome, creating the final physical map [4]. Each clone was amplified in bacterial culture, isolated in large quantities, and sheared to produce size selected pieces of approximately 2 or 3 kb. Then, the DNA fragments were cloned into plasmid libraries that were sequenced to obtain a depth of coverage generally equal to 8. The assembly of all produced sequences recreated the BAC insert sequence in contigs (contiguous stretches of assembled sequence reads). In the final step, the produced contigs were refined, performing gap closure and sequence quality improvement (finishing), that generally produced a single contiguous stretch of high-quality sequence (typically with less than 1 error per 10,000 bases).

BAC-based sequencing (BAC)

The last improvements on the sequencing technologies and on the assembly purpose led to theorize and then apply the whole-genome sequencing (WGS) approach. In this new method, the genomic DNA is sheared directly into several distinct size classes and placed into plasmid and fosmid subclones. Oversampling the ends of these subclones to generate paired-end sequencing reads provides the necessary linking information to fuel whole genome assembly algorithms. This allows a more rapid sequencing process, but highly polymorphic or highly repetitive genomes remain quite fragmented after assembly.

whole-genome sequencing (WGS)

After all the improvements, with Sanger sequencing is now possible to achieve read-lengths (contiguous region of DNA sequence) of up to ∼1,000 bp, an accuracy per-base as high as 99.999%, and a cost on the order of $0.50 per kilobase.

The requirement of sequencing the whole genome of human and many other organisms in quickly brought to the automation and parallelization of some sequencing steps, and the creation of centers with hundreds of

DNA sequencing instruments. In spite of these, the time required for the sequencing as weel as the cost, the sequencing required time, and the cost were still too high to allow the sequencing of all organisms. A new DNA sequencing technology was required.

1.2 Sequencer characteristic

At the end of 1990s, the general common idea was the aim of very innovative changes in the sequencing technology. This highlights some questions: how evaluate the improvements? Which are the parameters or properties that characterize a sequencer? Which are the changes that could improve the technology?

The parameters that answer the questions are: cost, throughput, accuracy and completeness. The challenging task was to realize new sequencers that improve one or more of these characteristics, without compromising the others. Unfortunately, the efforts spent to decrease project cost or to increase throughput very often brought to a reduction in accuracy or completeness.

The sequencing cost

The sequencing cost is one of the most problematic aspects to consider in the development of new sequencing technologies. Only with a cost reduction, it would be possible to sequence every desired organism. The massive parallelization is the most effective parameter to control cost changes. The possibility to perform more reactions per experiment, reduces the required time and the costs. The higher the data density is, the less are the costs, which are also a significantly reduction in the volume of the used reagents, one of the principal causes of sequencing cost.

The sequencing throughput

The sequencer throughput is another important characteristic to control, interrelated with the cost. It depends on the speed of detection and the degree of parallelization. Moreover, the amount of produced sequences per run determine the genome coverage (how many times a base is covered on average), and consequently the final cost of the project. For a good and secure analysis, it is preferred an high coverage. So, this requires to find the right balance between the sequence coverage that can be reached and the reliability of the performed analysis.

Sequences accuracy and length

Finally, there are two other important characteristics to evaluate in a sequencer: the average quality and the length of the produced sequences. The sequence quality determines the accuracy of the analysis. If the produced sequence is full of sequencing errors, it requires a higher coverage to determine and solve the errors.
The sequence length depends on the problems encountered during the sequencing. For example, the sequencing is compromised when the DNA polymerase finds long homopolymeric (a long stretch of the same nucleotide)

or hairpin structure in templates. The sequence is truncated and it is not easy to evaluate the correct stretch length. Low complexity or repeated regions are other problematic situations, in which is not easy to determine the right final assembly, and long sequences aid in the sequence resolution. In a perfect situation, a set of long sequences of high accuracy will provide a high level of completeness. Unfortunately, as told before, a good high-throughput and cost reduction compromise sequence length. The sequence length depends on genomic task. For resequencing problem, an acceptable trade off is that the sequence would be long enough, and sufficiently accurate to align uniquely to the correct position in the reference. A statistics evaluation and simulation showed that 25-30 bases long is a good length threshold, under which is not advised to go [3] and [5]. For *de novo* assembly, the threshold goes up to 100 bases long, even if with mate-pair or paired-end technology, that generate two genome fragments separated by a known distance, it is possible to use also sequences 25-30 bases long. The information contained in the paired sequences allows a decrease of the required sequence length.

1.3 Next Generation Sequencing (NGS)

Next Generation Sequencing (NGS) era began in 2005, when the first new generation sequencer was commercially introduced. It was the GS 20 sequencer, developed by 454 Life Sciences (Roche). In few years, other platform were developed. The new technology is also known with the name of "second generation sequencers", with respect to Sanger sequencing methods that are called "first generation sequencers". The new sequencers are able to produce a large amount of sequences, called reads (contiguous regions of DNA sequence) in a single experiment, reducing costs and production time. This opens entirely new areas of biological inquiry.

Even if all platforms are quite different in the sequencing approach, their workflows are conceptually similar, and it could be summarized as follow (see Figure 1.2):

1. **DNA fragmentation**: library is prepared, randomly fragmenting the genomic DNA;

2. ***In vitro* adaptor ligation**: common adaptor are ligated to the produced fragments;

3. **Generation of polony array**: clonally clustered amplicons are generated for sequencing purpose. The available techniques are: *in situ* polonies, emulsion PCR or bridge PCR. At the end, PCR amplicons are spatially clustered on a single location, or on a planar substrate;

4. **Cyclic array sequencing**: the sequencing process consists of alternating cycles of enzyme-driven biochemistry and imaging-based data acquisition.

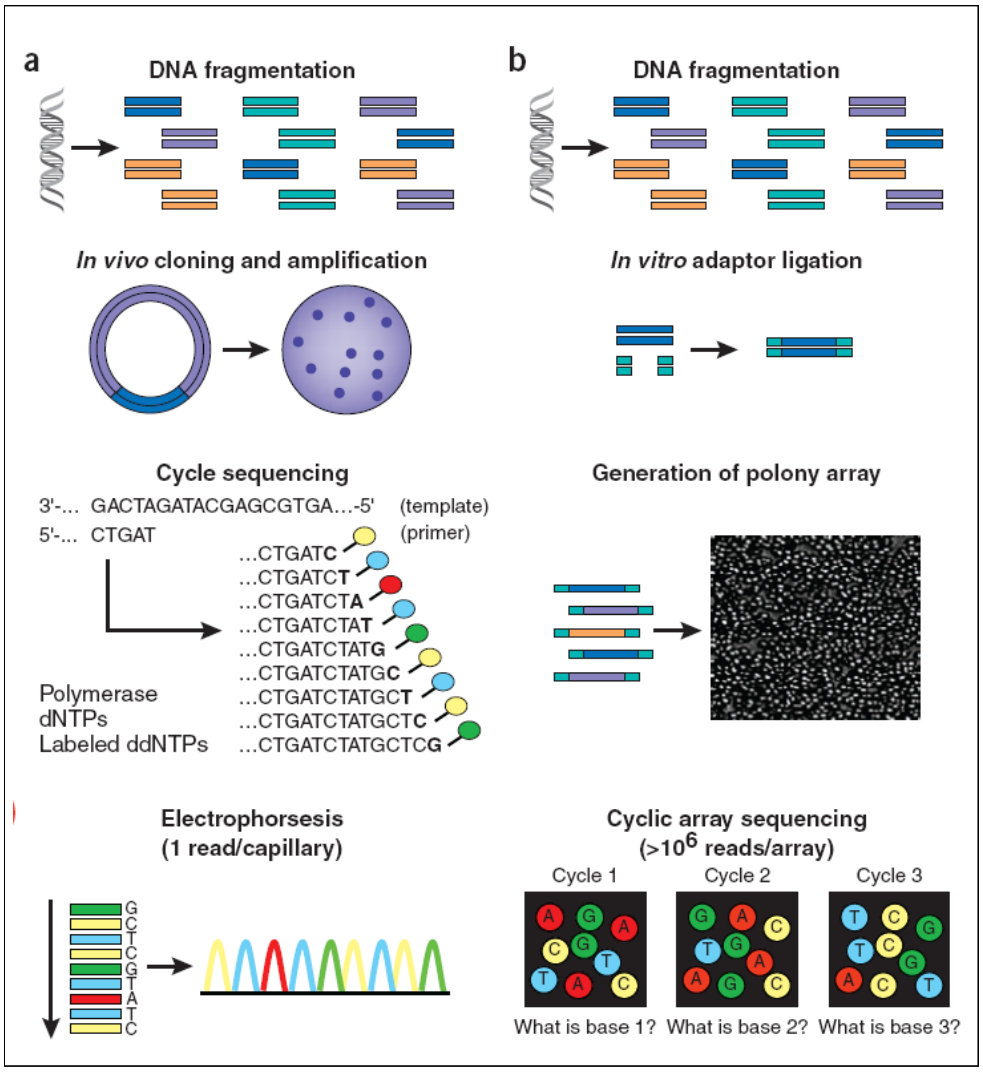


Figure 1.2: *Comparative workflow: first versus second-generation sequencing. (**a**) The Sanger sequencing workflow. genomic DNA is fragmented, then cloned to a plasmid vector and used to transform E. coli. Cycle sequencing creates the sequenced truncated fragments, with the ddNTP nucleotide at the end (one for each base). Finally, the four-channel emission spectrum is used to generate a sequencing trace. (**b**) Next Generation Sequencing workflow. DNA is fragmented, and adaptors are added ad the ends of the fragments. Fragments are fixed on arrays, and finally they are sequenced with cycle array sequencing. (Jay Shendure and Hanlee Ji [6]).*

The platform technologies could be classified into one of four categories:

microelectrophoretic, *sequencing by hybridization* and *sequencing by synthesis* methods.

Microelectrophoretic.

Even if the method is far away from the high-throughput and low cost of all next generation sequencers, microelectrophoretic method is still used, and a lot of work has been done to perform it. It is based on Sanger sequencing, and it achieves a high accuracy and long sequence production, that are two important aspects to consider for a good sequencing project.
The idea was to optimize Sanger sequencing method, trying to reduce the costs and to automate the process. This was applied using a 384-capillary automated sequencing machine, used to perform the separation of fluorescently labeled DNA sequencing fragments generated by the Sanger cycle sequencing reaction.

With microelectrophoretic sequencing method, it is possible to generate up to ∼1,000 bp raw sequences, with an accuracy of 99.99% with as few as three raw reads covering a given nucleotide. An heavily optimized sequencing centres could achieve a throughput of ∼24 bases/second per instrument.
The two most important advantages are the reduction of the reagent volumes and the ability to generate long sequences, with a high accuracy level. Unfortunately, the system presents a too low highthroughput, and it is costly and very conservative with respect to parallelization, therefore it could not be considered a relatively good option.

Sequencing by hybridization

This approach uses a set of differential hybridization of oligonucleotide probes to decode a target DNA sequence (Figure 1.3). The easier approach is to immobilize the DNA that is to be sequenced on a membrane or glass chip, and then to carry out serial hybridizations with short probe oligonucleotides (for example, 7-bp oligonucleotides). The probes are designed to have a common part and they differ only for few bases. This information could be used to infer the unknown sequence.

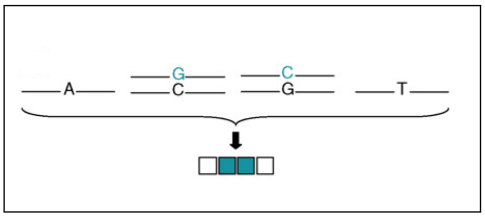


Figure 1.3: *Sequencing by hybridisation. Four oligonucleotides, each identical except for the central base, are immobilized on arrays, each in its own location (Bentley D. [3]).*

An advantage of sequence by hybridization is the large amount of sequence that could be sequenced. It is possible to achieve a great level of parallelization. The high-throughput could be used for SNPs detection, and all applications that require a lot of data for a valid statistics.

On the other hand, it is very important to design *ad hoc* probes, that avoid cross-hybridization to incorrect targets as a result of repetitive elements or chance similarities. The accuracy of base-calling relies on the ability of the method to discriminate between exact matches and single base mismatches. Some oligonucleotide sets will work better than others because the hybridisation characteristics of each set will vary depending on base composition, although there are experimental conditions that have been used to reduce this effect.

Sequencing by synthesis
In this class, there are all methods that amplify DNA molecules in spatially separate locations in a highly parallel array. Then, the produced sequences are used as templates for sequencing. The use of an array assures that all templates are ordered, and they could be easily located in a coordinate system.

Pyrosequencing

There are two different sequencing by synthesis methods: pyrosequencing and fluorescently labeled sequencing. In the pyrosequencing, when a nucleotide is incorporated into the growing DNA strand, pyrophosphate is released and it is enzymatically converted to ATP. When the ATP comes into contact with the enzyme luciferase, light is produced. A significant limit of this technology is the correct synthesis of the homopolymeric regions. In fact, the signal intensity must be correlated with the number of bases incorporated. The signal intensity variation is not measurable for long homopolymeric regions, and consequently, it is not easy to correctly determine the number of equal bases that compose the region.

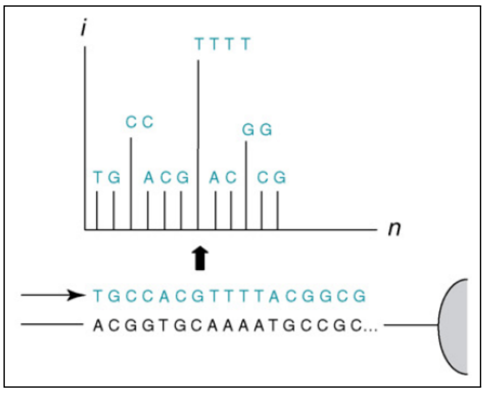


Figure 1.4: *Pyrosequencing. Template (black letters) is attached to a bead surface (shaded), and a primer directs DNA synthesis, which involves addition of one deoxynucleoside triphosphate per sequencing cycle. When dNTP comes in contact with the enzyme luciferase, light is produced (modified by Bentley D. [3]).*

The fluorescently labeled sequencing utilizes a dense array of small adapter molecules covalently bound to a glass surface. DNA template is then bounded to a small percentage of these adapters, and the fragments are replicated with several rounds of PCR. The final result is a very dense cluster of DNA templates bounded to the adapters. The sequencing process starts, adding fluorescently labeled reversible dNTP terminators, and DNA polymerase. When the dNTP is added at the correct position, the polymerization is temporarily halted, and the unincorporated reagents are then washed away. A laser excites the bound fluorescent labels, and the signal is recorded. The process could start a new cycle, using another dNTP.

Fluorescently labeled sequencing

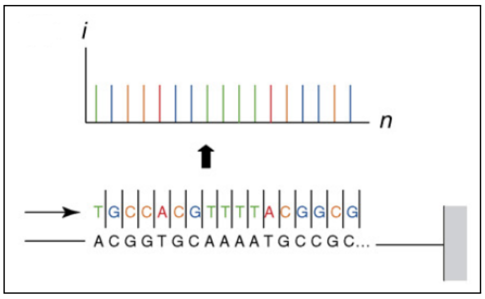


Figure 1.5: *Fluorescently labeled sequencing. Template (black letters) is attached to a flat surface (shaded), and polymerase and a mix of four base-specific fluorescently labeled reversible chain terminators is added. The polymerization is halted, and a laser excites the bounds, and the signal is recorded (modified by Bentley D. [3]).*

1.4 Next Generation Sequencers

Three platforms for DNA sequencing actually are in reasonably widespread use: the Roche/454 [1], the Illumina/Solexa Genome Analyzer [2], and the Applied Biosystems SOLiD^{TM} System [3]. The other existing platforms, like Helicos, have a less impact at the commercial and biological level.

The three platforms significantly differ in the type of sequencing chemistry they use, and this causes different strength and weakness points. They are briefly described below.

1.4.1 Roche (454) sequencer

The first sequencer commercially was Roche (454) GS 20 introduced (in 2005). It is based on pyrosequencing, which uses the pyrophosphate molecule released on nucleotide incorporation by DNA polymerase. This event fuels a downstream set of reactions that ultimately produces light from the cleavage of oxyluciferin by luciferase (see Section 1.3 for a more detailed description).

[1]http://www.454.com/

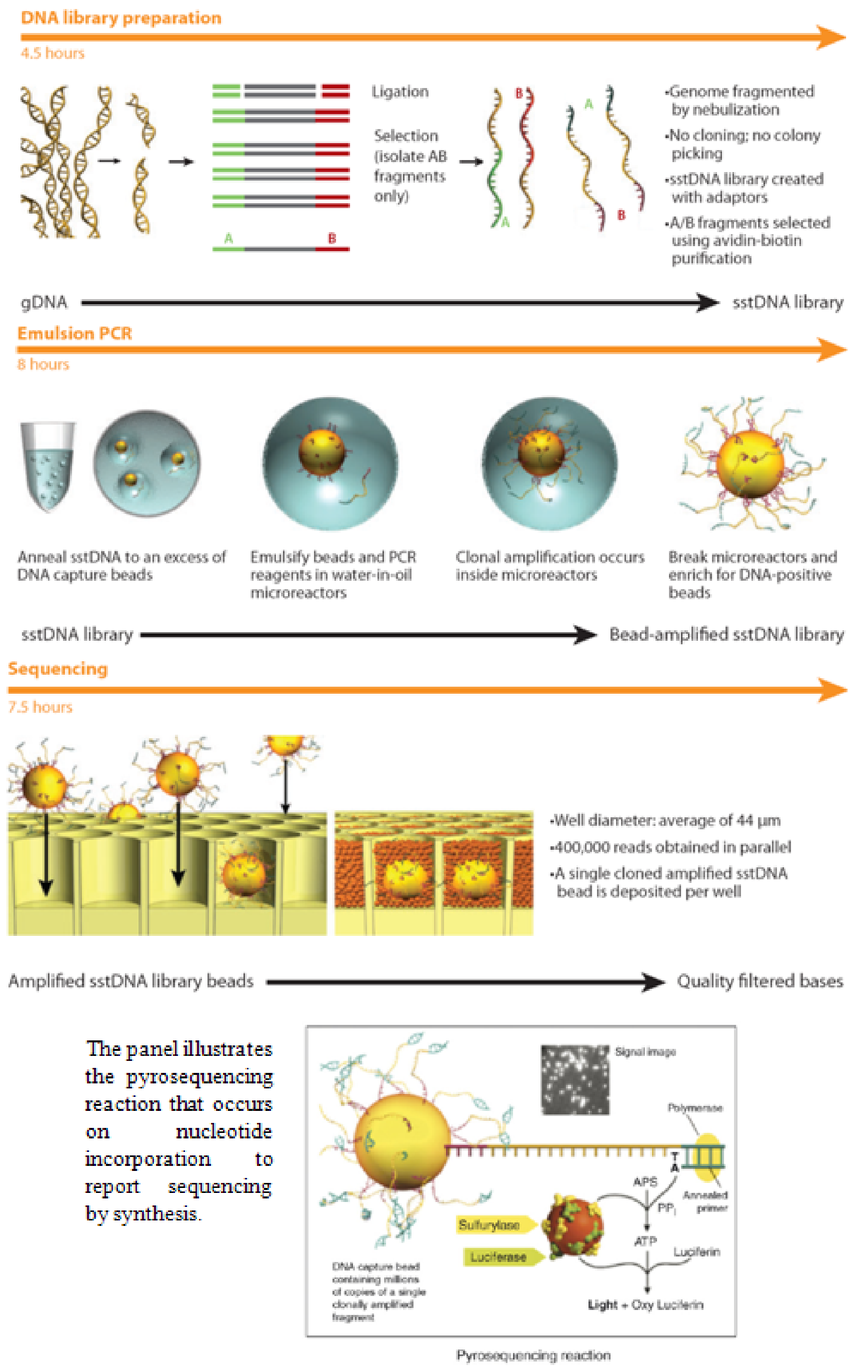
[2]http://www.illumina.com/technology/sequencing_technology.ilmn

[3]http://www.appliedbiosystems.com/absite/us/en/home/applications-technologies/solid-next-generation-sequencing.html

The 454 sequencing workflow begins with the DNA library preparation 1.6. The DNA is fragmented, and 454-specific adapters are ligated to the DNA fragments. The produced fragments are mixed with a population of agarose beads, that have on the surface oligonucleotides complementary to the adapter on the fragment library. This allow the bead association with a single fragment. Then there is a library amplification step, on which each bead is isolated into individual oil-water micelles that also contain PCR reactants, and thermal cycling (emulsion PCR) of the micelles produces approximately one million copies of each DNA fragment on the surface of each bead. Finally, the sequencing step: beads are arrayed into a picoliter plate (PTP; a fused silica capillary structure), providing a fixed location for each fragment. Enzyme-containing beads that catalyze the downstream pyrosequencing reaction steps are then added to the PTP and the mixture is centrifuged to surround the agarose beads. The plate is seated on the opposite CCD camera, that records the light emitted by each bead each time a new dNTP is added.

As told before, the main drawback of this method is due to the difficulty of the pyrosequencing technique to solve long homopolymeric regions. Over the point of detector saturation it is not possible to count how many equal dNTPs are added. So, these regions are prone to base insertion or deletion errors, during the sequencing. By contrast, thanks to the addition of a single dNTP each time, substitution errors are rarely encountered in these reads.

The current 454 sequencing system is Genome Sequencer FLX System, with long-read GS FLX Titanium chemistry. It is able to generate more than 1,000,000 individual reads with high-quality read length of 400 bases per 10-hour instrument run. Soon, a new sequencer will produce reads longer than the previous ones, approaching 1,000 bases. Even if it is not able to reach a high-throughput respect other sequencers, the high quality and length of the produce sequences allow to use this technique to solve many genomic tasks, re-sequencing, *de novo* sequencing and transcriptome analysis.


DNA library preparation
4.5 hours
Ligation
Selection (isolate AB fragments only)
A
B
•Genome fragmented by nebulization
•No cloning; no colony picking
•sstDNA library created with adaptors
•A/B fragments selected using avidin-biotin purification
gDNA
sstDNA library
Emulsion PCR
8 hours
Anneal sstDNA to an excess of DNA capture beads
Emulsify beads and PCR reagents in water-in-oil microreactors
Clonal amplification occurs inside microreactors
Break microreactors and enrich for DNA-positive beads
sstDNA library
Bead-amplified sstDNA library
Sequencing
7.5 hours
•Well diameter: average of 44 μm
•400,000 reads obtained in parallel
•A single cloned amplified sstDNA bead is deposited per well
Amplified sstDNA library beads
Quality filtered bases
Signal image
Polymerase
T
A
APS
PP$_i$
Annealed primer
Sulfurylase
Luciferase
ATP
Luciferin
DNA capture bead containing millions of copies of a single clonally amplified fragment
Light + Oxy Luciferin
Pyrosequencing reaction

The panel illustrates the pyrosequencing reaction that occurs on nucleotide incorporation to report sequencing by synthesis.

Figure 1.6: *The three main steps of* **454** ***Workflow****: library preparation, fragmentation and adapter ligation; fragment amplification with Emulsion PCR; sequencing step with pyrosequencing technique (Mardis E. [7]).*

Briefly, the performance parameters of the Genome Sequencer FLX Titanium system are [4]:

- **Obtain more comprehensive data**: generate more than 1,000,000 individual reads with improved Q20 read length of 400 bases per 10-hour instrument run;
- **Expand project capabilities**: It combines long single reads and Long-Tag Paired end reads to completely assemble genomes-often within a single run.
- **Reduce cost per result**: ultra-high throughput delivers the data at an affordable price while longer reads reduce over-sampling requirements;
- **Increase productivity**: genomic libraries are constructed in hours in a single tube;
- **Driving results, not the accumulation of data files**: a package of easy-to-use software tools is included, for an easy data interpretation and analysis.

1.4.2 Illumina Genome analyzer

In the 2006, a new sequencer was produced, Illumina Genome analyzer. It is based on sequencing-by-synthesis and it uses a fluorescently labeled method.

As in the 454 workflow, the Illumina process starts with a DNA library preparation, in which DNA is randomly fragmented, and Illumina-specific adaptor are ligated at both ends of the fragments. The fragments are attached at the surface of a flow cell, on which there are oligos that bound with the adaptors on the fragments. The amplification step is performed with a bridge PCR: DNA polymerase is used to produce clusters of approximately one million copies of the original fragment. This allow to reach the required signal intensity for base detection during the sequencing. The sequencing implies that the four nucleotides are simultaneously added to the flow cell channels, along with DNA polymerase, for incorporation into the oligo-primed cluster fragments. The process is halted to perform an imaging step for the acquisition of each base incorporation. Then the 3' blocking group could be removed and the next incorporation steps continue. See `http://www.illumina.com/pages.ilmn?ID=203` for an animation of this process.

[4] `http://454.com/products-solutions/system-benefits.asp`

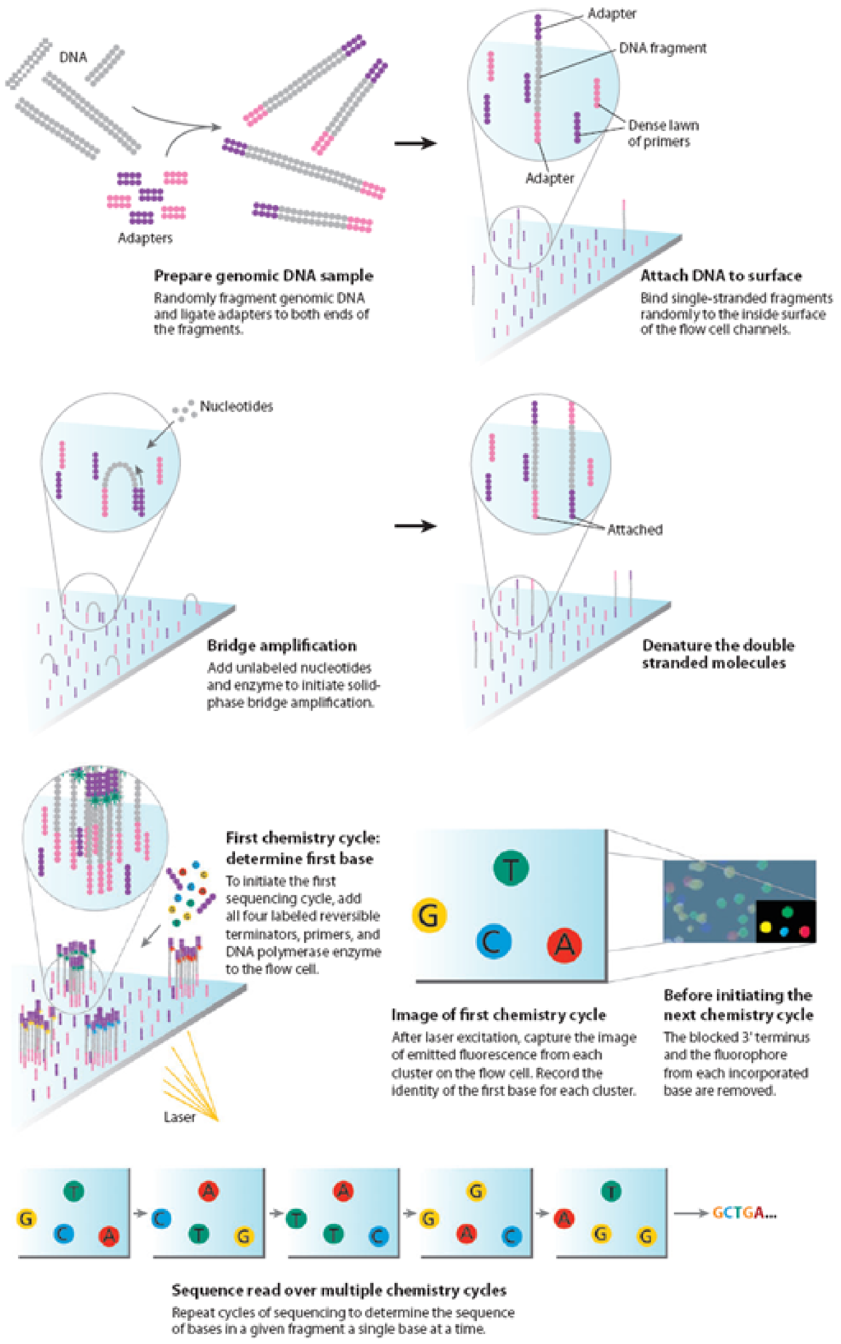
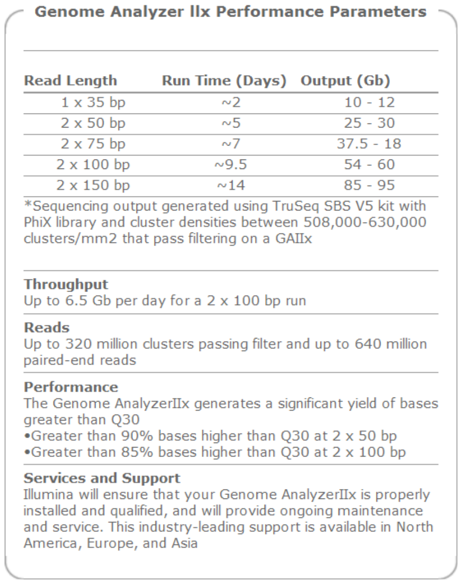

DNA
Adapters
Prepare genomic DNA sample
Randomly fragment genomic DNA and ligate adapters to both ends of the fragments.
Adapter
DNA fragment
Dense lawn of primers
Adapter
Attach DNA to surface
Bind single-stranded fragments randomly to the inside surface of the flow cell channels.
Nucleotides
Bridge amplification
Add unlabeled nucleotides and enzyme to initiate solid-phase bridge amplification.
Attached
Denature the double stranded molecules
First chemistry cycle: determine first base
To initiate the first sequencing cycle, add all four labeled reversible terminators, primers, and DNA polymerase enzyme to the flow cell.
Laser
Image of first chemistry cycle
After laser excitation, capture the image of emitted fluorescence from each cluster on the flow cell. Record the identity of the first base for each cluster.
Before initiating the next chemistry cycle
The blocked 3' terminus and the fluorophore from each incorporated base are removed.
GCTGA...
Sequence read over multiple chemistry cycles
Repeat cycles of sequencing to determine the sequence of bases in a given fragment a single base at a time.

Figure 1.7: *The three main steps of* ***Illumina Workflow****: library preparation, fragmentation and adpter ligation; fragment amplification with bridge PCR; sequencing step with fluorescently labeled sequencing (Mardis E. [7]).*

Illumina produces reads that are shorter than 454, only around 100 bases long, but with high-throughput. Read-lengths are limited by multiple factors that cause signal decay and dephasing, such as incomplete cleavage of fluorescent labels or terminating moieties. The dominant error type is substitution, due to the simultaneously addition of all the four nucleotides in the reaction mix, rather than insertions or deletions of the 454 platform.
The last commercialized sequencer is Genome AnalyzerIIx [5] . It offers a powerful combination of 2 x 150 bp read lengths and up to 640 million paired-end reads per flow cell, enabling a broad range of high-throughput sequencing applications. Evidenced by a vast number of peer-reviewed publications in an ever-broadening range of applications, Illumina sequencing technology with the Genome AnalyzerIIx is a proven platform for genomic discovery and validation.


Genome Analyzer IIx Performance Parameters

Read Length	Run Time (Days)	Output (Gb)
1 x 35 bp	~2	10 - 12
2 x 50 bp	~5	25 - 30
2 x 75 bp	~7	37.5 - 18
2 x 100 bp	~9.5	54 - 60
2 x 150 bp	~14	85 - 95

*Sequencing output generated using TruSeq SBS V5 kit with PhiX library and cluster densities between 508,000-630,000 clusters/mm2 that pass filtering on a GAIIx

Throughput
Up to 6.5 Gb per day for a 2 x 100 bp run

Reads
Up to 320 million clusters passing filter and up to 640 million paired-end reads

Performance
The Genome AnalyzerIIx generates a significant yield of bases greater than Q30
•Greater than 90% bases higher than Q30 at 2 x 50 bp
•Greater than 85% bases higher than Q30 at 2 x 100 bp

Services and Support
Illumina will ensure that your Genome AnalyzerIIx is properly installed and qualified, and will provide ongoing maintenance and service. This industry-leading support is available in North America, Europe, and Asia

Figure 1.8: *Genome Analyzer IIx Performance Parameters.*

1.4.3 Applied Bioststems SOLiD^{TM} Sequencer

Applied Biosystem SOLiD^{TM} Sequencer was commercialized in October 2007. It is based on sequencing-by-ligation, as mentioned in the name **S**equencing by **O**ligo **Li**gation and **D**etection, **SOLiD**. It is quite different from the others next generation sequencers, because it develops a color space system: the produced sequences are not represent of bases as usual, but by colors.

SOLiD Sequencing process starts with the sample preparation: library is fragmented and two oligo adaptor-linked are attached at the fragment boundaries. The fragments are subsequently bound to $1\mu m$ magnetic beads, that present complementary oligos, and each bead-DNA complex is amplified by emulsion PCR.
After the amplification step, the beads are covalently attached to the surface of a specially treated glass slide, that is placed into a fluidics cassette within sequencer. The SOLiD System is able to process two slides each time: mean while in a slide are added sequencing reactants, in the other the imaging process is performed, and viceversa

SOLiD system develops a 2-bases encoding system (see Figure 1.9): a pool

[5] http://www.illumina.com/systems/genome_analyzer_iix.ilmn

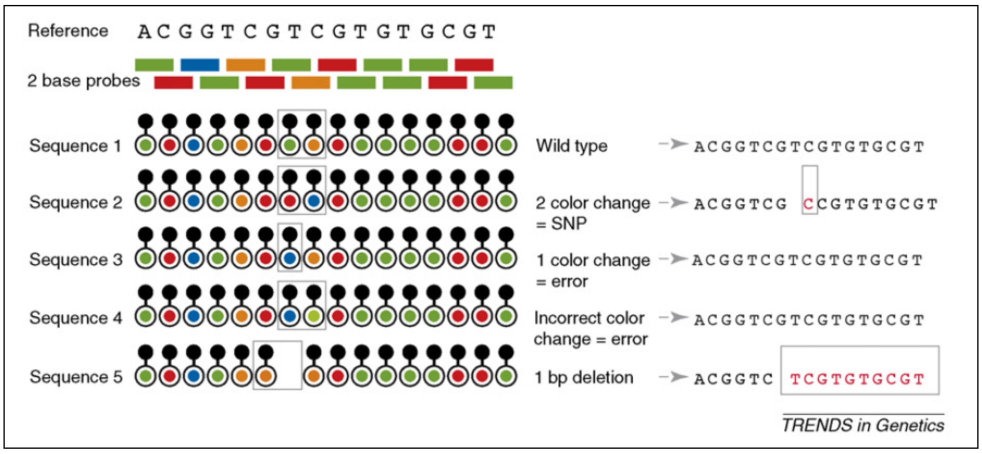

Reference ACGGTCGTCGTGTGCGT
2 base probes
Sequence 1
Sequence 2
Sequence 3
Sequence 4
Sequence 5
Wild type ACGGTCGTCGTGTGCGT
2 color change = SNP ACGGTCG CCGTGTGCGT
1 color change = error ACGGTCGTCGTGTGCGT
Incorrect color change = error ACGGTCGTCGTGTGCGT
1 bp deletion ACGGTC TCGTGTGCGT
TRENDS in Genetics

Figure 1.9: *Principles of two base encoding (modified by Mardis E. [8]).*

of probes with dual bases encoding, and 4 fluorescent dyes. Each probe is a semi-degenerate 8-mer oligonucleotides with a specific pattern: the first 3 bases are degenerate (n), the last three are universal (z), and the 4th and 5th bases are the two bases to interrogate. With four bases (A, C, G, T), there are 1024 different dinucleotide combinations, that correspond to 1024 different probes. Each dye is associated to 4 dinucleotides, so a single color could encode for 4 different dinucleotides. This allows to control the possible errors caused during the sequencing, in fact a base is determined by two colours, and errors bring inconsistency in the produced sequence.
When a matching 8-mer hybridizes to the DNA fragment sequence adjacent to the universal primer 3' end, DNA ligase seals the phosphate backbone. After the ligation step, a fluorescent readout identifies the fixed base of the 8-mer depending on the cycle number. A subsequent chemical cleavage step removes the sixth through eighth base of the ligated 8-mer by attacking the linkage between bases 5 and 6, thereby removing the fluorescent group and enabling a subsequent round of ligation. The process occurs in steps that identify the sequence of each fragment at five nucleotide intervals. A second round of sequencing starts with the hybridization of an n-1 positioned universal primer, and subsequent rounds of ligation-mediated sequencing, and so on. An overview of the SOLiD workflow is presented at `http://marketing.appliedbiosystems.com/images/Product/Solid_Knowledge/flash/102207/solid.html`.

SOLiD produces reads that are much shorter than all other sequencers (50 bases), and around 300 million reads per run. The color space solution has been of great impact both at bioinformatics and informatics infrastructures. A lot of work and ad hoc tools have been developed to manage this data. SOLiD produces a very high-throughput and its 2-base enconding system allows to determine and correct quite all sequencing errors. Moreover, this

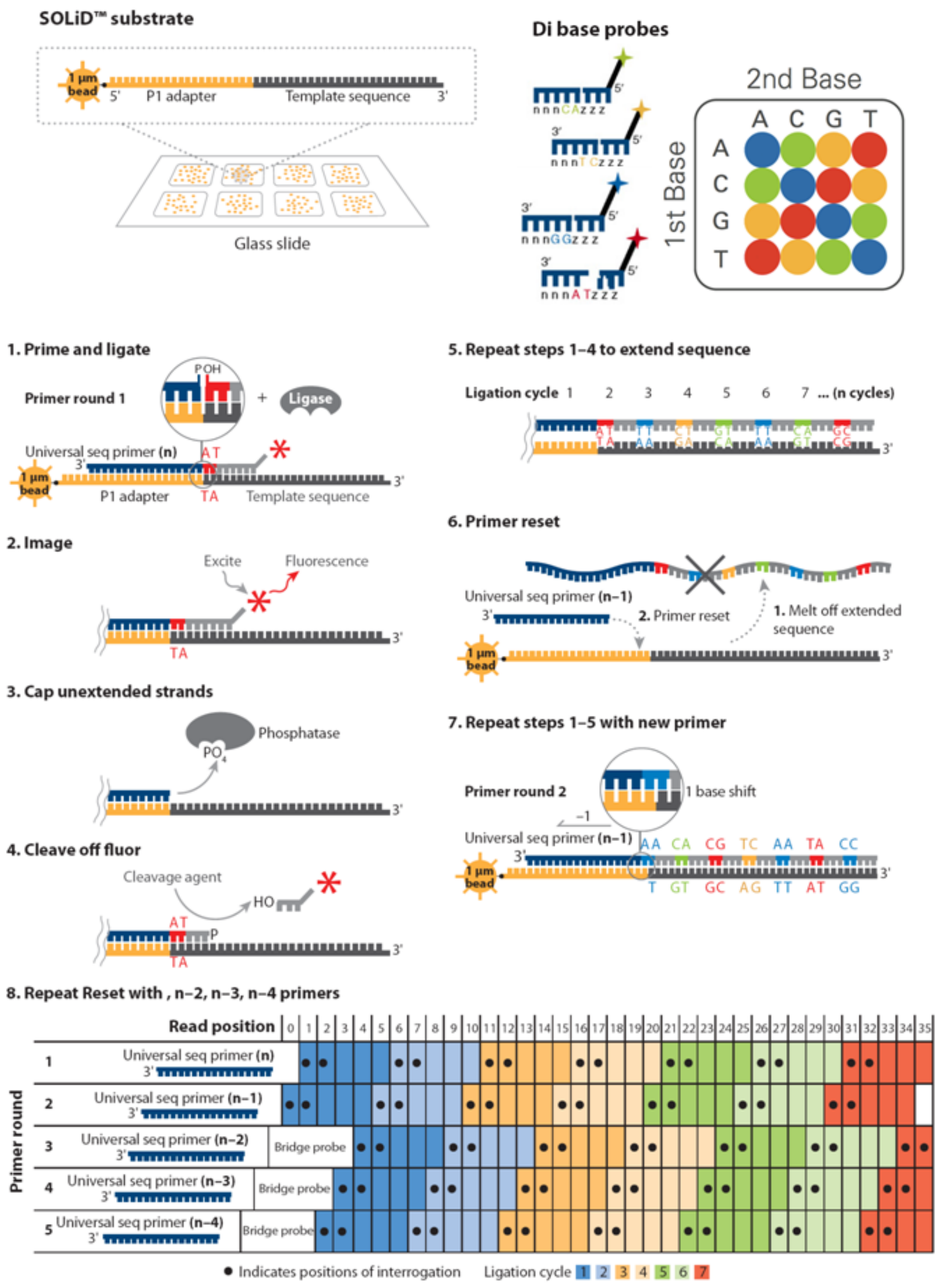


Figure 1.10: ***SOLiD Workflow****: a fragment is bounded to the magnetic bead, amplified by emulsion PCR and covalently attached to the glass slide. The sequencing step is performed using 16 different probes, with an associated flourescent dye. The probes are ligated, then there is a flourescent detection phase, and finally probes are removed for the next ligation (modified by Mardis E. [7]).*

discriminates an error from a polymorphism event.

The last commercialized sequencer is SOLiD^{TM} 4 System [6], [7], [8]. Its benefits are:

- **Scalable system**: The SOLiD^{TM} 4 System provides 100 GB for the $6,000 Quality Genome;
- **Superior accuracy**: More than 80% of the bases have quality values >30 for higher confidence in results;
- **Uniform coverage**: *ad hoc* reagents improve coverage to enable the discovery of rare variants in (GC/AT-rich) regions of the genome for fewer false negatives;
- **Expanded application support**: The use of barcoded paired-end sequencing that detects somatic mutations, novel splice variation, and fusion transcripts with less input material;
- **Automated sample preparation**: 80% reduction in hands-on time.

1.5 Sanger vs Next Generation Sequencers

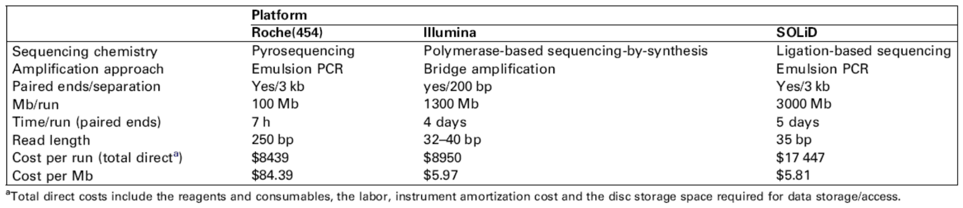
Next Generation Sequencing have a great impact at biological level and open a new field of reserch. There are three main advantages of second-generation relative to Sanger sequencing. First of all, the construction and clonal amplification of library in vitro solve several bottlenecks that restrict the Sanger sequencing. Then, array-based sequencing enables a much higher degree of parallelism than conventional capillary-based sequencing. This allows to obtained, in parallel hundreds of millions of sequencing reads. Finally, the immobilization of template DNA on planar surface allows to use a less quantity of reagent volume, reducing the costs.
The advantages of second-generation DNA sequencing are currently offset by several disadvantages. The most important is the final read length, that is much shorter than Sanger one, and raw accuracy that is lower than the achieved quality in conventional sequencing. Both of these disadvantages could be solved increasing the produced sequence coverage or developing new tools that work on this direction.

[6] *main page*: `https://products.appliedbiosystems.com/ab/en/US/adirect/ab;jsessionid=W1JwNlkVThGQhTzlhQhnJ9GTnMRyK1TmwZrTQzpJVJgqsrXGmRpQ!1265730959?cmd=catNavigate2&catID=607061`

[7] *product description*: `https://products.appliedbiosystems.com/ab/en/US/adirect/ab?cmd=catNavigate2&catID=607061&tab=DetailInfo`

[8] *product benefits*: `http://www.appliedbiosystems.com/absite/us/en/home/applications-technologies/solid-next-generation-sequencing/next-generation-systems/solid-4-system.html?abhomepage=eur`

The above consideration suggest that, by now, it does not exist a sequencer that is absolutely better than the others. In fact, each of them has different characteristics that advantage it in some applications, and disadvantage in others. So, before the realization of any kind of genomic, it is important to determine which is the best sequencing technology for the task, and sometimes, it is possible to apply a mixed approach, using more than one technology in the same project. For small-scale projects in the kilobase-to-megabase range, Sanger sequencing method remains the technology of choice for the immediate future. While, large-scale projects depend entirely on next-generation sequencing. There are important differences among the next generation sequencing platforms themselves (see Table 1.11). Some applications (e.g., resequencing) may be more tolerant of short read-lengths than others (e.g., *de novo* assembly). This suggests that in some applications the high-throughput of SOLiD and Illumina platforms are to prefer to the long reads produced by 454 technology. In other cases, it is more useful to use less reads, but of a greater length.



	Platform		
	Roche(454)	Illumina	SOLiD
Sequencing chemistry	Pyrosequencing	Polymerase-based sequencing-by-synthesis	Ligation-based sequencing
Amplification approach	Emulsion PCR	Bridge amplification	Emulsion PCR
Paired ends/separation	Yes/3 kb	yes/200 bp	Yes/3 kb
Mb/run	100 Mb	1300 Mb	3000 Mb
Time/run (paired ends)	7 h	4 days	5 days
Read length	250 bp	32–40 bp	35 bp
Cost per run (total direct[a])	$8439	$8950	$17 447
Cost per Mb	$84.39	$5.97	$5.81

[a]Total direct costs include the reagents and consumables, the labor, instrument amortization cost and the disc storage space required for data storage/access.

Figure 1.11: *Comparing metrics and performance of next-generation DNA sequencers (Mardis E. [7]).*

1.6 Fragment and Mate-pair library

The DNA sequencing could be performed creating two different kind of libraries: the “fragment library” and the very useful “paired-end” or “mate-pair” tag library (PET). In the first case, the library is prepared with a DNA fragmentation into pieces of specific length. In the second application, paired-end reads are produced. Paired-ends are short paired tags (20-50 base pairs) extracted from the two ends of target DNA fragments, that are linked and sequenced to create the tag library. After the sequencing, the tags are paired, and then mapped on the genome reference, allowing to find the boundaries of the target DNA fragments in the genome landscape and to their relative orientation and separation. The paired-ends have a separation estimate, provided by software, as the fragments size distribution measured on a so called library of reads.

There are multiple methods for constructing PET structures (see Figure 1.12), and all available new sequencing platforms can sequence it. The original PET

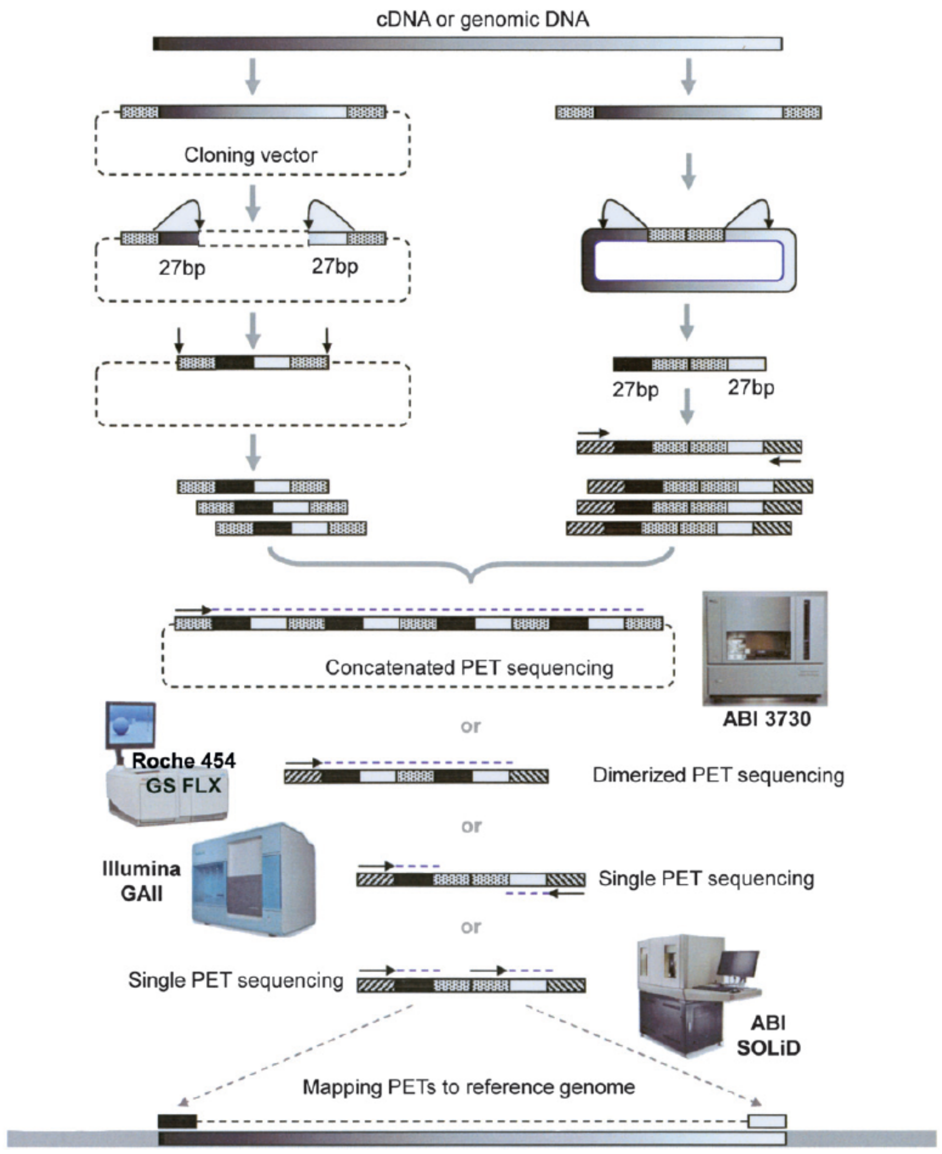


Figure 1.12: *Schematic view of PET methodology (Fullwood M. J. [9]).*

method was "cloning-based" using plasmid (cosmid, fosmid or BAC) vectors to link 5' and 3' tags. A recent alternative involves a direct circularization of the target DNA fragments with linker oligonucleotides that covalently join the two ends of a DNA fragment. Before Next Generation Sequencing instruments, the traditional method for short tag sequencing was to concatenate the tags into long stretches of DNA, using Sanger sequencing methods. Now, the next generation sequencers work well with paired-end constructs, because they could produce a large amount of short reads, as required in paired-end sequencing. Each platform develop a particular paired-end sequencing protocol, as briefly show in Figure 1.12.

Paired-end in Roche/454

Roche/454 links two units of paired-end tags together to form a diPET template, that will be sequenced. Using this approach it is able to generate half of million paired-end sequences in a single run. Illumina/Solexa is able to perform paired-end sequencing in three way. First, it can sequence paired-end constructs made from long DNA fragments (cDNA or genomic DNA). It prepares on the flow cell the single-strand DNA templates and it executes the first reads. After that, it synthesided the second-strand DNA to read the other direction. The second approach is to sequence the entire paired-end tag construct, since the sequencer can read more than 75bp. Finally, the third approach is to bypass paired-end tag construction and sequence the two ends of DNA fragments using the paired-end sequencing method described above. Although this last approach is simple and straightforward, it is limited to short DNA fragments that can be amplified by bridging PCR on the surface of the flow cells, and therefore, only short fragments of a few hundred base pairs can be paired-end sequenced by this method. For longer DNA fragments, paired-end sequencing has to be done through making paired-end tag libraries first. The first approach is able to produce long paired-end library (LPE) of thousand of bases, that are called in Illumina system mate-pair library. The last approach produces only short paired-end library, that are insert of few hundred of bases, called paired-ends. SOLiD/Applied Biosystem sequences paired-ends in a easy way. It cuts the two ends of a fragments, and it attacks at the ends of an internal adaptors. After that, it adds two new adaptors at both ends and the first tag is read using the primer that primes onto the flanking adaptor, while the second tag is sequenced from the middle linker of the paired-end construct.

Paired-end in Illumina/Solexa

Paired-end in SOLiD/Applied Biosystem

All the three most used next generation technologies have advantages and disadvantages. Roche/454 seems to be the best sequencer to use, but it is too expensive, due to the reagent cost for running. Illumina/Solexa is cheap and high-throughput, but paired-end sequencing takes up to a week of running time. Finally, SOLiD/Applied Biosystem is cheap and probably has the highest throughput but also the slowest runs. Thanks to the high-throughput and low cost, Illumina/Solexa and SOLiD/Applied Biosystem are the most used sequencers for paired-end creation.

Paired-end libraries are essentially to perform a *de novo* accurate assembly of small and large genome, using next generation sequencing data, due to their ability to span repeats longer than individual reads. Besides, they are very useful in finishing task after the assembly execution. In fact, they help the gap closure procedure, suggesting which are the reads that fall in the gaps. Paired-end reads are very often used to study genome structural variation, like insertion or deletion of genome sequences, or chimeric regions, and in transcriptome analysis.

2 Gene Prediction

During the first year of my Ph.D., I have been involved in the study of gene prediction problem, with the aim of developing a new tool able to create a training set of secure genes for an organism with Next Generation Sequencing data. As it will be soon explain in the following introduction section (see Section 2.1), the task is not easy to solve, and I encountered a lot of different difficulties. Moreover, the issue has yet been well handled, and I tried to contribute with a very innovative and useful technique, as described in the Sections 2.2, 2.3, 2.4

2.1 Introduction

Gene prediction (or gene finding) is the problem of identifying stretches of sequence (gene) in genomic DNA that are biologically functional, and to define their internal structure. This is one of the first and most important steps in understanding the genome of a species once it has been sequenced. However, this task is not keeping pace with the large amount of raw data coming from the sequencing of many organisms (both in the plant and animal kingdoms) over the last years. The Next Generation Sequencing technologies have further increased the amount of produced data, with the consequence urgent need of very accurate and fast tools that elaborate them. For all of these reasons, gene prediction problem has become a central issue in the field of bioinformatics.

Gene finding in a prokaryotic genome is an easier task compared to eukaryotic gene prediction, because of the presence of a higher gene density and the absence of introns in their protein coding sequences. This fact explains why the first problem has quite been solved, while the second one is yet a very challenging task, as described below.

An eukaryotic gene is composed by a transcribed region and cis-regulate factors for the gene expression, such as the promoter one.

This last control both the initiate site and the extent for the transcription of protein coding regions, and it is mostly found in the 5' end of the gene. The genes are separated by intergenic regions, and they are internally composed by coding sequences (exons), alternated with noncoding ones (introns), that are removed before the translation process by the splicing mechanisms. Very often introns contain stop codons, and this complicates the gene finding process and the searching of the exact exon boundaries. The coding regions are limited at 5' and 3' by the untranslated terminal regions (UTRs), which seem to play a role in the post-transcriptional regulation of gene expression, such as the regulation of translation and the control of mRNA decay.

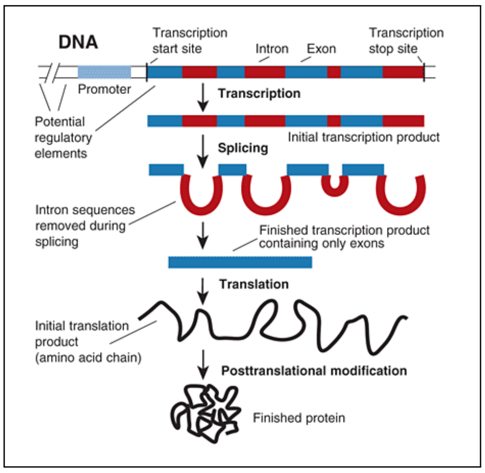

DNA
Transcription start site
Intron
Exon
Transcription stop site
Promoter
Transcription
Potential regulatory elements
Initial transcription product
Splicing
Intron sequences removed during splicing
Finished transcription product containing only exons
Translation
Initial translation product (amino acid chain)
Posttranslational modification
Finished protein

Figure 2.1: *The gene transcription and translation process in eukaryotic organism.*

Genes are also composed by many specific functional sites (also called signals), that regulate transcription (transcription factor binding sites and TATA boxes), splicing (donor and acceptor sites and branch points), polyadenylation (poly(A) site) and translation (initiation site, generally ATG with exceptions, and stop codons). All these sites have to be discovered during the gene prediction process. The eukaryotic genes may code for multiple proteins thanks to the presence of alternative promoters or terminators, or through alternative splicing mechanisms. Moreover, very often the gene finders predict a lot of false positive genes for eukaryotic organism. Finally, small exons cannot be recognized using any of the composition-based methods that are successful for prokaryote coding regions. All these considerations explain why finding protein-coding genes in eukaryotic genomic sequences with in-silico methods remains an important challenge in computational genomics, despite many years of intensive research studies.

Existing approaches fall into two groups with respect to the technique they utilize: *intrinsic* or *ab initio* methods and *extrinsic* or *similarity-based* ones. The first class uses only the information contained in the input genomic sequence: it searches for typical patterns that generally characterize coding boundaries, and other signals inside and outside gene regions. Some programs that implement these strategies are Genscan [10], Augustus [11], HMMGene [12] and GeneID [13]. The second type applies the information coming from external sources as EST and proteins, as done by GeneWise [14].

2.1.1 *Ab initio* or *intrinsic* approach

These techniques use the statistical properties of particular patterns and signals, extracted from the genomic sequence of the studied organism. They are classified into consensus (or signal sensors) and nonconsensus (or content sensors) methods.

Consensus methods

The first group searches for a match with a consensus short sequence motif, that could be of the following type [15]:

- *Transcriptional signals*: the most often used signals are the initiator or cap, located at the transcription start site (TSS), and the $A + T$_rich TATA-box, typically around 30 bp upstream of the TSS;
- *Translational signals*: one of the most important signal is the 'Kozak signal' that is located immediately upstream of the initial ATG;
- *Splicing signals*: they are characteristic patterns positioned at the 5' and 3' ends of the intros, called donor and acceptor splice sites respectively. The most frequent motif is the GT and AG, also known as canonical sites; the other variants are similar to this one, but occur with lower frequency.

A large number of programs implement this technique, using weight matrices in place of the perfect consensus sequence. The match between the position in the pattern with a residue in the matrix gives a specific cost [16], that characterizes its goodness, and that will be used for the further similarity score computation. Some examples of these tools are GeneSplicer [17] and Splieceview [18].

Nonconsensus methods

The second group of ab initio techinique is based on the study of the different statistical and conservative properties, that characterize and discriminate the coding sequence (exons) from the non-coding ones (intronic, untranslated and intergenic regions). There are several measures that could be used, but the most utilized is the hexamer composition. This is the better parameter to distinguish between the two kind of regions, as reported by Fickett and Tung [19], and applied in Sorfrind program [20]. The Markov model chain is one of the most frequent statistical approach applied in the computation of this measure. It is a stochastic model assuming that the probability of the appearance of a given base (A, C, G or T) at a given position, X_i depends only on the k previous nucleotides X_{i-1}, ..., X_{i-k}, for some constant k, that is called the order of the Markov model. This means that a fifth-order

Markov model captures local dependencies in sequence, at the level of the hexamer.
Indeed coding regions tend to have higher $C + G$ content compared to non coding regions. These differences can help the prediction problem.

The *ab initio* methods could be further classified into trained methods (e.g. Glimmer [21]), that use unbiased sets of coding regions, and untrained ones (e.g. Genescan [10]), which utilize statistical properties in order to discriminate genes.

2.1.2 ***Similarity-based*** or ***extrinsic*** approach

Extrinsic or similarity-based approaches make use of information coming from external evidences that are compared to the reference genome. This is why they are also called comparative gene prediction methods. The basic assumption is that the functional regions are more conserved than non-functional ones. So, the similarities found aligning the external evidences (as proteins, EST, or information about similar organisms) to the genomic sequence, discriminate the coding stretches from the not coding ones, and infer the gene structure. Sometimes it is possible to use EST or proteins coming from the studied genome, that give a real evidence about the coding regions. Unfortunately, this point of strength is also a weakness one: if the used information is not accurate, or the quality of the input database is low, the performed prediction could be compromised.

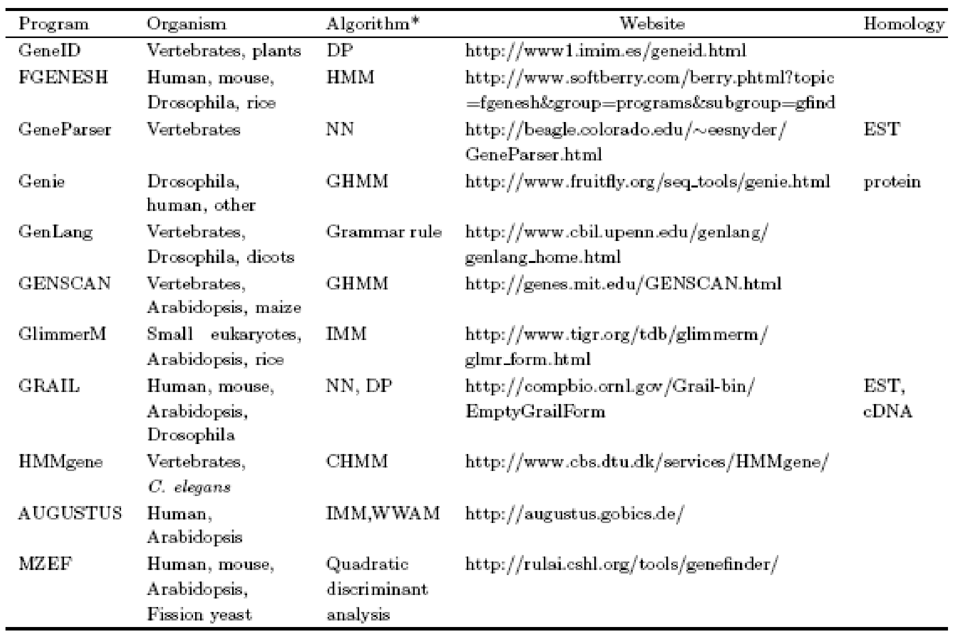
As mentioned above, similarity-based techniques generally use three kinds of reference sequences:

- The *proteins* are an useful information, and they could be found in SwissProt and PIR database. Almost 50% of the genes could be predicted with this strategy, but the presence of untranslated regions requires the application of other methods;

- The *ESTs* (expressed sequence tags) are another evidence, to compare to the cDNA (complementary DNA);

- The comparison at genomic level between similar organisms is another approach, that strictly depends on the similarity level between the two organism.

ESTs and proteins could be aligned locally using BLAST family tools, or globally using program like GeneWise [14].



Program	Organism	Algorithm*	Website	Homology
GeneID	Vertebrates, plants	DP	http://www1.imim.es/geneid.html	
FGENESH	Human, mouse, Drosophila, rice	HMM	http://www.softberry.com/berry.phtml?topic =fgenesh&group=programs&subgroup=gfind	
GeneParser	Vertebrates	NN	http://beagle.colorado.edu/~eesnyder/ GeneParser.html	EST
Genie	Drosophila, human, other	GHMM	http://www.fruitfly.org/seq_tools/genie.html	protein
GenLang	Vertebrates, Drosophila, dicots	Grammar rule	http://www.cbil.upenn.edu/genlang/ genlang_home.html	
GENSCAN	Vertebrates, Arabidopsis, maize	GHMM	http://genes.mit.edu/GENSCAN.html	
GlimmerM	Small eukaryotes, Arabidopsis, rice	IMM	http://www.tigr.org/tdb/glimmerm/ glmr_form.html	
GRAIL	Human, mouse, Arabidopsis, Drosophila	NN, DP	http://compbio.ornl.gov/Grail-bin/ EmptyGrailForm	EST, cDNA
HMMgene	Vertebrates, *C. elegans*	CHMM	http://www.cbs.dtu.dk/services/HMMgene/	
AUGUSTUS	Human, Arabidopsis	IMM,WWAM	http://augustus.gobics.de/	
MZEF	Human, mouse, Arabidopsis, Fission yeast	Quadratic discriminant analysis	http://rulai.cshl.org/tools/genefinder/	

Figure 2.2: *The table summarizes some gene prediction programs, reporting the organism on which they could be used, the base strategy and the online link, where it is possible to find them (modified by Hwan J. [22]).*

2.1.3 Combiner methods

Extrinsic or ab initio methods alone are not able to predict all genes. Only approximately half of them can be found by homology with other evidences contained in online databases. To find the remaining 50%, a good solution is to use prediction methods. The integration of the results obtained by more different evidences and tools allows to produce a more accurate prediction. A new class of programs was growing parallel to these ones based on this idea: the combiner methods.

Some examples of programs that apply a mix approach are:

- GenomeScan [23], that extends GeneScan, using blast alignments with protein sequences;

- HMMGene, that has an improved version that integrates into the HMM information coming from BLAST alignments of the query sequence with cDNA, EST and protein sequences [24];

- Combiner, that creates the consensus of the predictions coming from several individual gene finders [25], searching for contradictions and consensus to produce one output gene structure;

- Jigsaw, that is the most interesting program in the combiner class methods [26]. Its idea is to integrate the information coming from different kind of sources: EST, cDNA, aligned proteins, gene prediction coming from prediction programs,... It is composed by an initial learning phase, in which it learns from a training set, how safe is each input source. This phase assigns a score to each evidence, that is proportional to the ability to predict the training set genes. Successively, there is the prediction phase, that integrates the information coming from each source, considering the assigned scores, and predicts the genes.

In the Table 2.2, there is a brief list of the most used gene prediction programs. For each of them, it is specified the type of the organism for which they are trained, the kind of algorithm that it applies, the web site from which it is possible to download it, and the evidences that it uses.

2.2 A new gene prediction approach

As mentioned in the previous section, many efficient programs were able to predict quite all genes (high sensibility), reducing the number of false positive (high specificity). The gene prediction problem seemed to be quite solved, and it seemed not useful the development of a new tools. However, a very challenging and important task was not yet implemented: the creation of an *ad hoc* training set of well known genes. Indeed the prediction programs use this kind of information, to learn the rules that define gene structure to be applied in the prediction phase.

This is the weak point for all these methods: for new sequenced organism there are no available training set, and the programs do not work.
To overcome this problem, the naive idea is to employ the genes of a very similar and well known organism, that could simulate equal characteristics. However, even if this solution is very often applied with good results, sometimes it is not safe and feasible, because of the risk of a not strictly related knowledge acquisition. A better solution would be the creation of the training set from the considered organism itself, but this requires a lot of very expensive experimental work. The research moved on this direction: how could be used this solution in few time and low cost?
The creation of a training set of genes requires the identification of the segments that compose them and their structure, as the coding regions. Consequently, the gene training set creation problem has become the identification of these elements in few time and low cost. The coding regions are portions of DNA sequences that code for proteins, and that compose the transcriptome of the organism. Its sequencing, and the mapping of the obtained reads over the genome give the evidence about these regions displaced

over the DNA. After that, the only task to perform is their extraction and reconstruction. As said before, next generation sequencers work in few time and low cost, producing a large amount of data, as required. This was the idea that seemed to solve the training set creation problem in an efficient way, and that suggested the development of this new tool.

There are some useful considerations to do before to explain the implementation of the method. The not expression of all genes suggests a limit: the method is not able to detect all of them; however, it does not want to perform a complete gene prediction, but its only aim is the creation of a secure training set, for the future prediction with the yet existing programs. Another consideration to do is about the state of the RNA during the sequencing: it could be not mature, and it could contains little pieces of introns. So, it is necessary to use information coming from the coding regions boundaries to create the correct gene structure.

The above analysis and all the done observations brought to a very innovative solution for the problem, with the application of Next Generation Sequencing data. The next step is the development of the idea, with the implementation of an useful tools for the creation of gene training set, proving the reliability of the thought method. I tested and improved my program on *Vitis vinifera* organism, that was studied in my laboratory in the project of “gene prediction and genome annotation”.

2.3 CreateGene: a tool for gene training set creation

CreateGene produces a set of secure genes for a studied organism, to use for the training phase of gene prediction programs. It requires a set of transcriptome reads of the organism, that would be mapped over the related genome sequence, to evidence the coding regions.

A basic requirement for this tool is a low computational time and allocation of RAM memory, to be run even on desktops, without particular computational power. Moreover, it is important that the code will be easily maintained, and it will be possible to easily add, delete or change the code without a complete *de novo* implementation. To satisfy the basic requirements, CreateGene has been developed with a modular structure: each module realizes a specific function, and all together they solve the create training set problem. They are organized in a pipeline divided into three phases: an initial pre-processing step on which data is prepared; an execution step that parses the signal created by reads alignment, and detects the coding regions; finally, post-processing step determines the gene structure. In the next sections, the three stages will be described.

2.3.1 Pre-processing step: data preparation

This phase elaborates the RNA-seq input reads, preparing them for the next coding regions detection. So, the first argument to deal with is the choice of the suitable sequencer to use for this task (see Chapter 1). Is it preferable to have a large amount of short reads, as with SOLiD/Applied Biosystem and Illumina/Solexa, or to produce few but longer ones, as with Roche/454? For this kind of application, the first situation is better than the second one, because of the greater amount of produced data that help the discrimination from background noise. This could be due to the presence of sequencing errors inside reads, that mislead the alignments, with the result of some false covered regions. The probability to have a wrong read alignment due to its errors is lower than a correct one. It is even lower the possibility to have two or more wrong alignments in the same place. Consequently with an high-throughput, the difference of the coverage (the number of reads that cover the region) between the false regions and the real ones, is greater, and it is easier to discriminate between them, detecting only the correct evidences.

The second argument to discuss concerns the alignment of large quantity of short reads into the genome. A transcriptome mapped on it has its coding regions divided by introns (see Figure 2.3, a); therefore, the mapped reads could be classified into two groups: the reads that align inside the coding regions (like the green ones in the Figure 2.3, b), that create the coverage signal; the reads that align between two coding regions divided by an introns (like the red ones in the Figure 2.3, b), that are spliced in two pieces. As it will soon describe, this second type is very important, because it gets the information about the relationships between coding regions.

The mapping tool should be very efficient: it should work with short reads, even in color space (for SOLiD/Applied Biosystem reads); it should align read allowing splicing and mismatches; finally, it should run with few memory and in a low computational time. It was used PASS [27], developed in our laboratory, because it was easier to modify its functionality and adapt to the problem requirements. It is able to work in a multi-threading modality, parallelizing the short read mapping, and decreasing the required computational time. PASS allows a fixed maximum number of mismatches and gaps in the alignment (at least four mismatches and zero gaps for this analysis), and it stores all rejected reads. Moreover, it is used the best-hit function: if a read maps in different places with different scores (that is determined by the number of required mismatches and gaps), PASS returns the only alignments that have the best score value. A reads could be labeled as 'unique', if it maps only once in the genome with the best score, 'not unique' otherwise. The PASS output format is GFF (General Feature Format [1]), see Figure 2.4.

[1] http://www.sanger.ac.uk/resources/software/gff/spec.html

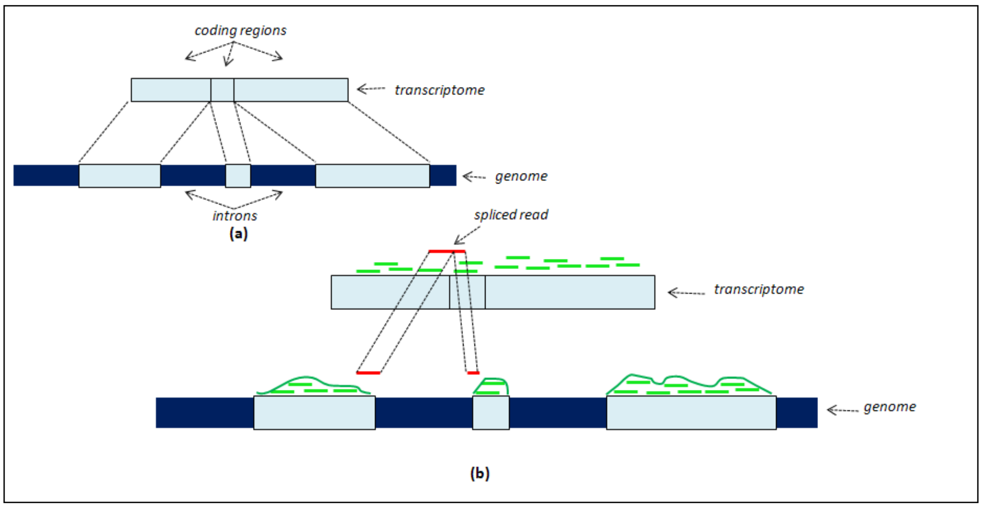


Figure 2.3: *On the left side, it is showd how a transcript maps on the genome: its coding regions are divided by introns. On the right side there is the representation of the two classes of reads: the reads that align inside a coding regions, and reads that aligned on a splice junction.*

```
scaffold_0   pass   match 6214487     6214519     33     -      .      ID=1edd0:1461:1;P="33-1";
```

Figure 2.4: *An example of a line of a file in GFF (General Feature Format) format. The first field is the name of the sequence, on which there is the feature; the fourth and fifth fields are the start and end position of the feature in the sequence.*

It is a standard format, studied to be easily parsed and processed. It is a tabular file, composed by nine fields, each of them describing a particular feature, and each line contains all the information about a single alignment. In the first column there is the name of the sequence that contains the alignment (it could be a chromosome, a scaffold, a contig,...), and in the fourth and fifth ones, there are the start and end position.

With a genome viewer, like GBrowse (Generic Genome Browser [2]) it is easy to plot the aligned reads on the genome, Figure 2.5. These tracks are called signals. In each position, the height of the curve corresponds to the number of reads that align and cover that position.

2.3.2 Processing step: coding region detection

The processing phase requires the analysis of the signal produced in the previous phase, for the gene extraction. How parse the signal, to extract them and their structure? As mentioned before, higher it is the signal, and

[2]`http://gmod.org/wiki/Ggb`

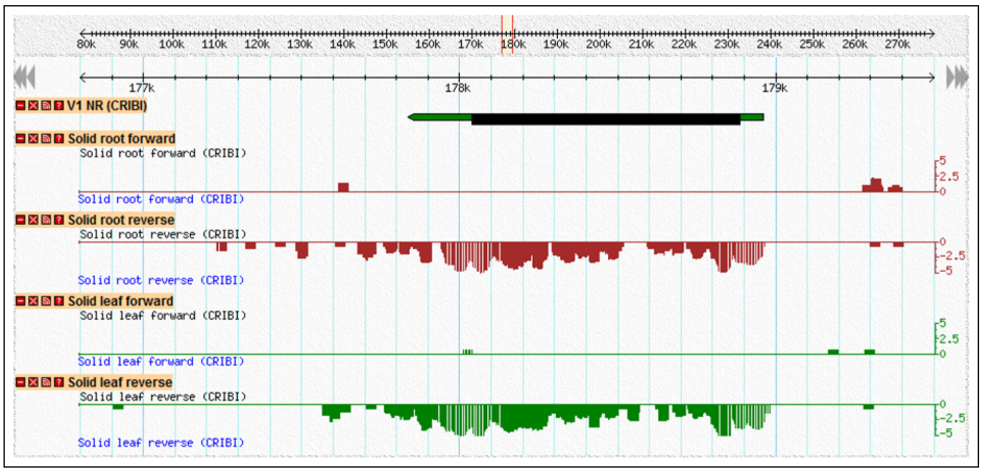

80k 90k 100k 110k 120k 130k 140k 150k 160k 170k 180k 190k 200k 210k 220k 230k 240k 250k 260k 270k
177k
178k
179k
V1 NR (CRIBI)
Solid root forward
Solid root forward (CRIBI)
Solid root reverse
Solid root reverse (CRIBI)
Solid leaf forward
Solid leaf forward (CRIBI)
Solid leaf reverse
Solid leaf reverse (CRIBI)
5
2.5
0
-2.5
-5

Figure 2.5: *A screenshot of the genome viewer GBrowse , on which some transcriptome reads have been loaded. The first track contains the genome, with its coding regions and non-coding ones. The other green and red tracks are the reads aligned on the genome.*

higher is the probability that the covered bases belong to a coding region. Nevertheless, there could be misleading scenario, caused by the background noise. The module should discriminate the two cases.

A sharp look at the signal plotted on the GBrowse, like in the Figure 2.5, highlights some problematic situations, to consider for a good use of the data. The signal is full of fluctuations, and the transition from a non-coding region to a coding one and vice versa, is not well defined. This could be due to the occurrence of wrong alignments, which causes background noise, and has to be filtered out.
A naive solution is the definition of a threshold: all the regions that have a coverage lower than the set value will be discarded, and only the other ones will be used. This strategy is very quickly to implement: the threshold depends on the produced coverage and on the genome type, and consequently it is not secure to determine an absolute value that works in each case. It would be better to estimate it for each experiment; Indeed the signal height depends on the genes expression level, and on the introduced sequencing errors. For example, there could be genes with a good coverage, because of their high expression level; differently, other genes can have a lower expression level, that has not to be confused with the background noise. So, in the same experiment, not all regions are equally represented and could be correctly predicted. However the aim of the project is not to predict all the genes but only a secure subset of genes. For this reason it is better to filter more positive regions than to introduce negative ones.

The above considerations led to the development of a very simple method, that evaluate a local threshold for each covered stretch of bases. It is a searching graph based approach, and it has been organized into two phases. In the first one, the stretches of adjacent bases with an associated non-zero score are clustered into homogeneous regions. Then, the created groups become the nodes of a graph, and the spliced reads are the edges that connect them. The most probable gene structure is discovered and reconstruct in the graph, using a searching algorithm. The gene discovering problem becomes the search of the best path(s) through the graph. Before a more detailed explanation of the method, it will be explained some basic notions about graph and cluster analysis theory.

A graph $G = (V, E)$ is a mathematical abstraction, used for object relationship representation. It is defined by a set of nodes $V = \{v_1, ..., v_n\}$ (also called *vertices*), used to represent the objects, and a set of edges $E = \{(v_i, v_j) | v_i, v_j \in V\}$ (also called *arcs*), that are the relations between them. Generally, a graph is drawn as a set of dots for the vertices, linked by lines or curves for the edges (see Figure 2.6). If the arcs could be traversed only in one direction, it is a directed graph, undirected otherwise. Starting from a node, and moving across the arcs, it is possible to visit the graph, across the paths. A simple path is a collection of distinct nodes: each node of it should be crossed exactly once.

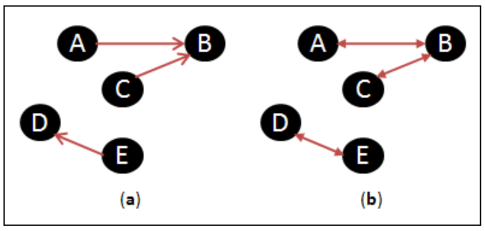


Figure 2.6: *A representation of a directed (**a**) and undirected (**b**) graph. The graph is composed by nodes (or vertices) and edges (or arcs). A connection between two nodes, represented by an edge, defines a relationship between the two nodes.*

The cluster analysis, or clustering, is the assignment of a set of objects (in this case bases) into subsets (called clusters), so that the elements in the same group have similar characteristics. For this application, it is used an hierarchical algorithms, with an agglomerative ("bottom-up") approach: initially, each base is a separate cluster, and they are iteratively merged into successively larger ones, using a distance measure. At the end of the process, each class should contain a stretch of at least two contiguous bases. This technique requires a good score system: which is the better distance measure, that define the similarity between bases?

For each base, the used score is the sum of all the reads that cover it, differentiating for the alignment type. Generally, a read that uniquely aligns has a greater probability to be correct than another one that maps on multiple positions over the genome. Anyway, a sequence could be rich of repetitive regions, stretches of equal motif, that allow multiple alignments of the reads, that do not have to be confused with the background noise. Also these case can be resolved as the reads map together in these positions, producing a higher coverage compared to background noise due to wrong matches.

Finally, a read could perfectly align without mismatches (all the bases match the sequence) or with some (one or more bases does not match). An alignment with less or none mismatches is more accurate and secure than with a lot of them. All these information are used to divide the alignments by type, and to rank them by importance, simplifying the discrimination process between false and true coding regions.

The score for each base is a vector of eight values, organized in decreasing order as follow: in the first four values will be counted all reads that uniquely align on that position, with zero, one, two or three mismatches respectively, while in the successively four there are the not unique alignments with zero, one, two or three mismatches (see Figure 2.7). If there are no reads that cover a base, its associated score is the zero vector $\overrightarrow{0} = (0, ..., 0)$. All the bases that are covered at least by a base are called non-zero bases.

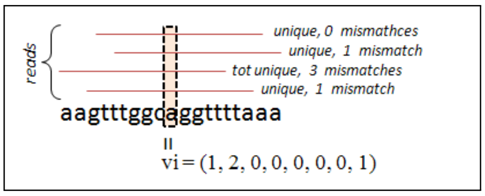


Figure 2.7: *An example of base score. Four reads cover the base a: two reads uniquely align with one mismatch, one read uniquely aligns with zero mismatches, and the last not uniquely aligns with three mismatches.*

Each base with the associated vector score represents a point in the $\mathcal{R}^8$. There are some measure that could be used to evaluate the distance between two points in the real space. One of the most used is the Eucledian one, also called inner product (see Figure 2.8), that evaluate the two points similarity as it was a ruler. It is defined by the equation:

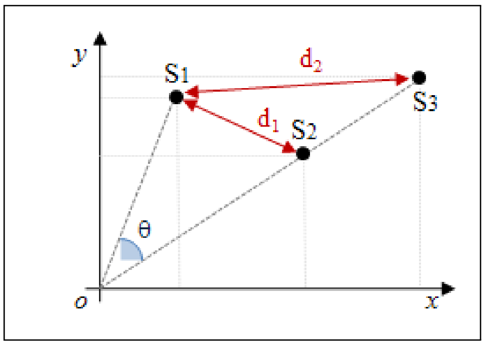


Figure 2.8: *An example of two points in the two_dimensional space $\mathcal{R}^2$, and the angle that they form.*

$$d(\overrightarrow{v}, \overrightarrow{u}) = \sqrt{\sum_{i=1}^{n} v_i u_i}$$

As shown in Figure 2.8, if the point S_2 is moved through its straight line into S_3, the distance with the point S_1 increases, from d_1 to d_2; anyway, they belong at the same straight line. Consequently, the measure is influenced by the expression level of the studied region. The cosine value is independent from this effect, and it is able to control the only similarity between the two points, evaluating the angle between the two straight lines that contain them. If the points are moved through it, the angle is the same, and the cosine measure does not change. If two bases have the same score, they will be coincident with a zero angle, and the cosine would be one. Greater is the distance between the two points, and greater is the angle between them. The cosine of the angle θ is defined as follows:

$$\cos(S_1, S_2) = \cos\theta = \frac{\overrightarrow{s_1} \cdot \overrightarrow{s_2}}{\|\overrightarrow{s_1}\|\|\overrightarrow{s_2}\|}$$

and its value fluctuates in the real interval $[-1, 1] \in \mathcal{R}$. This measure is very often used in the search engine system, to evaluate the similarity between two documents, without be influenced by their length.

All the above notions allow to define the cluster analysis for this problem. The approach starts with all the reads aligned over the genome. For each base, it computes the associated score, as explained above (see Figure 2.7). Then, it finds all non-zero regions contained in the genome. A non-zero region is a stretch of contiguous non-zero bases ($I = b_i...b_{i+m}$, $b_j \neq \overrightarrow{0}$ for each $j = i, ..., i+m$), that borders with zero-bases at each ends ($b_{i-1} = b_{i+m+1} = \overrightarrow{0}$).
The clustering phase divides each non-zero region I into a set of smaller homogeneous regions. A cluster is defined as a stretch of contiguous bases $c_i = b_i...b_{i+m}$ with an associated score, that is equal to the number and type of read aligned on them. The cluster score is the normalized sum of all distances between the points of the cluster:

$$score(c) = score(v_1, ..., v_n) = 2\frac{\sum_{i,j}\cos(v_i, v_j)}{n(n+1)} \quad 1 \leq i, < j, \leq n \tag{2.1}$$

and it fluctuates in the real interval $[-1, 1]$.

The clustering analysis system starts at time zero with a base for each cluster.

$$C = \{c_j | c_j = \{b_j\}\},\ j = i, ..., i+m,$$

and the system is defined as

$$S = \{C_k | k = 1, ..., N\},$$

where N is the number of clusters. For each group it is computed the associated score, that initially is equal to 1, because there is only an element for each set. The system energy is defined by the sum of all the cluster scores, normalized by the number of them, N:

$$E_t = \frac{1}{N}\sum_{k=1}^{N} score(c_k),$$

At each time t, the clustering process merges the nearest couple of clusters, that are the cluster that less change the system energy. The process goes on until the energy variation is limited by a certain threshold, otherwise it stops.

$$|E_t - E_{t-1}| < \tau$$

At the end, the set of computed clusters represent the homogeneous segments that are contained in the starting non-zero region. The wrong covered bases would be disjoint by the correct ones, and it would be possible to filtered them away for the further analysis.
All the obtained clusters become the nodes of the graph. Each couple of nodes that has at least a splicing read that connects them, is linked by an edge, whose weight is the number of splicing reads that join them. The graph is divided into connected components, that are subset of nodes in which exists at least a path that links each couple of them. The components represent the gene contained in the genome, and would have a high associated score, or they could be background noises, and would be composed by few nodes and would have low coverage. Each subgraph is studied to reconstruct the coding regions contained in it, or to filter the error cases. In presence of alternative splicing, a gene could assume more than one possible configuration, creating several best paths.

2.3.3 Post-processing step: gene reconstruction

All the gene structures found in the previous phase are checked to control if they satisfy the biological constraints, confirming their real nature of genes. If at most one biological constraint is not satisfied, the gene is rejected. The algorithm aims to find only good genes, applying a strict filtering policy that rejects the ambiguous and uncorrect regions.

The used biological signals are:

- *The analysis of the splicing sites.*
  If two nodes are connected by an arc, in the two linked borders there should be a splicing site definition. These are the donor and acceptor sites, that define the boundaries between coding regions and introns. As said above, there are more probable patterns than other ones, as AC-GT (canonical motif). For this purpose, only the more frequent ones are used. The splicing reads would help the splicing sites definition: if there are some inconsistency between the read alignment, this would reveal wrong or critical situations, that will not be considered.

- *The transcription starting site (TSS) and transcription ending site.*

They define respectively the starting and ending position for the transcription process The translation process translate each codon, that is a sequence of three adjacent nucleotides, into an amino acid that will compose the final protein. This suggests that the number of bases contained in the transcriptome region, from the TSS site to the end without the introns, should be a multiple of three.

All these constraints are controlled, and an operation of border correction is performed to adjust the genes with respect the biological rules. All the genes that do not satisfy at least one constraints are rejected. At the end of the process, it is returned a list of probable genes, with the correct structure.

2.4 Results and considerations

The tool has been tested on *Vitis vinifera* genome. The transcriptome of leaf and root has been sequenced with SOLiD/Applied Biosystem and Illumina/Solexa. The first one produced about 26 and 20 millions of reads respectively for root and leaf, while with the second one were obtained about 39 and 34 millions of reads respectively for root and leaf. All sequences have been aligned using PASS tool [27], as reported on Tables 2.1, 2.2 and 2.3. The produced Solexa reads aligned better than the SOLiD ones. It is important to remember that this data were produced in the 2008, and that SOLiD was just commercialized.

	SOLiD/Applied Biosystem		
	Aligned reads	*Number of reads*	*% Alignments*
ROOT_WTA	2, 169, 275	25, 857, 577	8.39
LEAF_WTA	2, 741, 068	19, 655, 622	13.95

Table 2.1: *In the table there are the results obtained for leaf and root with SOLiD/Applied Biosystem sequencing. There are the number of aligned reads, the number of total produced reads, and the alignment percentage.*

The testing phase was performed running the program on the *Vitis vinifera* genome, using the leaf and root reads, and some EST alignments. The produced set of genes was compared with the gene prediction version v1, produced by the "*Vitis v.* genome analysis consortium". This was the best available prediction for the organism. The specificity and sensibility value were used to determine the correctness and accuracy of the method. The sensibility defines the percentage of true predicted genes that were found over all real ones:

	Illumina/Solexa		
	Aligned reads	*Number of reads*	*% Alignments*
s_1_sequence.txt	2,669,028	3,053,393	87.41
s_2_sequence.txt	2,819,402	3,163,511	89.12
s_3_sequence.txt	2,819,294	3,160,627	89.2
s_4_sequence.txt	2,757,313	3,101,816	88.89
Total	11,065,037	12,479,347	88.67
s_1_sequence.txt	2,766,711	3,170,119	87.27
s_2_sequence.txt	2,794,762	3,187,711	87.67
s_3_sequence.txt	2,755,649	3,148,048	87.54
s_4_sequence.txt	2,634,039	3,102,762	84.89
s_6_sequence.txt	2,697,526	3,110,350	86.73
s_7_sequence.txt	2,653,092	3,120,558	85.02
s_8_sequence.txt	2,718,824	3,149,248	86.33
Total	19,020,603	21,988,796	86.50

Table 2.2: *In the table there are the results obtained for leaf with Illumina/Solexa sequencing. There are the number of aligned reads, the number of total produced reads, and the alignment percentage.*

$$Sn = \frac{TP}{TP + FN}$$

The specificity is the percentage of true positive predicted genes over the whole obtained by the method.

$$Sp = \frac{TP}{TP + FP}$$

Both the measures are very important to determine how many true and false positive genes can be detected by the tool. Figure 2.9 shows that the specificity is quite high for all the experiments. As mentioned in the above discussion, not all genes could be predicted because generally not all of them are expressed during the sequencing. So, the obtained results seem to be very interesting. On the other side, the computed sensibility value is quite

	Illumina/Solexa		
	Aligned reads	*Number of reads*	*% Alignments*
s_7_sequence.txt	4,292,061	4,886,384	87.84
s_8_sequence.txt	4,110,733	4,691,451	87.62
Total	8,402,794	9,577,835	87.73
s_1_sequence.txt	3,580,690	4,096,989	87.40
s_2_sequence.txt	3,658,856	4,179,039	87.55
s_3_sequence.txt	3,648,523	4,189,381	87.09
s_4_sequence.txt	3,597,991	4,136,629	86.98
s_6_sequence.txt	3,724,635	4,245,154	87.74
s_7_sequence.txt	3,698,637	4,235,932	87.32
s_8_sequence.txt	3,715,641	4,242,177	87.59
Total	25,624,973	29,325,301	87.38

Table 2.3: *In the table there are the results obtained for root with Illumina/Solexa sequencing. There are the number of aligned reads, the number of total produced reads, and the alignment percentage.*

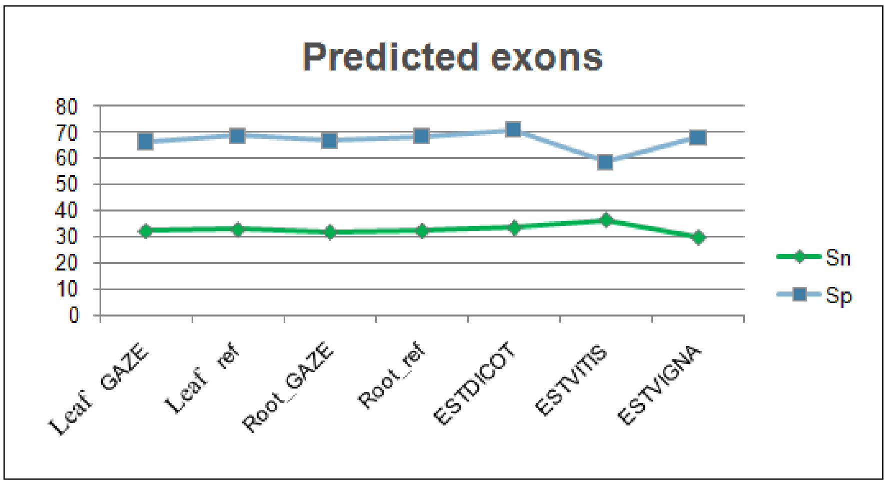


Figure 2.9: *In the graph there are the sensibility and specificity results obtained with the new prediction program.*

low, and this suggests that the tool predict a lot of false positive genes. Unfortunately, this is a problem because the starting purpose was to predict only the safe ones. This should be due to the presence of a great background noise, that is not easy to discriminate from the low expressed coding regions. Moreover, sometimes there could be pseudo genes, that are dysfunctional relatives of known genes, that have lost their protein-coding ability, or that are otherwise no longer expressed in the cell. They could be similar to still expressed ones, and they introduce two similar sequences in the genome, with the consequent alignment of the reads both in the correct and pseudo position. During the prediction phase, both of the positions are detected, but the last one is a false positive, and it increases the number of noises, decreasing the sensibility. The idea to solve this problem is to rank all genes using coverage value, and to filter all too low expressed ones. This would reject a lot of good results, but it could allow to reach a better sensibility.

3 Genome Assembly e Scaffolding

The assembly, and in particular the genome scaffolding, has been the second project in which have been involved. This issue has always been very complex (Section 3.1), and it has become even more challenging with the introduction of the next generation sequencing. The usual assembly techniques could not be used, and new ones should be found, that were able to work with large amount of data in few times and low cost. The state of the art in the 2009 (Section 3.2) provided a set of programs that did not satisfy the above requirements. This suggested the implementation of a new approach, called ConSort, that performs a good assembly, taking advantages from the available tools and overcoming their limits. The idea and the designed algorithm are described in Sections 3.3, 3.4 and 3.5.

3.1 Assembly and Scaffolding problems

The current sequencing technology can only produce little pieces of a sequence in contiguous and short reads, that can be put together to reconstruct the starting fragment. This process is called *assembly*.

Genome and Transcriptome assembly

The assembly could be referred to *genome* and *transcriptome assembly*. Even if the process could seem the same, and int both cases there are a set of sequences to map for the reconstruction of the original one; however, they have different aspects and characteristics. At first sight, transcriptome assembly is more computationally tractable than genome one, due to the smaller size of the target. Nevertheless, the different expression level of the transcriptome during the sequencing, that depends on the initial state of the used tissue, complicates the discrimination between background noise and correct reads. Here, it will be introduced the genome assembly problem, and the applied approaches for the creation of the best consensus sequence in few time, with low cost, and without introducing errors. For a more detailed explanation of the transcriptome one see *Birol et. al* 2009 [28] and *Kumar et al* 2010 [29].

As regards the genome assembly can be divided into two approach: *comparative* and *de novo assembly* [30].

Comparative and *de novo* assembly

The first approach uses the sequence of an organism, closely related to the considered one, as a guide during the reconstruction process. It is a much easier task than the other one, because it is sufficient to align the set of reads on the reference in order to characterize a newly sequenced organism. It is very often used in resequencing projects, where a reference genome is available, but it is not applicable to organisms that are sequenced for the first time, for which it is required a *de novo* approach. The aim is the genome reconstruction without consultation of previously resolved sequence.
Mathematically, the *de novo* genome assembly problem can be proved to be difficult, falling within a class of problems (NP-hard) for which no efficient computational solution is known [31]. In this situation, an heuristic approach is the only method that allow to speed up the process, to find a good enough solution, without requiring an impractical exhaustive search.
The comparative and the *de novo* approaches are not exclusive: even if a reference genome is available, some regions that differ significantly from it (e.g. large insertions) can only be reconstructed through *de novo* assembly. In the next sections, it will be discussed only this last class of methods.

De novo problems

The *de novo* genome assembly problem is like a jigsaw puzzle, where each read is a piece, and the final aim is to correctly position them. Unfortunately, it is a more complicated task, because the pieces, could contain sequencing errors, polymorphic situations (like SNPs), could be placed in more than one position (repetitive regions), or could be contamination of other organisms, and it is not easy or sometimes even impossible to correctly place them.
The highlight problems and limitations could be solved as follow:

- **Sequencing errors**
  Very frequently the sequencer could wrongly identify several bases of a read, inserting, deleting or substituting some of them. To obtain a high-quality assembly, overcoming this problem, it is useful to apply a redundant coverage, in which on average every nucleotide is sequenced many times. The probability to have an error on the same nucleotide is generally low, and more reads that confirm it will give a quite sure consensus.

- **Repeated region**
  They are similar or sometimes equal stretches of genome, that mislead the mapping of the reads, causing multiple alignments, for which it is not easy to find the best position. Generally, the assembly stops

on them, producing a set of disconnected pieces, divided by undefined gaps. The repeated region has always been the central challenging problem. Without repeats and with a good coverage value, an accurate assembly could be created in few time and without errors.

The magnitude of the problem depends on the used sequencing technology, that determines the length of the produced reads. If they were one base long, each of them would be repeated. On the other hand, if they could be long as the chromosome from one end to the other, repeats would pose no problem at all. In the middle, there are all the other sizes. The fraction of unique sequences increases with the read length, until eventually each of them is unique. If DNA fragments in the genome were random (which they are not), then the expected number of occurrences for any read would decrease exponentially as its length increasing, dramatically reducing the number of repeats in the genome. However, the complexity of real organisms makes some regions nearly impossible to be correctly assembled, with short reads. Consequently, the sequence length play a central rule in this problem.

If the produced reads are shorter than the length of the repeats, then all of them are not able to entirely span such regions. In this case the assembler program generate a very fragmented genome sequence, that can complicate the further analysis, as the gene prediction and genome annotation. In fact, in a fragmented situation, the genes can be divided into disjoint scaffolds, or they could be missed in the repetitive regions. The problem could be partially solved using paired-ends, pairs of reads with a constraint on their relative orientation and distance in the genome (see Section 1.6). Paired-ends allow to jump the repeats, join together separated stretches of bases, and help the gap filling. The feasibility and success of this technique depends on the insert average size. If it is lower than the gap size, a read could fall on it without joining together the fragments.
Even with paired-end approach it is difficult to solve the repeat problem, and to reconstruct these regions. In fact, the assemblers collapse them in unique fragments, reducing or even losing the genomic complexity.

- **Contamination problem**
  Sometimes the sequenced DNA could be contaminated by others organisms and it could be important to discriminate them before assembly. Even this task depends on the read length and it becomes particularly problematic when they are too short.

The above discussion has introduced all the problematics involved in the assembly, and the difficulty in the development of a good strategy. The most used technique is the concatenation of the reads with the longest overlap,

Contig

to form a contiguous sequence, called *contig.* This process is iterated more and more, each time merging the sequences with the longest overlap, until all of them are used. At the end, a set of disjoint contigs is created, whose position over the genome is not defined. As mentioned before, their length and correctness depend on many factors: the complexity of the genome, the number and length of the repeated regions, the quality of the produced reads, the amount of sequencing errors, the produced coverage and finally the used assembler. An assembler could be conservative at all, stopping in ambiguous situation and producing shorter, but more correct contigs. At the other extreme, it could solve critical points creating longer, but probably misassembled contigs.

Scaffolding

The created assembly is always far away to be the original genome sequence and a finishing step is required to create the chromosomes. The scaffolding process is used to find the correct order and orientation between all contigs, creating *scaffolds* (or sometimes called supercontigs or metacontigs). They are characterized by a set of ordered and oriented contigs, with stretches of N's that define the gaps between them. This step is very important to close the gaps, and it is performed using the paired-ends or genetic maps information.

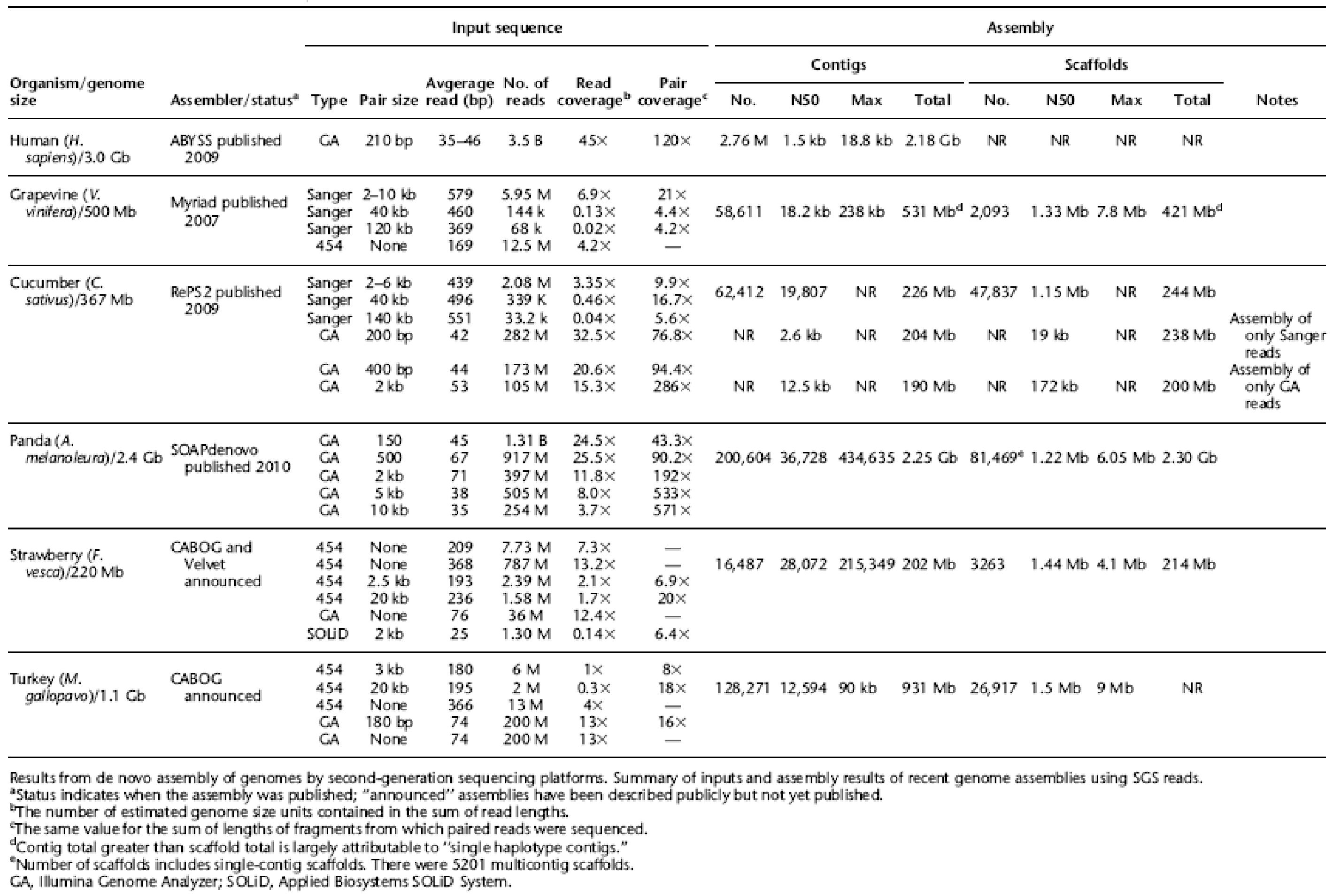
Sequencers and assemblers

The Sanger method is able to produce very long reads, that could be assembled applying overlapping techniques and is proper for the repetitive regions resolution. Anyway, it is too expensive and it requires a lot of time without not allowing to sequence all genomes, even very large. On the other hand, the next generation sequencers seem to overcome this limitation, generating a large amount of data in few time and reducing cost. Unfortunately, the produced reads are too short for the existing *de novo* techniques and repetitive problems: new approaches should be designed to solve this limitation and open the sequencing era to new organism.

Sequenced genome

From the early 90's, some small and large genome have been sequenced, initially with Sanger methods and then applying Next Generation Sequencers. In the 1995, the first bacterial genome was published (*Haemophilus influenzae*) with Sanger technology [33]. It requires 24,304 reads of an average length around 460bp. In 2001, the Human Genome Sequencing Consortium published the human genome project [34]. It required about 30 million reads, with length up to 800 bp, using Sanger sequencing technology and automated capillary sequencers. Other sequencing projects were published using this technique, but very often all the effort was moved on next generation sequencing approach.

The initial assembly consisted on mapping short reads to a reference genome. James Watson [35] genome was sequenced with 454 unpaired reads, while some individuals of African [36], Asian [37], and Korean [38] were all sequenced with Illumina. A combined approach was used to sequence the draft assembly of grapevine [39] (*Vitis vinifera*, genome size about 500 Mb),



Organism/genome size	Assembler/status[a]	Input sequence						Assembly								Notes
								Contigs				Scaffolds				
		Type	Pair size	Avgerage read (bp)	No. of reads	Read coverage[b]	Pair coverage[c]	No.	N50	Max	Total	No.	N50	Max	Total	
Human (*H. sapiens*)/3.0 Gb	ABYSS published 2009	GA	210 bp	35–46	3.5 B	45×	120×	2.76 M	1.5 kb	18.8 kb	2.18 Gb	NR	NR	NR	NR	
Grapevine (*V. vinifera*)/500 Mb	Myriad published 2007	Sanger	2–10 kb	579	5.95 M	6.9×	21×									
		Sanger	40 kb	460	144 k	0.13×	4.4×	58,611	18.2 kb	238 kb	531 Mb[d]	2,093	1.33 Mb	7.8 Mb	421 Mb[d]	
		Sanger	120 kb	369	68 k	0.02×	4.2×									
		454	None	169	12.5 M	4.2×	—									
Cucumber (*C. sativus*)/367 Mb	RePS2 published 2009	Sanger	2–6 kb	439	2.08 M	3.35×	9.9×	62,412	19,807	NR	226 Mb	47,837	1.15 Mb	NR	244 Mb	
		Sanger	40 kb	496	339 K	0.46×	16.7×									
		Sanger	140 kb	551	33.2 k	0.04×	5.6×									
		GA	200 bp	42	282 M	32.5×	76.8×	NR	2.6 kb	NR	204 Mb	NR	19 kb	NR	238 Mb	Assembly of only Sanger reads
		GA	400 bp	44	173 M	20.6×	94.4×									
		GA	2 kb	53	105 M	15.3×	286×	NR	12.5 kb	NR	190 Mb	NR	172 kb	NR	200 Mb	Assembly of only GA reads
Panda (*A. melanoleura*)/2.4 Gb	SOAPdenovo published 2010	GA	150	45	1.31 B	24.5×	43.3×									
		GA	500	67	917 M	25.5×	90.2×	200,604	36,728	434,635	2.25 Gb	81,469[e]	1.22 Mb	6.05 Mb	2.30 Gb	
		GA	2 kb	71	397 M	11.8×	192×									
		GA	5 kb	38	505 M	8.0×	533×									
		GA	10 kb	35	254 M	3.7×	571×									
Strawberry (*F. vesca*)/220 Mb	CABOG and Velvet announced	454	None	209	7.73 M	7.3×	—									
		454	None	368	787 M	13.2×	—	16,487	28,072	215,349	202 Mb	3263	1.44 Mb	4.1 Mb	214 Mb	
		454	2.5 kb	193	2.39 M	2.1×	6.9×									
		454	20 kb	236	1.58 M	1.7×	20×									
		GA	None	76	36 M	12.4×	—									
		SOLiD	2 kb	25	1.30 M	0.14×	6.4×									
Turkey (*M. gallopavo*)/1.1 Gb	CABOG announced	454	3 kb	180	6 M	1×	8×									
		454	20 kb	195	2 M	0.3×	18×	128,271	12,594	90 kb	931 Mb	26,917	1.5 Mb	9 Mb	NR	
		454	None	366	13 M	4×	—									
		GA	180 bp	74	200 M	13×	16×									
		GA	None	74	200 M	13×	—									

Results from de novo assembly of genomes by second-generation sequencing platforms. Summary of inputs and assembly results of recent genome assemblies using SGS reads.
[a]Status indicates when the assembly was published; "announced" assemblies have been described publicly but not yet published.
[b]The number of estimated genome size units contained in the sum of read lengths.
[c]The same value for the sum of lengths of fragments from which paired reads were sequenced.
[d]Contig total greater than scaffold total is largely attributable to "single haplotype contigs."
[e]Number of scaffolds includes single-contig scaffolds. There were 5201 multicontig scaffolds.
GA, Illumina Genome Analyzer; SOLiD, Applied Biosystems SOLiD System.

Figure 3.1: *The de novo assemblies of next generation sequencing projects (Schatz M. C. [32]).*

combining the Sanger method with the 454 reads. While, the draft genome sequence of cucumber [40], *Cucumis sativus*, was obtained using a combination of Sanger and Illumina sequencing.
The first *de novo* assembly exclusively NGS was recently performed on the large genome of giant panda (*Ailuropoda melanoleura*), by the Beijing Genome Institute [41]. It has been used only Illumina reads averaging 52 bp and was done with the SOAPdenovo assembler. Finally, the last two published genomes have been the genome of woodland strawberry [42], *Fragaria vesca*, sequenced using Roche/454, Illumina/Solexa and SOLiD/Applied Biosystem, and the genome of *Theobroma cacao* [43], sequenced with Roche/454, Illumina/Solexa and Sanger sequencers.
See Figure 3.1 for a brief summary about *de novo* short read assemblies.

In the next section, there will be an introduction about the basic assembly approaches, and the current state of the art. Although, even if a lot of work has been done yet, both at sequencing and assembly level, it is clear that genome assembly is not a solved problem, and older technologies, like Sanger, have not to be dismissed.

3.2 Assembly techniques

The assembly techniques are divided into three categories: *Greedy Graph-based*, *Overlap/Layout/Consensus (OLC)*, and *de Bruijn Graph*. All of them create a graph (see Section 2.3) using the reads or k-mers (extracted from them), and reconstruct the assembly searching for the best path. This kind of problem is made solvable in polynomial time through the application of *ad hoc* heuristics or approximation algorithms.
As mentioned in the previous sections, the reads produced with next generation sequencers are very short (about 30-100 bp), and contain sequencing errors, or can belong to polymorphic and repetition regions. These characteristics result in the formation of particular structure in the graph, that are labeled as (see Figure 3.2):

- **Spurius**: the path is divided into two alternative ones, a short and long one, caused by sequencing errors toward the end of the read;
- **Bubbles**: the path diverges and then converges again, due to the errors or polymorphisms contained inside the assembled reads;
- **Frayed rope pattern**: two paths converge and successively diverge, because of the repeats within the target genome;
- **Cycles**: the path converges on themselves, when there are repeated regions in the target, as short tandem repeats.

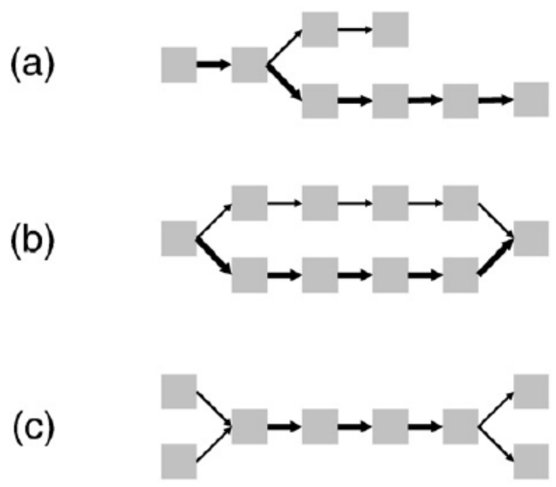


Figure 3.2: *A representation of all possible problems in short read assembly. (**a**) "Spurius": the main path is divided into two alternative paths; (**b**) "Bubbles": the main path diverges and converges; (**c**) "Frayed rope pattern": paths that converge and then diverge.*

All the previous situations can occur together, creating very complex graph structure, that are often very difficult to solve without the introduction of artifacts and errors. The goodness of an assembly is measured computing its size and accuracy. The first one is usually given by statistics including N50 value, maximum, average and combined total length:

Assembly quality measure

$$N50 = l_i : \sum_{j=N}^{i} l_j = L \cdot 50\%, l_1 \leq l_2 ... \leq l_N$$

where l_i is the length of the i contig, N is the total number of contig, and L is the size of the genome. The accuracy is often given by the degrees of mate-constraint satisfaction and violation.

3.2.1 Greedy Graph-Based Approach

In general, a greedy approach is based on the assumption that at each stage it is necessary to make the locally optimal solution with the aim to arrive at the global one. Not all problems could be solved using this approach, because there are situation in which the global optimal solution could not be reached, falling in local suboptimal points.

The basic idea is to concatenate reads or pieces of yet assembled sequences that overlap, to reconstruct the original genome sequence. This means that the prefix of a read shares sufficient similarity with the suffix of another one. The basic operation is repeated over and over again, until not other overlaps exist, and the sequencing process is finished. In a greedy approach the next overlap to consider is defined by the next-highest scoring one. This value is given by the length of the common region and its identity level (percentage of base pairs shared by the two reads). As said before, this strategy could not always reach the optimal global solution. In fact, it could happen that a read is used to create a contig, but it could have helped other ones to grow even larger, and so the algorithm stops at a local maximum. For example, a greedy approach may misassembled repeated regions.

Some programs based on this method are:

- **SSAKE**, *Short Sequence Assembly by progressive K-mer search and 3' read Extension* [44] (2006).
  It was the first assembler developed to work with short-reads. It was able to assembly millions of unpaired sequences of uniform length, using a prefix tree, on which the longest possible overlaps between two of them was searched. It uses the high-coverage to check and solve sequencing errors contained toward the end of the read.

- **VCAKE**, *Verified Consensus Assembly by K-mer Extension* [45] (2007).
  It works very similar to SSAKE, but it makes significant improvements in handling error, using high depth of coverage.

- **SHARCGS**, *SHort-read Assembler based on Robust Contig extension for Genome Sequencing* [46] (2007).
  It operates on uniform-length, high-coverage, and unpaired short reads, with high accuracy and speed. SHARCGS is more robust in hardly errors, and virtually never generates misassemblies. In a preprocessing phase, all reads are filtered by quality, to remove all the sequencing errors. Then, the reads are assembled with a greedy approach similar to the one explained above, and the produced contigs are merged in the post processing step, to create the final assembly.

3.2.2 Overlap/Layout/Consensus (OLC) Approach

The Overlap/Layout/Consensus (OLC) approach was designed to work with large genomes and long sequences, like Sanger or 454 reads. Some example of software that develop this strategy are Celera Assembler [47], Arachne [48, 49], CAP and PCAP [50], Newbler [51], CABOG [52, 53] and Edena [54].

In an overlap graph, the nodes are the reads and the arcs are their overlaps. The paths are the potential contigs. If there were no sequencing errors and

too long repetitive regions, the graph will have an unique path, that will be the final chromosome.
Three operations are used in an overlap graph:

1. *Overlap computation*
   The overlaps must be pre-computed, requiring a series of computationally expensive all-against-all pair-wise sequence alignments. For a more efficient execution, it is used a seed to find all similar keywords contained in the reads. When an overlap is discovered, and a minimum percent of identity is satisfied, an extension phase tries to find if the sequences align.

2. *Layout stage*
   The overlap graph is created, and this leads to an approximate read layout. This could require a large amount of computer memory to store all the nodes and edges.

3. *Consensus stage*
   The most probable consensus is searched moving through the graph, performing the contig creation. This step can be parallelized: each connected components of the graph (that is each contig) can be separately processed, reducing the required computational time.

The most used and well-known OLC programs are:

- **Newbler** (454 Life Sciences).
  It is the most used assembler program. It is able to create very long and error-free contigs, using unpaired reads of approximately 100-400 bp. In the first phase it generates unitigs from reads. Initial miniassembly uses good overlaps, and it is the starting point for the next assembly, that merges the overlapping unitigs into contigs. It is able to build scaffolds from paired-end data.

- **CABOG**, *Celera Assembler with the Best Overlap Graph* [52] (2008).
  It is a revised pipeline of the Celera Assembler, modified for combination of ABI 3730 and 454 FLX reads. It parses the native SFF files produced by the 454 FLX pyrosequencing machines, discarding 454 reads that include at least one unresolved base (the letter N). After that, it recognizes mated reads as those that contains 454 linker sequences, from which it generates one or two shorter, linker-free pseudo reads, plus a distance constraint set to the estimated mean separation.

3.2.3 The de Bruijn Graph Approach

The OLC methods cannot be used with short read: all the attempts done to use the existing assemblers with this new kind of data failed or performed very poorly. The main cause is the request of a minimum read length, or a minimum amount of the overlap that is too long for short reads. Moreover, the huge amount of produced data required a lot of memory and computational time to work with OLC existing programs. It was necessary to develop a new technique that would be able to work with short reads and perform very well.

The de Bruijn graph (DBG) approach is widely applied to the short reads (Solexa and SOLiD) assembly problem. The method has been developed outside the biology field. The original aim was to represent strings from a finite alphabet, where the nodes are all the strings of a fixed-length, and the edges are the perfect overlaps between them. In the assembly problem, it is used a variant of the de Bruijn graph, called k-mer graph. All the words of fixed-length k that are found in the input reads become the nodes of the graph, and the edges are their fixed-length overlaps (see Figure 3.3). The graph contains a path that corresponds at each original sequence that has produced it. In an ideal scenario without errors and repeats, the k-mer graph would contain an unique Eulerian path, that visits each edge exactly once. In the reality, it is more complicated, and it can contain cycles and alternative paths, that have to be evaluated for the creation of the more probable assembly. The main factors that complicate the k-mer graph are:

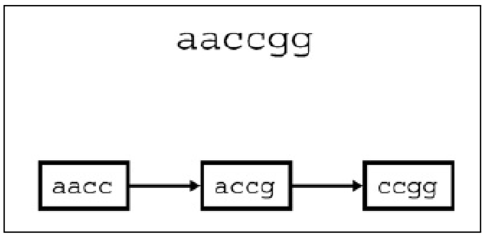


Figure 3.3: *A starting sequence and the associated k-mer graph, with $k = 4$.*

1. *DNA is double stranded*
   Each sequence could be represented in the forward or reverse form, requiring a double quantities of nodes to manage them. This suggest to use both forward and reverse k-mer in the same nodes.

2. *Repeats*
   There are a lot of different kind of repeats, like tandem, inverted, imperfect repeats. They introduce cycles in the k-mer graph, complicating its structure.

3. *Palindrome*
   A Palindrome is a sequence that is its own reverse complement. This kind of sequence induce paths that fold back on themselves.

4. *Sequencing errors*
   The errors could introduce alternative paths or internal bubbles.

The de Bruijn graph has very interesting properties that make it very attractive for assembly problem with a large amount of short reads. The k-mer graph requires a limited amount of memory, that is independent from the number of used reads. Moreover, the method does not require all-against-all overlap discovery, and it compress redundant sequences.

Only the assembly programs based on de Bruijn graph have demonstrated the ability to assembly very short reads. The best assemblers that apply this approach are: Euler [55, 56, 57] and all its variants like EULER-SR [58] and EULER-USR [59], Velvet [60], ABySS [28, 61], AllPaths [62, 63] and SOAPdenovo [64].

- **Euler** software.
  It was firstly developed to work with Sanger sequences, and then was modified to work even with short reads, initially Roche/454 , then also Illumina/Solexa and SOLiD/Applied Biosystem. The method is so called by the Eulerian approach on which it is based. It starts with a filtering phase in which it tries to solve all sequencing errors, filtering the reads with low probability. Then, it works on the created graph, solving the still present sequencing errors and repeat situations.

- **Velvet**.
  It is able to manipulate de Bruijin graph efficiently both to eliminate errors and solve repeats. It starts with the construction of the graph, using hash-table structure to speed up the operations, and to reduce the computational time.

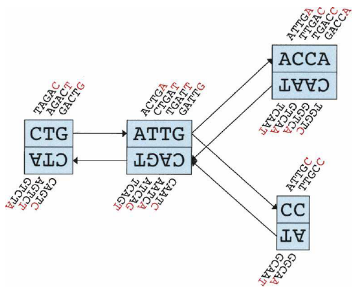


Figure 3.4: *A schematic representation of the de Bruijn graph implemented in Velvet assembly (Zerbino D. R. [60]).*

It provided a graph such as that one shows in Figure 3.4. After that, it executes a simplification phase, on which it simplify the graph without loss of information. The basic idea is to merge together all the adjacent nodes that have exactly one outer and one inner edge. Then, the final phase corrects all errors and solves all repeats that are present in the graph, creating an eulerian path.

- **ABySS**, *Assembly By Short Sequencing.*
  It is a parallel assembler for short reads. It is able to work with a limited memory allocation and few computational time, thanks to the parallelization of the operations into a multi-threading execution. It develops a simplification of Euler and Velvet graph.

- **AllPaths**. It has been developed to work with Illumina/Solexa paired-end reads. It is organized into three steps: first of all, it performs a preprocessing phase, on which it tries to solve sequencing errors present in the k-mer, using a filter based on quality and k-mer frequency. Then, it construct all possible "unipaths", computing the perfect overlaps between the reads, seeded by k-mers. Finally, it creates the k-mer graph, and it starts the problem solution, applying heuristics, to find which is the more probable path.

- **SOAPdenovo**.
  It is a new implementation of the yet existing SOAP program, for short read assembly. It works like all the other ones based on de Bruijn approach, and it tries to solve sequencing errors and repeats, constructing the most probable assembly.

3.2.4 Scaffolding approach

In the above section, all the assembly methods and the developed programs, that works mostly with unpaired short reads, have been described. Their idea is to start from a large amount of reads, produced by the next generation sequencers, and to find the best overlap between them.
The scaffolding approach is a finishing method to complete the assembly. The basic idea is to start from a set of pre-assembled reads, and to use the paired-end or mate-pairs to find the right order and orientation of the contigs, creating the final scaffolding.

- **Bambus** [65].
  It is a flexible scaffolder, that works with different kind of sources, like mate-pairs or paired-end sequences, but even other information coming from independent mapping. It is able to manage large genome. It is a graph-based approach, on which the contigs are the nodes of the graph, and the paire-end are the edges, that link adjacent contigs. It is organized into two phases:

  - it orients all contigs, finding a consistent orientation between them, constrained by the mate-pair information. It is important to solve all the inconsistent situation that could happen, that can be caused by erroneous constraints or by misassembled contigs.
  - it orders all the oriented contigs, finding their correct position in the scaffolds. It tries to solve bubbles and cycles caused by background noises and repeats, to find the more probable scaffolding.

- **SOPRA** [66].
  It is a scaffolding algorithm for paired-end reads, based on statistical optimization. It starts with a pre-processing phase, on which it tries to check the consistency of the input contigs, using statistical approach. After that, it creates the graph and it tries to find the best scaffolds.

3.2.5 Other software

Other programs have been developed to work with short reads, and to help the assembly process, like EagleView and MAIA.

- EagleView [67].
  It is a genome assembly viewer for next generation sequencing technologies. The aim was to develop a very efficient tool, able to visualize a large amount of short reads, like Consed [68] for Sanger sequences. The viewer would help the assembly problem resolution, showing how the reads overlaps between them, and allowing to hypothesize possible assembly solutions.

- **MAIA** [69], *Multiple Assembly IntegrAtor.*
  The program is a genome assemblies integrator. The basic idea is that each assembler developed a different technique, and it could have disadvantages and advantages, different from other approaches. So, if severals assemblies obtained with different programs are integrated, it is possible to take advantage from their positive characteristics and perform a better final assembly.

3.3 ConSort: a new scaffolding approach

Without errors or repeated regions, the assembly process will generate the original genome sequence, starting from a set of reads. Unfortunately, this situation is very far from the reality, and the assembly produces a set of contigs, whose position over the genome is undefined. To find their correct order and orientation, it is necessary a scaffolding phase. A lot of very

efficient programs have been designed and implemented to assembly large amount of short reads; although, few of them perform the scaffolding phase, and none returns a finished assembly. Nowadays, this is realized through laboratory experiments: it would be helpful to have an efficient, automated and robust scaffolding tool.
A set of contigs and their relationships can be used as starting point for a scaffolding phase. The first one is produced by the existing assemblers, while the second kind of information can be extracted by mate-pair and paired-end libraries.

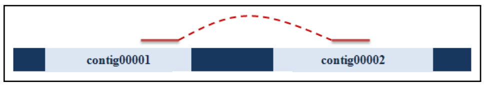


Figure 3.5: *A schematic representation of the genome, with the contigs and the paired-end mapped on them. The paired-end links together the two contigs, defining the order and orientation.*

A paired-end is a couple of reads that constrains two genome points, defining their relative position and distance (see Section 1.6). If the paired-end reads are mapped on the set of contigs, they will give the information about their connections (see Figure 3.5), order and orientation. In an ideal situation, on which all contigs are linked together without errors, the scaffolding problem is not difficult to perform. Starting from a contig, and following all the connections, it will create the final sequence. However, in the reality there are wrong connections or lack of information that complicate the problem. This is solvable using a large amount of links that confirm the connections and discriminate the background noise. Next Generation Sequencers are able to produce a lot of paired-ends in quickly few time and inexpensively.

This method works with any kind of starting contig sets, and paired-end libraries; anyway, it is suggested to apply the following strategy:

- contigs: sequenced and assembled respectively with 454 and Newbler. They will be longer and more error-free than the ones obtained with other technologies. The length of the produced reads helps to avoid misassemblies due to low complexity or repetitive regions. Moreover, Newbler has been designed *ad hoc* to work with 454 reads and correct its possible errors.

- mate-pair library: sequenced with SOLiD (or Illumina).
  For this application, the read length is less important than the amount of produced data. The SOLiD/Applied Biosystem technology, with the color space implementation, performs a double control on the produced bases (see SOLiD properties explained in Section 1.4), and provides a better sequencing errors control.

The described idea is developed in a new scaffolding tool, **ConSort**, *Contig Sorting*. It is a set of algorithms for the genome scaffolding and the reconstruction of the “original” genome sequence. It starts from a set of contigs, generally obtained assembling next generation sequencing data, and from a set of mate-pair or paired-end libraries. It checks the contig consistency, it filters paired-end errors, and it orders and orients contigs into the final scaffolds, discriminating from background noise. With ConSort, we want to demonstrate that the scaffolding approach, starting with 454 contigs and SOLiD parired-end libraries, is a nice method to achieve good results.

3.4 ConSort requirements

The development of ConSort started with the analysis of the compulsory and desired requirements, suggested by the study of all the problems to solve and functions to realize. They are divided into implementation and functional requirements.

The first group defines the properties of execution and the characteristics of implementation that the tool should have:

- **Low memory and CPU usage**
  ConSort will work with a lot of data, coming from contigs and paired-end libraries, and it is important to avoid the two typical bottlenecks: the use of a huge amount of RAM memory, that is not always available, and the large computational time. To satisfy the requirements, it was designed a data structure that can be stored in memory, even for desktop computer, and can be easily use with low simple basic operations, without requiring too much computational time.

- **Easy maintenance: modular structure**
  The program could be modified and updated: some new features should be added, deleted or substituted. This should be realized fast and without changing the whole code.
  A modular architecture satisfies this requirements, allowing an easily maintenance of the code. ConSort functions are organized into modules: each of them realizes a specific task, and all together they obtain the scaffolding. Each module is like a “black-box”: the performed operations are invisible outside, and only the input and output format are known. All the modules are organized into a pipeline, and they communicate and share data using well designed interfaces. So, the changing of a module, satisfying the defined data format, does not compromise the functioning of the other ones.

- **Expandable: easy update**
  A lot of new information could be produced, like information coming from new sequencers, or other source of evidence. The program should

be able to adapt to the changes and use new kind of information, improving the performance. As explained above, the modular structure satisfies this requirement.

- **Robustness: conservative but innovative**
  It is to decide the correct balance between a conservative method that does not introduce errors, stopping in ambiguous situation, and an innovative one that solves the problems, with the risk of error introduction. ConSort should be able to work out ambiguous situation, deciding which is the more probable solution where possible. All the taken decisions are stored in a output file, that can be used for a manual validation of the produced output. This allows not to lose information.

The second class of requirements contain all features and functions that should be performed by ConSort :

- **Input data**
  It uses Newbler assembly and SOLiD paired-end format, and it can be easily updated to work with Illumina/Solexa data. A parsing module designed on this file format will recognize and extract the required information.

- **Output**
  The output would be the file with the scaffolding sequence and the list of all used information and taken decision.

- **Contig consistency**
  To produce a consistent assembly, it is important to correct the starting data: misassembled contigs. A consistency module will integrate all the available information to revise the wrong contigs.

- **Paired-end library elaboration**
  Even the paired-ends can contain wrong bases, and it is necessary to realize a package of modules that pre-elaborate them, filtering all errors and computing the library insert length. This estimation could be easily obtained looking at the library preparation phase, but in this case only a range of values can be derived, therefore it is preferable to extract them from the paired-end analysis.

- **More than one input library**
  ConSort should be able to work with more than one paired-end library, because of the advantage to work with different insert length, that can solve various gap sizes.

- **Scaffolding**
  This is the main requirement: ConSort should order and orient all

contigs in the final scaffolds, evaluating the distance between them. All the other functions help this step, preparing the data and information for its execution.

- **Gap filling**
  This is a fundamental function for the finishing phase: the gaps among contigs should be filled to complete the genome sequence.

3.5 ConSort algorithm

The previous requirements have been integrated and developed in ConSort: a modular structure that realizes the scaffolding process using contigs and mate-pairs libraries. It has been implemented on C++, and supported on Linux.

In Figure 3.6 is drawn the pipeline developed in ConSort, with all modules and their organization. This is a high level view, that shows the workflow and interaction among the functions, but their internal organization is hidden. This will be explained in more detailed in the next sections.
ConSort starts with a set of contigs and paired-end libraries in input, and performs the following operations:

1. **Spectral Correction**.
   It analyzes the input paired-end (or mate-pair) libraries to correct sequencing errors. So far, this module can be used for SOLiD data only.

2. **Redundancy**.
   The module normalizes the mate-pair libraries, to save only a copy for each identical paired-end type. It is very important to filter these data, to reduce background noise.

3. **Pre-Pairing**.
   In some cases, the paired-ends are made of only one end, due to sequencing errors, and they cannot be used for the scaffolding aim. For this reason, they are not useful information and could be deleted. The module realizes this function, and at the end only the paired reads are used.

4. **Trimming**.
   The paired-end reads are then filtered by quality, discarding the reads with a low global quality, or removing the low quality bases at the end of a sequence. This phase allows to reduce the background noise, and increase the number of alignments, removing the low quality bases.

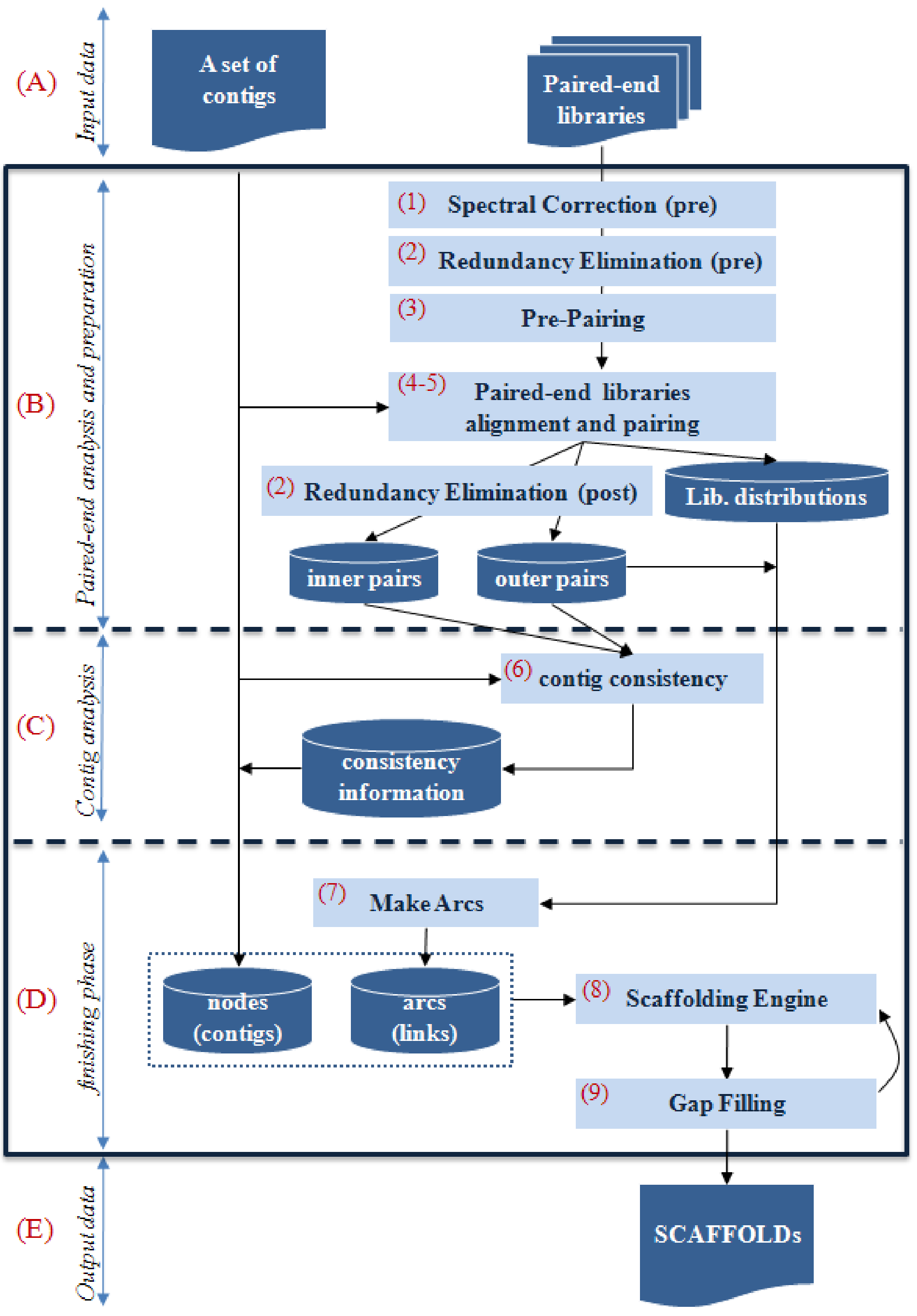


Figure 3.6: *A schematic workflow of the modular pipeline developed in ConSort. (**A**) and (**E**) are respectively the input and output data. The other points show the modules used for: the paired-end analysis and elaboration (**B**); the contig consistency (**C**); the finishing phase (**D**).*

5. **Alignment and Pairing**.
   The workflow aligns and pairs the pre-elaborated reads on the contigs, reconstructing the pairing. At the end two class of mate-pairs are produced: the inner pairs, that have both pairs of the couple aligned in the same contigs, and the outer ones, that align between two disjoint contigs. The module computes the library length distribution: the mean value and its standard deviation.

6. **Consistency**.
   The consistency analysis is performed with the information coming from the mate-pairs that aligns on the contigs. The misassembly and errors are detected, and the useful information to solve them are stored in an output file, that will be used by the scaffolding phase to correct them and produce the final assembly.

7. **Make arcs**.
   The mate-pairs are transformed into arcs. Each of them is a cluster of all links that connected the same contigs.

8. **Scaffolding**.
   This phase solves the 'tangle" situations, to perform a correct scaffolding. It applies the information extracted during the previous steps, to analyze and solve the ambiguous situations. If there are not enough evidence to untie the knots, the critical points are broken, and scaffold elongation terminates.

9. **Gap filling**.
   This is the last step toward the final scaffolding. It is applied to close the gaps and complete the genome sequence.

3.5.1 Input data

Contigs

They represent the result of the assembly of reads produced by an assembler, like Newbler. The given file should have a multiple sequence FASTA format, obtained concatenating several single FASTA files, one for each contig. This is a text-based format for storing either nucleotide or peptide sequences, in which base or amino acids are represented using single-letter codes.
The file begins with a single-line description, generally introduced by a ">" (greater-than), and it contains the sequence name (contig in this case), its length and eventually other optional information. For example, Newbler specifies the number of reads that have been used to assemble the contig. In the following lines, the stretch of letters defines the sequence data.

Paired-end (or mate-pair) libraries

ConSort works with one or more paired-end libraries. It is important to keep them divided by length, for a more accurate analysis. For each paired-end library, there are four associated files. Each read of the mate-pair has associated a couple of files: one containing the sequence of the read, the other one the quality file (multi-FASTA format).

3.5.2 Spectral Correction

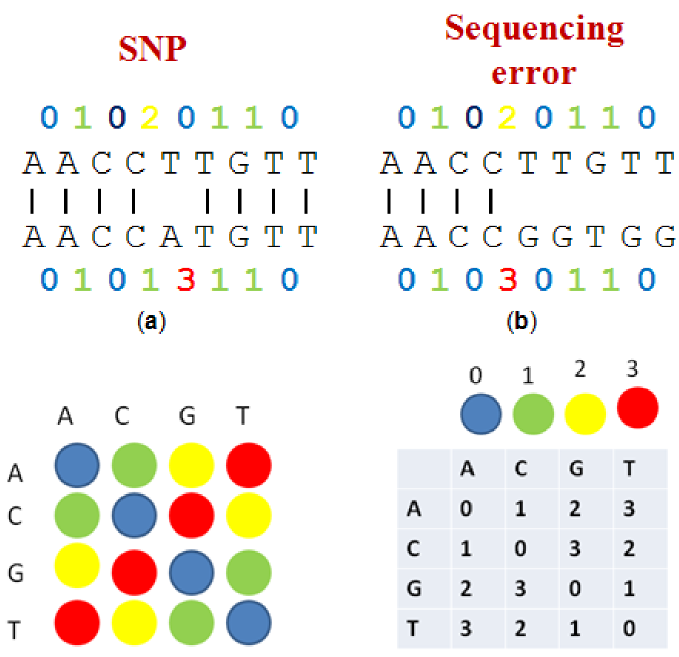
As explained in the Chapter 1, the reads produced by next generation sequencers very often contains errors. This can compromise the further analysis and increase the background noise (see Section 3.5.5 for a more detailed discussion). For example, a read with errors cannot align losing information, or it can align in wrong position and with mismatches producing wrong evidence.

An error correction phase is fundamental to reduce the number of wrong or low quality bases, and to improve other analyses. It is known that SOLiD and Illumina sequencers make substitution errors, rather than insertion and deletion one, consequently only the substitution cases are considered and solved. Moreover, with a high amount of reads each base is covered many times, and the probability that some or all sequences have an error in the same position is very low. Each read is produced independently from the others, and even the error events occur independently. Consequently, there is no reasons to have an error in one position confirmed by all the other reads.

Anyway, if the sequenced genome is heterozygous, there could be SNPs (Single Nucleotide Polymorphisms) occurrence: some position could have more than one correct base value. Consequently, it could happen that two reads align in the same position, with different values, both of them correct. In this situation, the reads would be divided into two groups: half confirms a base and the other half the other one. With the SOLiD double color encoded system, SNPs can be easily discriminated from errors. In fact, while an error in the sequence changes the correct phase of the colors, and the equivalent base sequence, a SNP changes only two colors and a base in the equivalent sequence, like in the example of Figure 3.7.

SAET (SOLiD Accuracy Enhancer Tool) [1] has been used to perform the spectral correction. It is able to increase the number of mapped reads by 40 to 50%, and to work with the genome of complex organisms. It implements a modified version of th spectral alignment error correction algorithm proposed by Pevzner (2001). It extends the original technique, introducing quality information and developing the color-space properties, as described above.

[1] ABI Applied package



	A	C	G	T
A	0	1	2	3
C	1	0	3	2
G	2	3	0	1
T	3	2	1	0

Figure 3.7: *A comparison between two reads that differ for a single position, due to a SNPs (**a**) or due to a sequencing error (**b**). A SNP changes two colors, and the produced sequences mismatch only in the SNP position, while in case on error, the two sequences are different starting from the error itself.*

System requirements: memory and computational time

The module works in a multi-threading configuration, reducing the computational time, and it does not require too much memory. In fact, it uses temporary files on which storing partial results, allowing not to store everything in memory. In the used system, the module runs in **4 hours**, using **10 processors**, and elaborating **300 million reads**. It requires about 48GB of RAM memory for 1Gbp region.

Input parameters

Type	Explanation	Format
number of CPU	the number of available CPU, to determine the parallelization level (number of thread that could be used). This value depends on the used system	integer value

Type	Explanation	Format
genome length	an approximate value of the genome length, that is used for the computation of the average coverage. This value is given by the contig assembler	integer value
sequence file	the file with the sequences to correct	multi-FASTA
quality file	the file with the qualities associated to the sequence file	multi-FASTA

Output

Type	Explanation	Format
sequence file	the sequence file with the corrected bases	multi-FASTA
quality file	the quality file with the updated qualities	multi-FASTA

3.5.3 Pre-alignment pairing

As explained in Section 1.6, a mate-pair (or paired-end) is made of two ends, that constraints two pieces of the genome with a relative orientation and separation. The name of the reads of each couple differs only for the suffix value, that allows to discriminate which part is considered. For example, for SOLiD mate-pair reads are used F3 and R3. The notation help the pairing phase: the two separated reads can be joined in the couple.

In some cases, the paired-ends are made of only one end, due to sequencing errors. Consequently, all the pairs could be divided into two groups:

- single reads: the reads that have lost their paired end. They are computationally not interesting, because they do not give pairing information.

- paired reads: the reads that are paired. These will be used for further analysis.

The module performs the pairing: it divides the paired-ends into the two groups, considering the case in which a paired-end is made of two reads, and the case in which only one end exists. This step is very useful to reduce the amount data to elaborate, discarding all of them cannot be used for the further analysis. In fact, the scaffolding task works with paired-ends, and all the single reads cannot be used in the workflow.

This phase executes another interesting function: it changes the data format, merging the sequences and quality information in a single file. This allow to speed up the next steps of the pipeline. The data are transformed into a compressed format, fastq [2], that allows to reduce the storage memory. It is a textual file, similar to the multi-FASTA, in which each sequence is introduce by a comment line, followed by the bases and the quality ones.
The module runs in a couple of hours per run, and it did not require too much memory.

Input parameters

Parameter	**Explanation**	**Format**
Paired-end_1 sequence file	the file with the sequences for the first half of the mate-pair	multi-FASTA
Paired-end_1 quality file	the file with the qualities for the first half of the mate-pair	multi-FASTA
Paired-end_2 sequence file	the file with the sequences for the second half of the mate-pair	multi-FASTA
Paired-end_2 quality file	the file with the qualities for the second half of the mate-pair	multi-FASTA

Output

Type	**Explanation**	**Format**
paired_1	the file with the sequences and the qualities for the first half of the mate-pair, that are paired after the sequencing	fastq

[2]`http://maq.sourceforge.net/fastq.shtml`

Parameter	Explanation	Format
single_1	the file with the sequences and qualities for the first half of the mate-pair that are not paired after the sequencing	fastq
paired_2	the file with the sequences and the qualities for the second half of the mate-pair, that are paired after the sequencing	fastq
single_2	the file with the sequences and qualities for the second half of the mate-pair that are not paired after the sequencing	fastq

3.5.4 Trimming

The spectral correction module tries to correct the sequencing error, but sometimes not all the errors could be corrected. For example, if the coverage is too low, it is not easy to discriminate sequencing errors from SNPs, and it is better to maintain the called base. A similar situation happens when it is not clear which is the "best base": if in a genomic position a type of base is not confirmed by more reads than other types, no decision could be taken, and it is safer not to change it, introducing possible new errors.
The sequencing errors are very often related with the low quality of the considered base. In general, the sequencer efficiency decreases with the length of the reads. As the read grows, the quality decreases. Very often, reads are characterized by a good average quality in the middle, and a low quality at the ends. Sometimes, there could be reads with a global low quality, due to some problems occurred during the sequencing.

A read full of unidentified errors, or with a general low quality can wrongly align, increasing the background noise, or not align at all, losing information. If a read has some wrong bases at the ends, it is not possible to align it allowing only few mismatches, and the read information will be lost. But, if the low quality bases at the ends of the read are removed, its quality will improve and it would be easier to align it. This is the basic aim of the trimming module: trying to study an average quality for each reads, filtering all reads that have a global quality lower than a certain fixed threshold, or removing only the bases at the ends characterized by low quality value.

The trimming phase uses two windows to evaluate the global quality of a read, and the quality at its ends. The first window (W) runs over the reads and computes the average quality of all the bases contained in the window. If it is found at least a window, whose average quality is lower than a fixed threshold (T), the read is rejected. Otherwise, a second trimming phase is applied, using a shorter window (w), that works at both ends: from 5' end toward the 3' and vice versa. For each window position, it is computed the average quality, and if this is lower than a fixed threshold (t), the window is moved one step on, and the process continues. The process stops when it is found a window whose average quality is lower than the fixed threshold, or until the read length is less than a fixed threshold l.

It is not easy to determine the correct value of all the parameters.

- The quality threshold could be determined by the sequencer properties. For example, the technical characteristics of SOLiD sequencers determine that under a quality value equal to 9, the bases are not reliable.
- The window length can be statistically computed, evaluating the variations of the percentage of alignments over a randomly computed sample of data using different window values.
- The minimum value of the read length is very difficult to fix. It is known that the longer is the read, the more precise could be the alignment: the probability to fall in repetitive regions or to align in multiple places decreases. The long reads have a high specificity level, but the longer is the read, and the more difficult is to align it if there are errors inside. Anyway, shorter reads are easier to align, even if they can decrease their specificity. Consequently, the idea to trim the ends, deleting the wrong bases and allowing to align a shorter but correct read is very interesting and useful. Nevertheless, it is important to control the amount of background noise that it could be introduced reducing the read length.

The module is integrated in the PASS package, that computes the correct parameters value, as it was previously explained. Before the alignment PASS performs the trimming operations for each read, filtering the read if it has no good quality, or removing the low quality bases at the ends. To perform this operation, PASS uses the file with all the sequences and the file with the associated qualities. For each read, it stores the information about the read fragment that has been saved.

3.5.5 Alignment

The module aligns all the reads on the reference genome (the set of contigs in this case), searching for the best alignment, which has the best score (less number of mismatches). Due to the presence of SNPs or yet not corrected errors, the alignment is performed allowing a certain number of mismatches. Moreover, as discussed in the previous sections, a genome could have many repeated regions, that are quite similar among them. So, there could be reads that fall in these regions and align in more than one position. Even this possibility has to be considered, and multiple alignments are allowed

This module is performed using PASS [27], a tool developed in my laboratory, that is able to align a large amount of short reads, (generated with Solexa, SOLiD or 454 technologies). PASS is able to work in a multi-threading operational mode, parallelizing the process and reducing the computational time. Unfortunately, it loads into RAM memory all the genome of the studied organism, and for large genome requires a lot of memory. It is proportional to the length of the used organism.

Input parameters

Type	Explanation	Format
number of CPU	the number of available CPU, to determine the parallelization level (number of thread that could be used). This value depends on the used system	integer value
reference genome file	the genome sequence to use as reference, on which reads will be mapped	multi-FASTA
reads to align	the file of the reads to align (sequences and qualities)	multi-FASTA or fastq
identity percentage	the minimum percentage of identity that a read alignment should have to be considered a good alignment	multi-FASTA
gaps	the maximum number of allowed gaps	integer
b	only the best-hit alignment are used	character ‘b’

Output

In output, PASS stores in a GFF format file all the useful information about the alignment: the name of the contig where the read aligns; the start, end position and strand of the alignment; the number of bases of the alignment; some useful information about the alignment, like the name of the read, the trimming and quality information.

Type	Explanation	Format
alignment file	the file contains all the aligned reads	GFF format
aligned reads	the file contains all the reads aligned	fastq
log file	the file contains a log of all operation performed during the analysis, and some statistics about the alignment	txt
not-aligned reads	the file contains all the reads that do not align	fastq

3.5.6 Pairing

The alignment module finds the best alignment for each read, if this exist, but it does not consider the original paired-end nature of the reads. Each end is separately aligned, and a pairing step should be performed, to reconstruct the pairs. This module realizes this function: it pairs and classifies the paired-ends using the order, orientation and distance information. At the end, it computes the library length distribution. This module has been realized using the tool PASS_pair of the PASS package.

Paired-end pairing and classification

The genome is composed by a set of disjoint sequences, contigs, and the way by which the paired-ends align and pair on them, produces three classes of paired-ends.

- "pair-IN": the paired-ends that have both ends mapped on the same contig;
- "pair-OUT": the ends matches on different contigs;
- "single": only one end of the mate-pair maps.

The main difference between the first two classes is due to the applicability of the biological constraint to the first class, that cannot be used for the second one. For a paired-end of the pair-IN class the distance, the order and the orientation can be computed and controlled; while, for the other class, these values are undefined, and all related considerations cannot be applied here. Moreover, it is not easy to filter the background noise present in this class, and to evaluate this kind of paired-ends. The pair-IN class is used to compute the library length distribution, while the pair-OUT is used to create connections between contigs, that would be used in the scaffolding process.

Another possible paired-end classification considers an alignment characteristic: a read could map with an uniquely best alignment or could not align on the genome. The intersection of the two classifications returns the first paired-end division:

- unique pair in: the paired-ends that uniquely align on the genome, inside the same contig;
- not unique pair in: the paired-ends that not uniquely align on the genome, inside the same contig;
- unique pair out: the paired-ends that uniquely align on the genome, and that are divided between two contigs;
- not unique pair out: the paired-ends that not uniquely align on the genome, and that are divided between two contigs;
- unique single: the single reads that uniquely align on the genome;
- not unique single: the single reads that does not uniquely align on the genome.

These classes can be further divided. For the paired-ends of the pair-IN class, the distance and the orientation can be studied, and this allow to determine if the paired-end is correct or not. Using this result, paired-ends could be divided into correct or wrong paired-ends for distance or orientation.
The not unique pair out could be divided into two subclasses: the class where only one of the two reads of the mate does not align uniquely (one not unique pair out) and that one where both reads do not align uniquely (both not unique pair out). The Pairing module stores all paired-ends in the correct class, as explained in more detail in the output section (see Section 3.5.6).

Library length distribution

In the scaffolding phase, the insert length of the library is a very important information to use for the evaluation of contig distances, as it will be seen in the *Make arcs* Section 3.5.9. Consequently, it is very important to use a correct value. During the experimental preparation of a paired-end library, it is known which is the size range of the inserts. This gives an idea about the insert length, but a lot of experimental variables can influence it, misleading the evaluation. To overcome this problem, it is possible to find a better value analyzing the pair-IN. In fact, for all the paired-ends that fall in this class it is possible to compute the real distance, and this allow a better evaluation of the insert length value.

The plotting of all paired-end distances gives the library distribution. In an ideal situation, the library preparation should produce a Gaussian curve, with the an value (μ) and the related standard deviation (σ). The curve is composed by a central region where falls the great majority of the data, that give the insert length value to use, and two smaller regions at right and left, where fall the errors (chimeric insert) or less probable values (see Figure 3.8). Anyway, often the distribution has different shapes, due to many possible experimental and computational factors.

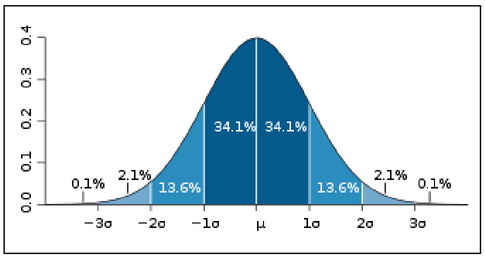


Figure 3.8: *The normal (or Gaussian) distribution $N(\mu, \sigma)$, where μ is the average value, and σ is the standard deviation. As the figure above illustrates, 68% of the values lie within 1 standard deviation of the mean; 95% lie within 2 standard deviations; and 99.7% lie within 3 standard deviations.*

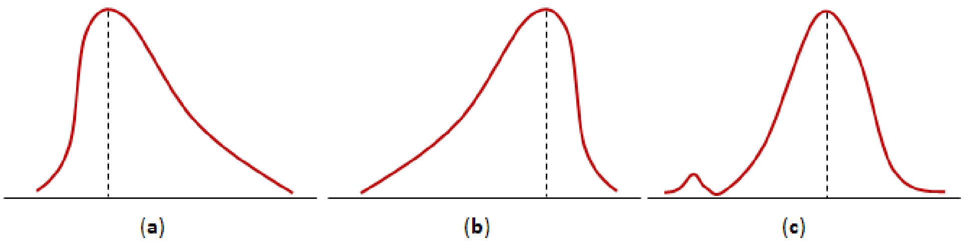


Figure 3.9: *Three possible asymmetric shapes of the curve:* (**a**) *the curve is unbalanced to left, due to a lack of small paired-ends;* (**b**) *and* (**c**) *large quantities of short paired-ends remain in the library during the sequencing.*

First of all, the shape of the curve strictly depends on the problems occurred during the experimental steps of the library preparation. For example, it is possible that paired-ends of small length remain in the cut gel-slice: this will unbalance the curve on the left side or produce another little Gaussian at the left of the main one (see points (b) and (c) of Figure 3.9). On the other hand, it could happen that many long paired-ends enter during library preparation, and only a small quantity of short one, with the creation of an asymmetric curve (see point (a) of Figure 3.9). In both cases, the Gaussian has an asymmetric shape, and the average length and standard deviation

are misled.
Even the computational estimation of the the library length could be subjected to some problems. In this phase, it is applied the idea that paired-ends map and pair on the same contigs, implicitly assuming that contigs are large enough to contain the paired-ends. The assumption is not always satisfied: the used contigs could be very short with respect to the the library length, because of the low sequence coverage of the reads used for the assembly. The result is that only the smaller paired-ends are able to map and pair on the contigs, and the computed curve does not represent the complete distribution, but only a part.
The considerations and problems should be take into account during the insert length evaluation. When there is an asymmetric Gaussian, the average length value can be misled, and influenced by the largest part of the curve. This can be detected, comparing the mean value with the median: the more is their distance and the more is the curve asymmetry.
When the library is greater than the average length of the contig, the insert evaluation should be corrected, computing the statistic for the only biggest contigs. In this class will be entered all the greatest paired-ends, and the smallest ones do not influence the mean value computation.

First of all the module plots all paired-ends of the pair-IN class, to determine the shape and characteristics of the distribution. It computes an initial mean value μ, and the standard deviation σ. Then, it detects if the computed insert length evaluation is correct, or it tries to correct it. It fixes the μ and σ value, and it computes the interval $D = [d_1, d_2]$ on which paired-end could be considered good pairs. Finally, it performs the paired-end classification:

- it determines if the paired-end is an unique or not paired-end;
- it determines if the paired-end is a pair-IN , pair-OUT or single;
- For pair-IN , it controls distance and orientation:
  - wrong orientation $\rightarrow$ unique wrong strand or not unique wrong strand;
  - wrong distance, that means that paired-end length is out of the interval $D \rightarrow$ unique wrong distance or not unique wrong distance;
- For pair-OUT it divides all the mates in unique, one not unique and both not unique;
- single class is divided into unique and not unique.

Input

The required data are the output of the alignment program.

Type	**Explanation**	**Format**
R3 read file	the file with R3 read of the paired-end aligned	GFF
R3 read file	the file with F3 read of the paired-end aligned	GFF
reference genome file	the genome sequence to use as reference, on which reads have been mapped	multi-FASTA
library type	this set of parameters defines if the data are paired-ends or mate-pairs, and the sequencer that have produced it	string

Output

Type	**Explanation**	**Format**
UNIQUE PAIR	the set of all paired-end reads that uniquely align on the genome, in the same contig, with a correct order, orientation, and which distance respects the library insert length.	GFF
UNIQUE WRONG D	the set of all paired-end reads that uniquely align on the genome, in the same contig, with a correct order, orientation, but with a wrong distance value respect the to library insert length.	GFF
UNIQUE WRONG S	the set of all paired-end reads that uniquely align on the genome, in the same contig, with a wrong orientation	GFF
NOT UNIQUE PAIR	the set of all paired-end reads that perform multiple alignment on the genome, in the same contig, with a correct order, orientation, and which distance respects the library insert length.	GFF

Type	Explanation	Format
NOT UNIQUE WRONG D	the set of all paired-end reads that perform multiple alignment align on the genome, in the same contig, with a correct order, orientation, but with a wrong distance value respect the to library insert length.	GFF
NOT UNIQUE WRONG S	the set of all paired-end reads that perform multiple alignment align on the genome, in the same contig, with a wrong orientation	GFF
DISCARDED PAIR	all reads that map on the same contig, by that do not enter in any of the previous classes	GFF
UNIQUE SINGLE	all half of the pairs, that uniquely align on the genome, and that does not have the other half aligned	GFF
NOT UNIQUE SINGLE	all half of the pairs, that perform multiple alignments and that does not have the other half aligned	GFF
UNIQUE PAIR OUT	the set of all the pairs that uniquely align on the genome, in two disjoint contigs	GFF
ONE NOT UNIQUE PAIR OUT	the set of all the pairs that have one half that uniquely align on the genome, in two disjoint contigs, and the other half performs multiple alignments.	GFF
BOTH NOT UNIQUE PAIR OUT	the set of all the pairs that have one half that uniquely align on the genome, in two disjoint contigs, and both halves have multiple alignments	GFF
log file	in the file there are all the performed operations, and some statics about pairing (as explained before)	txt

3.5.7 Redundancy

Sometimes, there could be some regions that are more sequenced than the others, that causes the production of identical mate-pairs, and the consequent presence of regions more covered than others. The redundancy introduces a wrong coverage, and could mislead the further analysis that are based on the number of reads that confirm some evidences. It is useful to normalize the starting set of data, and to consider only a mate-pair for each group of equal pairs.

This problem can be solved at two different level of the pipeline: after the spectral correction, or after the pairing phase. The first application is the best one, because it consider the original reads, after the error corrections, and only the equal ones are rejected. Anyway, in case of reads with low quality at the ends, it can happen that the bases would be wrong, and this does not allow to correctly delete the redundancy step. So, it seems to be reasonable to apply this module after the pairing, and to delete all the mate-pairs which alignment is equal.

Input parameters

Type	Explanation	Format
unique pair	the file with the unique pair data	GFF
unique pair out	the file with the unique pair out data	GFF

Output

Type	Explanation	Format
unique pair no red	the unique pair file without redundancy	GFF
unique pair out no red	the unique pair out file without redundancy	GFF
log file	in the file there are all the performed operations, and the statics about the percentage of filtered redundancy	txt

3.5.8 Consistency analysis

Sometimes the assemblers can wrongly merge together stretches of contiguous bases that do not correspond to the original sequence. This effect, called misassembly, can be caused by different reasons: the presence of many repeated and low linguistic complexity regions, the large quantity of inserted

sequencing errors, or the uncorrect decisions taken during the assembly process. For example, the Roche/454 is prone to insertions and deletions, because it is not able to correctly determine the length of long homopolymeric region, stretches of identical bases (see Section 1.4.1). On the other hand, the Illumina/Solexa and SOLiD/Applied Biosystem are subjected to substitutions (see Section 1.4.2 and Section 1.4.3). Furthermore, for example the Newbler assembler is often not able to determine the correct sequence of repetitive regions, or it collapses them into a unique consensus. Therefore, the produced set of contigs should be checked before the scaffolding phase, to avoid all the ambiguous and inconsistent situations and to create a final uncorrect assembly.
The purpose of the consistency module is the analysis of each contig to discover if there are some evidences about the presence of possible misassembled regions. At the end, it will be filled a list with all the found information, that could be used in the scaffolding phase, to solve the ambiguous situation and find the most probable solution, if it exists.

Misassembly classification

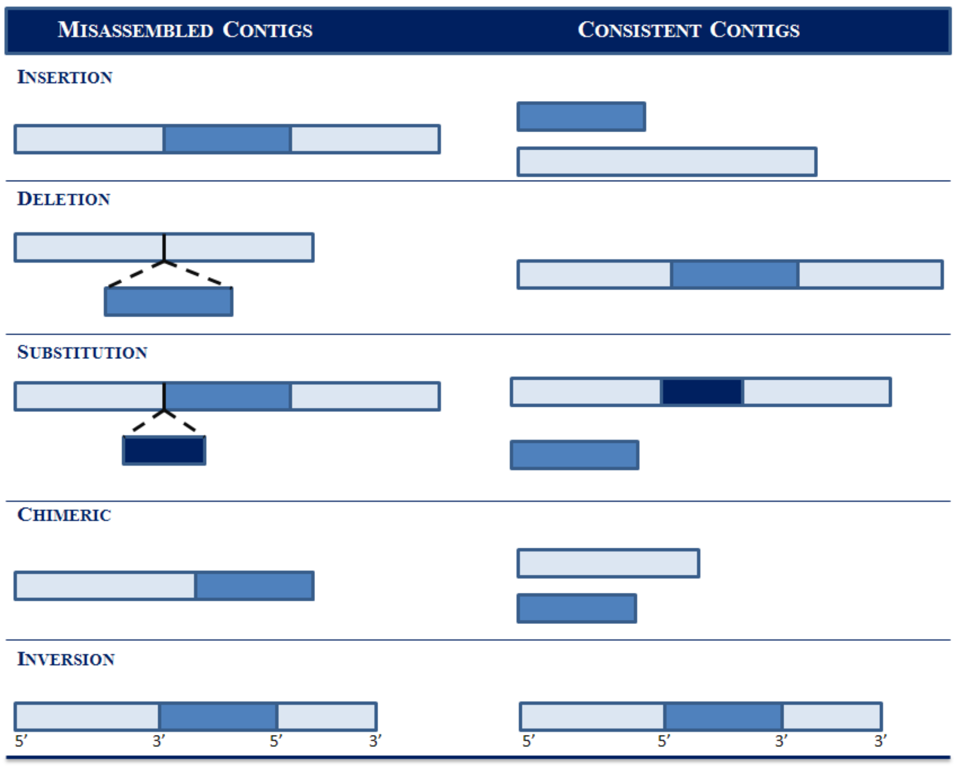


Figure 3.10: *In the figure, the misassemblies classes are shown: insertion, deletion, substitution, chimeric and inversion.*

There can be different types and causes of misassembly, organized into the following cases (see Figure 3.10):

- **Insertion**: a wrong segment is contained inside a contig, probably due to the very repeated or low complexity nature of the points of insertion, which can mislead the assembly. To correct the problem, the wrong part has to be sliced off, and the left ones are to be joined, obtaining two new sequences.
- **Deletion**: a stretch of sequence is not assembled in a contig, and the insertion of the lack piece in the proper position will make it consistent. This is a difficult task, because the missed part could not have been sequenced, consequently it is not present among the available data, or it could not have been correctly assembled. Eventually the absent sequence could be reconstructed, but this is an hard operation. All these considerations explain the greater degree of complication presents in this problem with respect to the previous one.
- **Substitution**: contig contains a wrong part of the sequence, that can be substitute with the right one. This is the fusion of the insertion and deletion cases. The correct piece has been swapped with a similar but wrong one, due to the lack of the correct one, or other assembly problems. The solution requires to take away the mistaken segment, and to insert the missed one in the right position.
- **Chimeric regions**: the contig is composed by two disjoint portions, that have been erroneously merged together, due to the similarity of their boundaries. The identification of the cut at the breakpoint, where the two not adjacent genomic regions have been erroneously joined would solved the problem, and two separated consistent sequences will be produced.
- **Inversion**: the contig is composed by the right pieces, but the orientation of one of them is wrong. To solve the problem, the region should be flipped inside the contig.

Evidences

Looking at the genome sequence alone, the only analysis that could be performed is about its degree of complexity, but this does not detect potential misassemblies. Accordingly, other *ad hoc* evidences should be computed for this task. Which kind of information could be used in this regard? And, is it possible to extract it from the yet existing data, without requiring other sequencing or experimental analysis?
The available assembly is provided with the measure about how many reads

have been used to create each contig and each base. This is very useful for discovering the repetitive nature of the genome, but alone it is not interesting for the misassembly itself. On the contrary, the paired-end libraries seem to well model this problem. In fact, their mapping and pairing on the genome are interrupted in the inconsistent regions, underlining some potential structural errors. If two segments have been wrongly merged together by the assembler, there would be no mate-pairs that connect them; differently, there will be other links that show the bonds with the correct pieces positioned away on the genome.
This discussion demonstrates that all the needed information could be extracted from the available data, answering the above questions. Taking into account of all the evidences coming from the paired-end libraries and the genome sequence, the following tracks could be obtained:

- **(local) physical coverage**: it determines, for each base of a contig, the number of times it is spanned by mate-pair reads. In case of conistency, this value would be quite homogeneous over all the genome, except at most for the ends of the contigs, where obviously there is a lack of data. All the other significant signal variations could hide problematic situations, and are to be further investigated. As a matter of fact, whereas a misassembly occurs, the physical coverage can fall even to zero, and the amount of fluctuation strictly depends from library length and type of problem.
  This trace is computed using the mate-pairs contained in the unique pair class, because it is less probable that the single alignments could be wrong, and only the bond inside the same contig could give a safe pairing. If there is more than one library, they are separately elaborated, because in this way it is easier to make more specific considerations about the critical regions. Each track defines for each base of each contig its physical coverage value.

- **indel information**: it evaluates the library distribution shift from the expected one, in presence of insertions and deletions.

  In the first case the increase of the average insert length will be related to the the extension of the inserted sequence, and the curve moves on the right; in the second case the average will decrease with the same law, shifting to the left. When the suspected regions are deleted or added, the library will have once again its usual shape. In theory, it is always feasible to detect an insertion of any length, because the moving on the right is possible up to infinity. However, this is not an easy task, and greater is the insertion and more difficult is the analysis. Whereas, there is a lower bound for discovering the deletion that is imposed by the library length, since the distribution cannot move under zero.
  This track is also based on another interesting observation: in presence

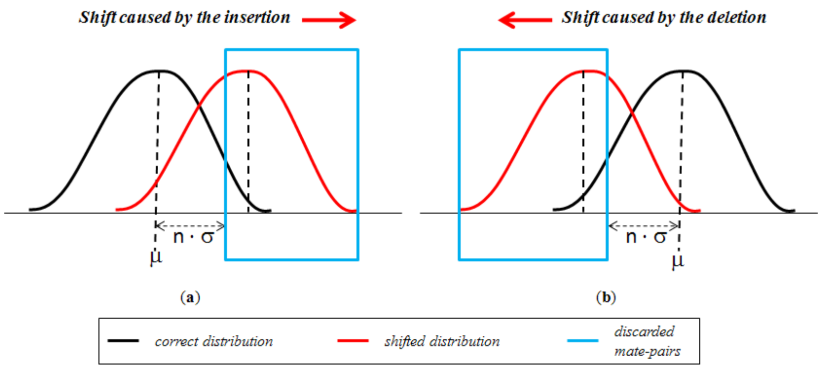


Figure 3.11: *(a) insertion case: the curve moves on right; (b) deletion case: the curve moves to left.*

of assembly errors variations, the reads that map the point of misassembly are spliced into two non adjacent regions. This is a difficult analysis to execute, due to the hitch of the splicing alignment and its low reliability. In fact, for short reads, it is not easy to find the correct splicing alignment. However, if this information is confirmed by many reads, it achieves a higher level of confidence, and it could be used in the problem analysis.
This track is computed using two kind of mate pairs: the unique pair, and the unique pair wrong distance. If the variation is too small with respect of the library length, the mate pairs fall in the first class, and they detect small insertions and deletions; while, for greater indels, the data are classified as wrong distance links, and they discover large insertions and deletions. For the middle case, in which the variation of size is around the library length, the information would be taken from both the classes. The output information contains all the potential misassemblies found.

- **outer pairing**: it computes all the possible connections among contigs, using the unique pair out class. As said before, the uniquely aligned mate-pairs are more probably error free. In an ideal situation, the order, orientation and distance calcolated for a mate-pair should be consistent with all the other ones. In presence of misassemblies, there will be anomalous links inside the contigs, that sometimes could not respect the above constraints, highlighting critical situations.
All the mate-pairs are grouped into different clusters defined by: the couple of contigs they link, the reciprocal order and orientation. The track is composed by a set of arrows, one for each cluster, mapped on the contigs, with the following associated properties:

  - the pointing direction: 5' or 3', according to the mate-pairs;

  - the associated weight: defined by the class cardinality;
  - their size: it is equal to the length of the region covered by all the links of the cluster.

- **low complexity index**: it studies the decreasing of linguistic complexity of the genome. This measure determines the degree of sequence variability: the complexity decreases in presence of long stretches of repetitive short patterns.
  The signal is determined from the genome sequence itself. Each position has an associated value, computed counting all different patterns, of all sizes, contained in a window $W = w_1...w_l$ of a fixed length l, centered on the evaluated base. The used formula is:

$$complexity = \sum_{i=1}^{l} |\{s \subset W : s = s_1...s_i : s_j \in \{A, C, G, T\}\}|$$

- **repetitiveness index**: it is a very interesting and useful measure, because it shows the level of repetitiveness present in the genome.
  In the repeated sequences, quite often it is not possible to align the reads uniquely, with the consequent lack of mate-pairs information. Without the calculation of this index, it is not easy to understand if the absence of data is due to assembly or to alignment problem. For example, if the reads cannot be mapped, there will be a decrease of the physical coverage due to a repetitive region rather than to a misassembly.
  The track defines for each base the repetitive level on the genome of a window centered on it [70]. The higher is the value and the higher is its repetitiveness.

  At first sight, this measure could be confused with the previous one, and they could seem complementary. If the complexity signal decreases, the repetitiveness index increases. However, they are quite different and independent, and this explain the necessity of computing and using both of them. For example, there could be a very low complex pattern that is placed only once in the genome, and so it has both a low complexity value and even a low repetitive index.

Algorithm

In the previous section all the available evidences used to perform the consistency analysis has introduced. Each of them covers specific aspects of the problem, and their integration into a common program will be a useful tool for misassembly detection. Nevertheless, even if a lot of information confirm the same hypothesis, the resolution remains an hazardous issue for

many reasons: often it is not sure to have discovered a real inconsistency; it is possible that the supposed solution could be not right; it is very difficult to determine how to correct a critical region, without introducing others unwanted artifacts. This explains why this module does not directly repair the contigs, but it just saves all the found information into a list of evidences. The correction phase will be performed during the scaffolding, where there will be a global vision of all contigs and their relationships, that could help the adjustments.
The developed algorithm parses the input tracks, and merges together the shown information to discover all possible hidden misassemblies. It tries to characterize each potential critical region, defining: its type, its size, and all the considerations that have highlighed and confirmed it. It is organized into a set of steps and controls, that are synthesized in the below framework. For each contig, it executes the following operations:

1. it checks if there is at least an evidence about a potential inconsistency, otherwise the contig is labeled as correct, and the workflow is terminated. The signals could be a decrease of the physical coverage, a shif of the library distribution from the expected value, an anomalous connections between contigs, or some information about particular repeated patterns.

2. if some characteristic factors have been detected, it continues the analysis trying to determine if it is a misassembly or not. In the second case, it stores all the suspicious evidence, that could be useful in the scaffolding process.

3. If it is an inconsistent situation, it starts the problem analysis:

    (a) it determines the kind of misassembly: insertion (small, medium or large), deletion (small, medium or large), substitution, chimeric region or inversion (small, medium or large).

    (b) it studies how to solve the problem: each kind of situation is characterized by different evidences, and for each of them it has been designed a special set of rules and controls to apply. These will be explained below.

    (c) finally, it stores all the extracted information in the output list.

A frequent event is the absence of some information due to the lack of data and not to the presence of errors. The problem becomes more complicated, and it could be more difficult to formulate a correct hypothesis. The following discussion is focused on the analysis of each possible misassembly, with all the evidences that could show it, and what can happen in absence of some of them. It is supposed to work with any one mate-pair library: the discussion could be easily generalized to a variable number of libraries.

Large insertion

As it has already said before, this problem can be further divided into large (or medium) insertion, if its length is greater or equal to $\mu + n\sigma$, or small one if it is lower than the fixed value. The two cases will be studied separately, due to the different kind of evidences that detect them. In this section, it will be discussed the first one, represented in Figure 3.12.

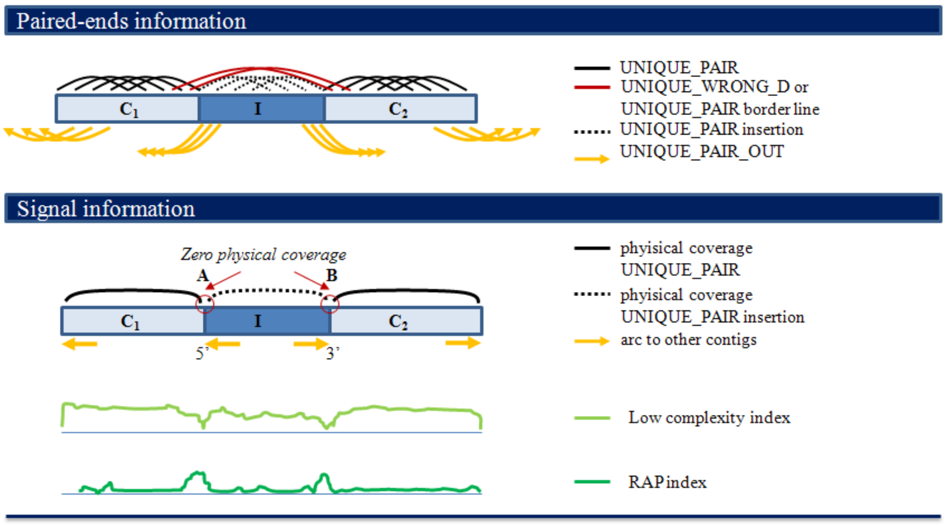


Figure 3.12: *The figure shows an example of large insertion, with all the information brought by the tracks in such situation. The contig is composed by the two correct parts C_1 and C_2, connected to the insertion I, around the points A and B, where the physical coverage falls to zero. If the insertion is around $\mu + n \cdot \sigma$, the mate-pairs can fall in the unique wrong distance class, or they can be borderline links of the unique pair one.*

- *Physical coverage.*
  In an ideal situation, the presence of a large insertion in the middle of the contig is marked by two points (A and B) where the physical coverage falls to zero, at 5' and 3' end of the central region (the circled ones in the Figure 3.12). This is due to the mate-pairs that cannot match between misassembled stretches of bases.
  There are several variants of the previous one, due to the nature of the sequence, and to the length of the three separated regions contained in the contig (indicated as C_1, I, C_2 in the figure). This would determine the presence or not of physical coverage.

  - if one or both of the two external regions C_1 and C_2 are too small ($\leq \mu + n\sigma$) for the used library ($N = (\mu, \sigma)$), they could not have internal signal. This situation has zero physical coverage, and

the two points A end B could exist only if the central region has physical coverage.
The indel signal could balance this lack of information, as it will shortly show.

  - The fact that the central region is large enough to contain mate-pairs would assure a certain value of physical coverage. In fact, if this does not happen, it could be due or to the borderline length of I, that could be verified comparing it with the library insert length, or to the low complexity or repeated nature of the sequence, that will be confirmed by the specific signals.

- *indel information.*
  In general, the nature of the large insertion classifies the mate-pairs, that connect the two external regions of the contig C_1 and C_2, as unique pair wrong distance. The shift of the library distribution to the left is equal to the length of the region I. If the original contig length (C_1 concatenated to C_2, without I) is too small, it happens that do not exist mate-pairs that connect the two sequences, and even this signal gives no information.
  All the mate-pairs that align in the critical points A and B would be spliced into two pieces, confirming that the internal region is worng. If this evidence is supported by enough spliced reads, they will help the identification of the breakpoints with quite high precision. Therefore, cutting of the I region, and pasting the C_1 and C_2 sequences generate the right contig.

- *outer pairing.*
  In the real genome, the inserted regions is linked to other pieces external to the contig. This causes the possible presence of some mate-pairs that link the middle segment with the correct ones. This information is very useful both to confirm that there is a misassembly, and to understand which is the right position of this sequence in the genome.

- *low complexity and repetitive indexes.*
  A misassembly is very often caused by repetitive or low complexity regions that have misled the assembler. The consequence is that where there is an inconsistent contig, the wrong segments are probably identified by a decrease of the first index, and an increase of the second one. This does not mean that each time there is a similar behavior, there would be a problematic region. These signals are very useful to confirm the formulated hypotheses, and they are always available.

Small insertion

This second case of the insertion problem is more difficult to discover than the previous one. In fact, in this situation the intrusive region I is too small to be detected with the usual evidences, and at first sight the contig looks very similar to a consistent one, as shown in Figure 3.13.

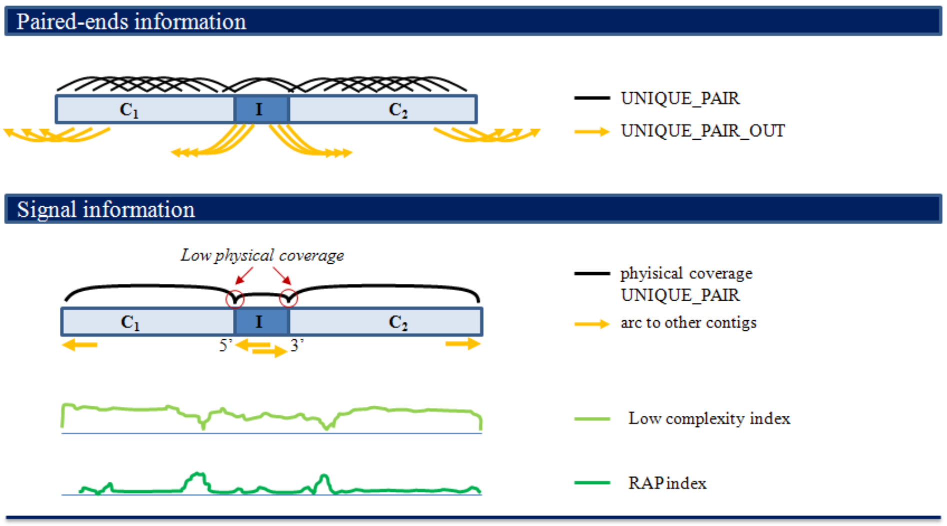


Figure 3.13: *The figure shows an example of small insertion, with the evidence of all the tracks. The contig is composed by the two correct parts C_1 and C_2, connected to a small insertion I, around the points A and B, where there is a decrease of the physical coverage.*

- *Physical coverage.*
  In a standard situation, some mate-pairs will be able to jump the middle region due to its small size, and the physical coverage does not fall to zero. Consequently, there is not the usual evidence that underlines the presence of a possible misassembly. Anyway, the tiny dimension of the central sequence does not allow the pairing of mate-pairs inside it, and the region does not contribute to the physical coverage value. Moreover, very often the fragments of C_1 and C_2, adjacent to I, are quite repeated, and the mate-pairs hardly map on them, inducing a further decreasing of the physical coverage.
  Sometimes, it could happen that the two part C_1 and C_2 would be very small, and mate-pairs cannot map on them. This causes a lack of physical coverage in this region and it complicates the analysis of significant signal variation in the misassembled region.
  The discussion highlights the difficulties to detect this kind of problem using this signal: the lower physical coverage in the interested region could be confused with its normal fluctuation. The detected variations

could be not due to the presence of a critical region, but to the lack of information caused by sequencing or alignment problems, that does not allow to detect the misassembly presence.

- *indel information.*
  This signal is more useful than the previous one, because it is given by the local variation of the library length. In this case, the mate-pairs are not classified on average as unique pair wrong distance, because their size does not get out from the fixed range ($\mu + n\sigma$). Anyway, there could be some arcs that are placed in the right part of the distribution, at the limit of the accepted interval, and the added insertion has moved them in the wrong distance class. But, this should be isolated cases, that could be not easily distinguish from background noise. In general, all the mate-pairs should fall in the unique pair class.
  Finally, the splicing phenomenon around the C_1 and C_2 regions confirms the evidence of a possible misassembly and it helps the identification of the breakpoints.

- *outer pairing.*
  This track provides the same information given for the large insertion problem explained before. The middle region could be connected to other contigs through mate-pairs, but this happens with a lower probability, due to the small size of the insertion. Besides, if this fragment is a consensus of repeated sequences, the reads would hardly map there. In conclusion, very often this track is not available in this situation.

- *low complexity and repetitive indexes.*
  These two indexes give the information about the nature of the sequence. With high probability, the small region has a certain level of repetitiveness, that explains the alignment difficulties and the lack of the other tracks.

Large deletion

Even in this case, the problem has to be divided into two ones, depending on the size of the deleted region. If its length is greater than $\mu + n\sigma$, it is classified as large deletion problem, otherwise it is classified as small one. The two situations have different characteristics and evidences, and they are studied separately. In the Figure 3.14, there is a schematic example of a large delition case with all the tracks.

- *Physical coverage.*
  The physical coverage behavior is very similar to the one found in the

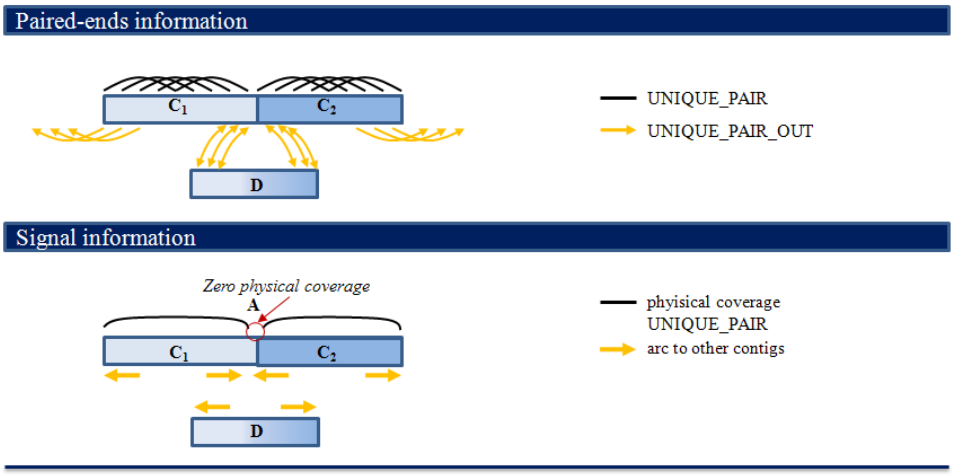


Figure 3.14: *The figure shows an example of large deletion, with the evidence of all the tracks. The contig is composed by the two correct parts C_1 and C_2, that are wrongly connected in the A point, where there is drop to zero of the physical coverage. The correct composition requires the insertion of the D region between the previous ones.*

large insertion problem. Even in this case, it usually falls to zero around the misassembly, but there is only one critical point A rather than two as in the other case. The signal decrease is justified by the deletion dimension: it is too large to be jumped by the mate-pairs. This also explains why the mate-pairs do not usually fall in the wrong distance class, and if there are some of them, they would be isolated cases, and it would be very difficult to distinguish them from the background noise. If one or both of the C_1 and C_2 regions are smaller than the library length, there would be zero physical coverage, with the consequent lack of signal, and no evidence about the critical region.

- *indel information.*
  In presence of a large deletion, the unique pair class does not change its mean value. The only visible effect is due to the reads that are spliced in the misassembled point A.

- *outer pairing.*
  If available, this track identifies which is the deleted region. In fact, the mate-pair will connect both C_1 and C_2 with the same piece D (as seen in figure), and this is shown by a couple of arrows that map in the middle of the contig.
  This evidence is not always available due to two possible reasons: the deleted region has not be assembled, and it is absent from the released genome, or it exist, but it has a high level of repetitiveness detectable in the next tracks. In both cases, the mate-pairs cannot match there

and the visible phenomenon is a set of unpaired reads that map around the misassembled point, classified as unique single.

- *low complexity and repetitive indexes.*
  The behavior is the same explained for the previous cases.

Small deletion

This is the second case of the deletion problem. In this situation, few bases have been deleted from the contig, and could be separately assembled (see Figure 3.15).

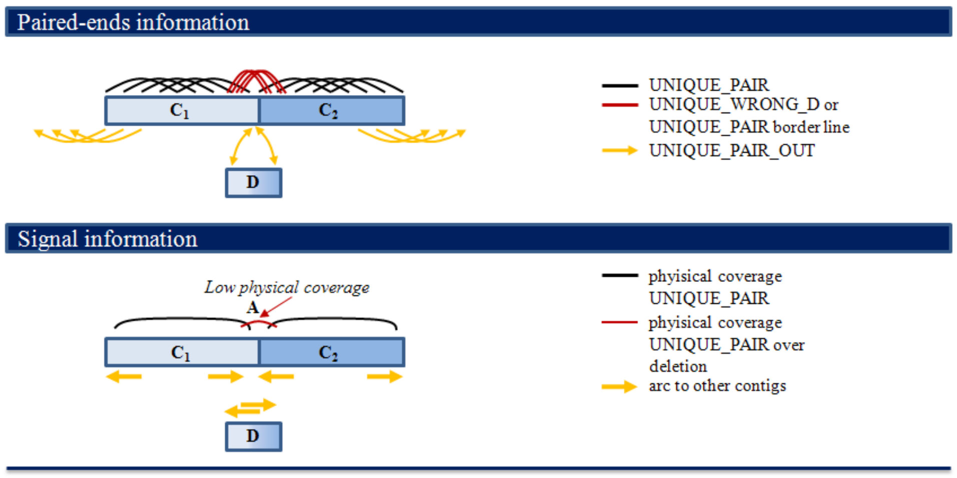


Figure 3.15: *The figure shows an example of small deletion, with the evidence of all the tracks. The contig is composed by the two correct parts C_1 and C_2, that are wrongly connected in the A point, where there is a low fluctuation of the physical coverage. The correct composition requires the insertion of the D region between the previous ones.*

- *Physical coverage.*
  The signal has a shape very similar to that assumed in the small insertion problem. Even in this case, there is a low decrease of the physical coverage, that could be confused with the normal fluctuation of the signal. This is due to the ability of the mate-pairs to jump a small deleted region and map inside the contig. The links that connect the D region with C_1 and C_2, are not present in the signal, causing its drop around the critical point A. These fluctuations are not so significant to be safely discriminated against usual background noise or lack of data.

- *indel information.*
  In this case the mate-pairs that are able to jump the deleted region

would have a lower length, than the correct one. If the change is very small, the mate-pairs are still classified as unique pair, while if it is quite large, they fall in the unique wrong distance class. In the first case, it would be difficult to determine the distribution variation, while in the second one, it would be easier. In both situations the alignment of the spliced reads could confirm the presence of a misassembled region.

- *outer pairing.*
  The behavior for this track is equal to the previous one studied in the large deletion problem. Even in this case, the mate-pairs could show which is the deleted region to add the contig. But, the probability to have this kind of information is very low, and sometimes there is only the evidence of a set of single reads mapped around the breakpoint A.

- *low complexity and repetitive indexes.*
  The behavior is the same explained for the previous cases.

Chimeric region

Sometimes the assembler links together too disjoint regions, causing a chimeric situation, as seen in Figure 3.16.

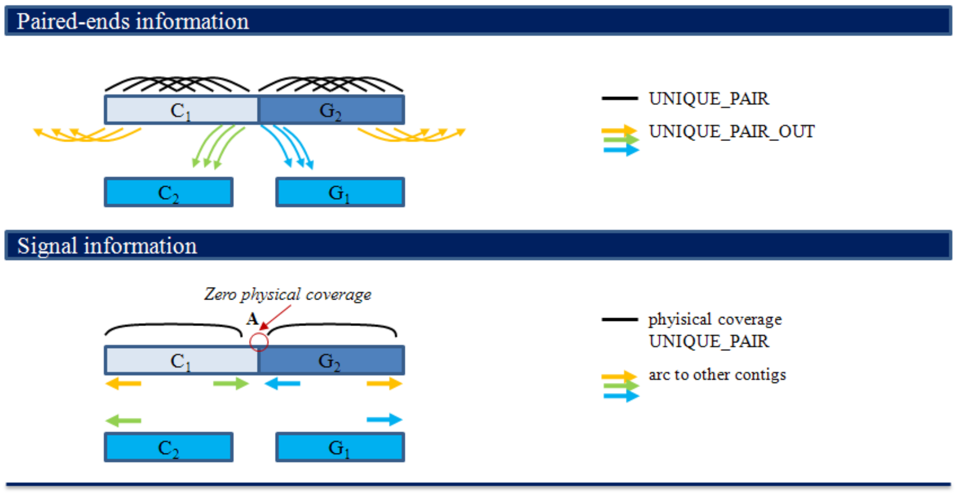


Figure 3.16: *A schematic representation of a chimeric situation, in which two disjoint contigs have been merged together, in A.*

- *Physical coverage.*
  The union of two disjoint regions causes a decrease to zero of the physical coverage in a unique breakpoint A, like for the deletion problem. The different nature of the two fragments does not allow the

mapping of mate-pairs between them, with the consequent expected lack of information both in the unique pair and wrong distance classes. Sometimes, it could happen that one or both of the two pieces are too small to allow the mapping of the mate-pair inside them. This causes a general low or zero physical coverage in all the contigs, and it is not possible to detect the chimeric region.

- *indel information.*
  In this case, the type of misassembly does not provide a signal. There is no shift of the library distribution, or mate-pairs that do not correctly pair. This justifies why for this class the track does not give any contribution in the detection of the problem.

- *outer pairing.*
  If it is available, this is very useful for the determination of the chimeric region, and to discover which are the correct connections. Like for the deletion case, even in this situation there will be arrows that link it with other contigs. Anyway, while in the the previous case the arrows connect the same regions, the misassembled and deleted ones, here they link different contigs.

- *low complexity and repetitive indexes.*
  The behavior is the same explained for the previous cases.

Large inversion

This situation is very similar to the large insertion one. Even in this case, an internal misassembled regions influence the correct mapping of the paired-ends. The event can be divided into: large (or medium) inversion, if its length is greater or equal to $\mu + n\sigma$, or small one if it is lower than the fixed value. The two cases will be studied separately, due to the different kind of evidences that detect them. In this section, it will be discussed the first one, represented in Figure 3.17.

- *Physical coverage.*
  The discussion is very similar to the large insertion one. In a general configuration, the physical coverage falls to zero in two points A and B, that correspond to the two points in which the internal fragment has been wrongly oriented. In this case, the mate-pairs are able to span around A and B, but in wrong strands. Consequently, the unique wrong strand class will be filled, and this will help the discrimination between large insertion and inversion case.
  If the insertion is quite enough large as hypothesized, its internal physical coverage could arise, as for the large insertion case. Finally, if

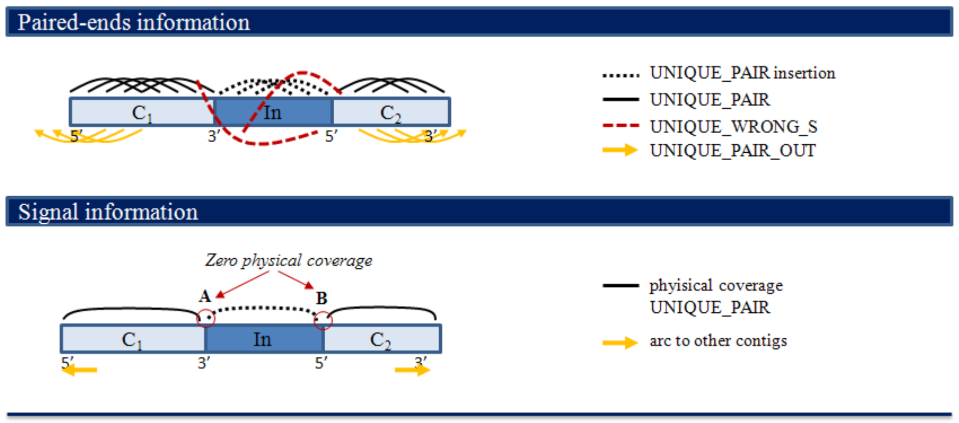


Figure 3.17: *A schematic representation of a large inversion situation, in which a piece of the contig has been uncorrectly oriented.*

the two external regions C_1 and C_2 are shorter than the library insert length, they could not present physical coverage.

- *indel information.*
  In this situation, the wrong data are classified in the unique wrong strand class. This is another evidence that highlights the inversion origin of the misassembly rather than the insertion one.
  Besides, the reads that fall in the breakpoints will perform a splicing alignment, dividing between them.

- *outer pairing.*
  Due to the fact that the inversion region belongs to the contig, there will not be connections that suggest anomalous situations. In the large insertion situation, the outer pairing will help to discover the inserted region, and to correctly place it.

- *low complexity and repetitive indexes.*
  The behavior is the same explained for the previous cases.

Small inversion

This is the second case of inversion, and as the previous one, it is very similar to the corresponding small insertion, as the Figure 3.18 shows.

- *Physical coverage.*
  The mate-pairs could be able to jump the inversion region, and consequently it is not easy to see a zero physical coverage. The only visible phenomenon is the fluctuation of the curve in the breakpoints A and B and probably the signal level decreases. If the contig is large enough

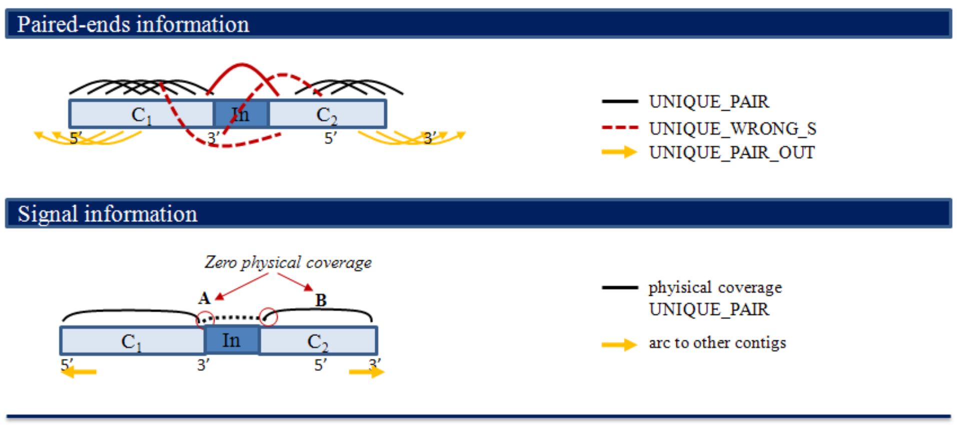


Figure 3.18: *A schematic representation of a small inversion situation, in which a piece of the contig has been uncorrectly oriented.*

with respect to the library insert length, the unique wrong strand class will contain all the mate-pairs that wrongly aligned on it.

- *indel information.*
  The discussion is similar to the previous one. The mate-pairs that fall around the breakpoint regions, will be classified as unique pair wrong strand.

- *outer pairing.*
  Even in this case, the signal is not present, because the inversion region belongs to the contig, and it has only a wrong orientation.

- *low complexity and repetitive indexes.*
  The behavior is the same explained for the previous cases.

Other ambiguous cases

There could be some situations that are very ambiguous, presenting all the misassembly characteristics, but they are only problematic regions, as the cases described below, shown in Figure 3.19

- *Physical coverage.*
  There is a decrease to zero of the physical coverage, in one ore more internal regions of the studied contigs, but all the other tracks do not give information about possible misassemblies. This situation could be explained looking at the low complexity and repeated signals. It could happen that in some very difficult stretches of bases, that are very repeated or low complex ones, the reads are not able to map. The

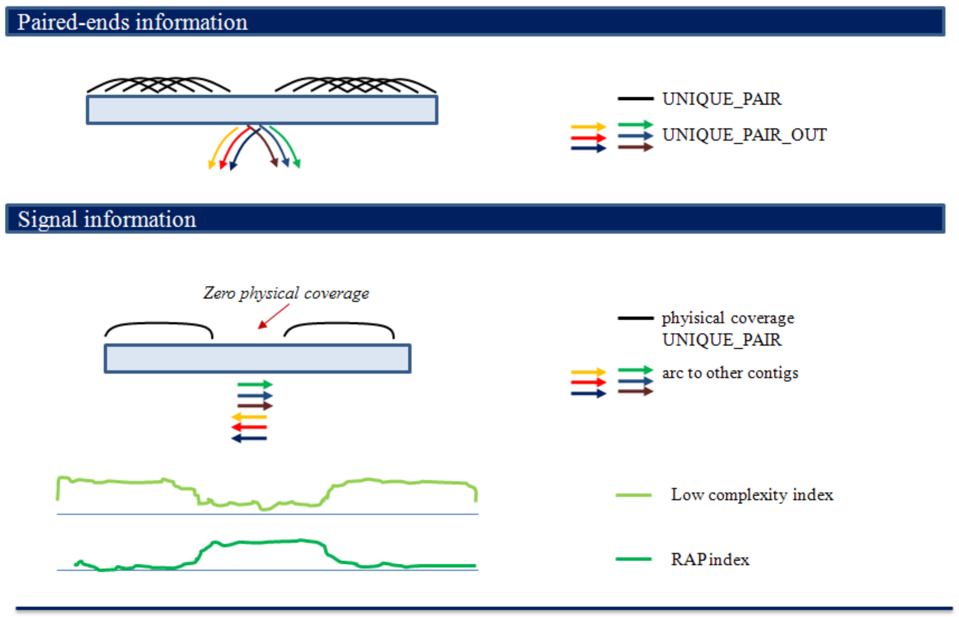


Figure 3.19: *In the figure is represented an ambiguous situation in which the contig seems to be misassembled, due to the low physical coverage and all the outer pairing. However, it is only a very repeated region, as could be seen the low complex and repetitive indexes.*

consequence is a lack of data during the creation of this signal, but this does not mean that there is a critical situation.

- *outer pairing.*
  There could be contigs that contain wrong arrows. The typical situation involves a very long fragment, in which a lot of repetitive sequence have been collapsed, and all the outer evidences align in these positions. This case has not to be confused with an insertion of a wrong region, and it is confirmed by the repetitive signal, which value will be very high in these points.

Input

Type	Explanation	Format
UNIQUE PAIR	see output of the Pairing module	GFF
UNIQUE PAIR OUT	see output of the Pairing module	GFF
UNIQUE WRONG D	see output of the Pairing module	GFF
UNIQUE SINGLE	see output of the Pairing module	GFF
reference genome file	the genome sequence, on which reads have been mapped	multi-FASTA

Output

Type	Explanation	Format
misassembly file	file with all information about contig misassemblies	GFF
evidence file	for each misassembly, it defines all evidence that confirm it	GFF

3.5.9 Make Arcs

As suggested by the name, the “Make arcs” module creates the connections (also called arcs, see Figure 3.20) between contigs, using the unique pair out class of data (see Section 3.5.6). An arc is obtained as the sum of all the paired-ends that join the same couple of contigs, and it is defined by the following parameters:

- *gap value*: it is an evaluation of the distance between the linked contigs;
- *order and orientation*: these are the information about which contig comes before, and in which orientation (5’-3’ or 3’-5’);
- *weight*: it is the number of paired-ends that connect the couple of contigs.

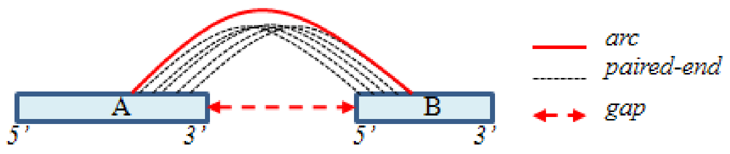


Figure 3.20: *An arc (red line) between two connected contigs, obtained by the sum of the five paired-ends (black lines) that induce it. Contig A comes before contig B, with the 5' 3' orientation.*

How is it possible to extract all the above information from the unique pair out class? To simplify the following discussion, it will be used the SOLiD notation: the two ends of a mate-pair are called respectively R3 and F3. During the alignment phase, both of them can align in the positive or negative strand, and there are four possible events: R3 aligned in plus (R3/+) or minus (R3/-), and F3 aligned in plus (F3/+) or minus (F3/-). The read pairing produces one of the possible combinations reported in the table below:

Paired-end		**Paired-end**	
R3/+	F3/+	F3/+	R3/+
R3/+	F3/-	F3/+	R3/-
R3/-	F3/+	F3/-	R3/+
R3/-	F3/-	F3/-	R3/-

For each couple of linked contigs, the algorithm divides all the involved paired-ends into the eight classes, that are then collapsed into four ones. In fact, thanks to the symmetric nature of the problem, shown in Figure 3.21, the combinations can be pairwise associated. Some statistical tests have confirmed that during the mate-pair division process, the corresponding counters have been filled by the same percentage.

After that, the paired-ends are labeled as correct or not, evaluating their length. It is not easy to define a distance measure for disjoint contigs, where the gap between them is unknow. However, it is possible to fix a threshold (τ), over which the data are not valid.

$$\tau = \mu + n \cdot \sigma$$

where μ and σ are respectively the mean value and the standard deviation computed for the library distribution, and n is a parametric value that defines the percentage of considered mate-pairs (generally $n = 2$ or 3). The

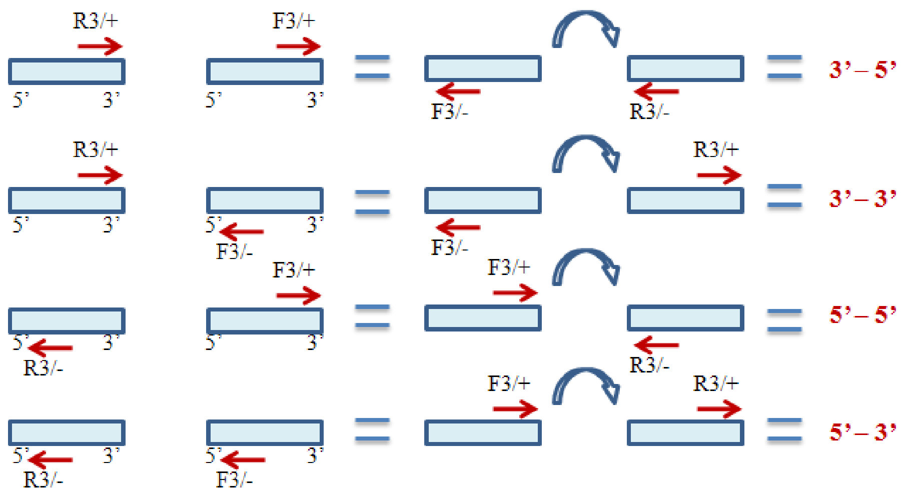


Figure 3.21: *The figure shows how the eight combination classes of paired-end could be reduced into four ones, due to the symmetrical nature of classes. For each class it is defined the induced order and orientation.*

paired-end is wrong if the length of the two pieces of the contigs spanned by the pair is greater than the parameter τ. This phenomenon could be explained by several reasons:

- introduction of background noises due to incorrect paired-end alignment. In this case, only few paired-ends should be wrong: if the paired-end would be an error, it would be an isolated case.
- a wrong distance evaluation caused by misassembled contigs. The consistency phase should have found some evidence about it.
- inaccuracy in the library length measure. If the used average insert has been wrongly evaluated, all paired-ends should have the same behavior.

For each class, the module divides the correct and uncorrect paired-ends, and for each division, it computes the induced gap length. In an easier approach, the distance between two contigs is computed as follow:

$$distance_i = \mu - \frac{1}{N}\sum_{i=1}^{N} l_i \tag{3.1}$$

where μ is the average insert length of the library; N is the number of paired-ends of the same class that connect the two contigs; and l_i is the the number of contig bases covered by the paired-end i. At the end of this process, a couple of contig has four associated counters, one for each class, and the division between good and not good paired-ends.

In an ideal error-free situation, each couple of contigs would have an unique kind of connection with a correct distance. But, in presence of contig mis-assembly, or wrong paired-ends, more than one counter could be filled for each couple, and some of them could have a wrong associated distance. In these situations, the module should filter the error, and decide which is the correct connection. To do this, it considers that the most probable situation is associated to the largest counter.

The developed algorithm applies the operations:

```
IF $\exists ! c_i : c_i \neq 0$
        IF $\tau_{min} \leq |\{p_j \in c_i : d(p_j) \leq \mu + n \cdot \sigma\}| \leq \tau_{max} \rightarrow$ good arc;
    ELSE no good arc;
ELSE    IF $\tau \leq \frac{max_i |\{p_j \in c_i : d(p_j) \leq \mu + n \cdot \sigma\}|}{\sum_i |\{p_j \in c_i : d(p_j) \leq \mu + n \cdot \sigma\}|} \rightarrow$ good arc;
    ELSE no good arc;
```

The connection has an associated set of four counters $C = \{c_1, ..., c_4\}$, where c_i is the number of paired-ends of the class i, that connect the two contigs, and i are the four possible contig connection combinations. Each counter could be divided into two sub counters: one contains the number of paired-ends that respect the average insert value, while the other one considers the number of wrong paired-ends. The set thresholds are defined as: τ_{min} is the minimum number of reads, τ_{max} maximum number of reads, $\tau = \mu + n \cdot \sigma$.

Input

Type	**Explanation**	**Format**
insert length	an evaluation of the library insert length (mean value)	integer
library error	the error of the library (standard deviation)	integer
contig list	a file with the list of all contigs to consider	header file
minimum	the threshold of the minimum number of available arcs, to accept the arc	integer
maximum	the threshold of the maximum number of available arcs, to accept the arc	integer

Output

Type	Explanation	Format
good arc file	the file with all good connections between the contigs	txt
no good arc file	the file with all rejected connections between contigs	txt
log file	the file with all the done operations	txt

3.5.10 Scaffolding Engine

As suggested by the name itself, this engine is the heart of the program: it integrates all the recovered information into ordered and oriented set of consistent scaffolds. This is a tricky task due to the presence of a lot of ambiguous situations, in which it is not clear how contigs stay together. ConSort tries to extricate such doubtful scenarios and to solve inconsistent cases, but at the same time, whenever it meets uncertain solutions, it applies a conservative policy, avoiding the introduction of unwanted artifacts. It isolates the critical contigs, stopping the scaffold elongation, and it records the taken decisions. All the evidences and hypotheses used for the troubleshooting are stored in an output file, that will be very useful for a manual validation of the obtained results.

Graph problem

The graph is the mathematical structure that best models the scaffolding problem: the contigs are the nodes, while the mate-pairs form the arcs. More precisely, it is a directed graph with a topological order, in which: each contig occupies a fixed position in the space, it has an orientation, two of them could not share the same position if they are not similar, and the path moves in 5' versus 3' direction. In an ideal scenario, the graph could be decomposed into disjoint components, each of them representing a scaffold. Each component contains exactly two external and some internal nodes. The first ones are the initial and final points of the unique maximum linear path, that visits all the other contigs exactly once (see Figure 3.23 (a)). The research could be easily performed, beginning from the first node, and following the path until the last one is reached. This could be done in a polynomial time, proportional to the number of nodes contained in the scaffold. Unfortunately, the reality is very far from the theory, and the obtained network very often could be more complicated, falling in the NP-hard (non-deterministic in polynomial-time) class problem. Such a situation could be made feasible only using heuristic and approximation algorithms, that reduce the computational time.

With the graphical tool Cytoscape [3], the obtained overview was plotted (an example in Figure 3.22), and this showed the intricate shape of the graph. It was composed by some isolated cases with single or few joined nodes, and a big tangle of super-connected contigs. This last one, called decision node, has more than one inner and/or outer arc. In the first case, the lack of information does not allow to place it in the correct scaffold, but nothing can be yet done to overcome the problem. While, the second situation could be studied, and if there are enough evidences, it is possible to simplify it, solving the found ambiguities, and extracting the linear path.

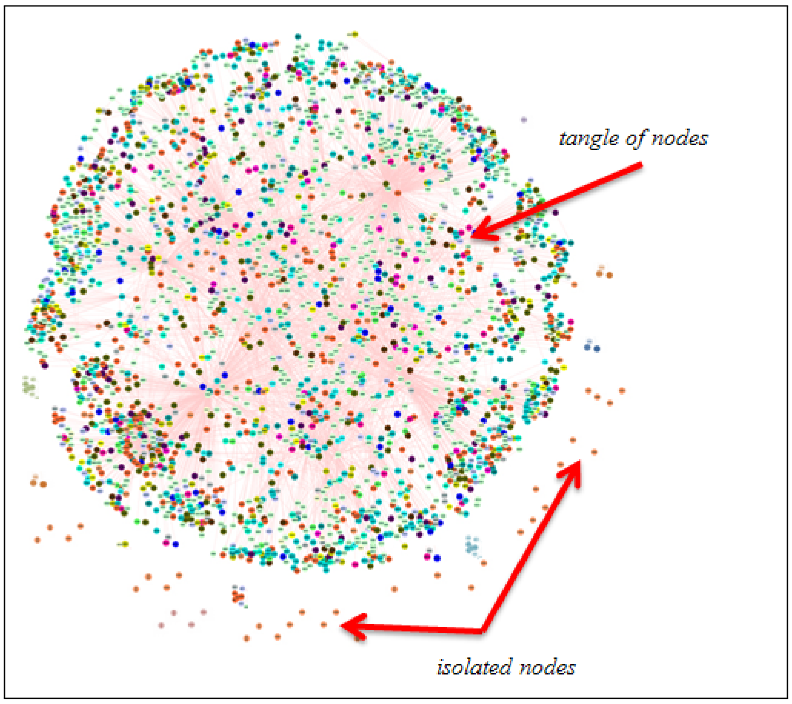


Figure 3.22: *In the figure, the graph of all nodes and their connections shows its complicated structure: a big tangle and some isolated nodes.*

As discussed in “Consistency analysis” Section 3.5.8, there are many reasons that cause a such situation, as sequencing errors, inconsistent paths and misassembled contigs. All of these are extrinsic motivations, that are only due to the sequencing and assembling phases, and for this reason, their level of resolution is strictly dependent on the quantity of available data. If there are enough evidences, they could be easily detected, and all useful information could be extracted for the future correction. Beyond this class, there is the intrinsic one, that is characterized by the genome composition, and its degree of repetition and complexity. This case is much more difficult

[3] `http://www.cytoscape.org/`

to solve, and sometimes even with a large amount of available data, it is not possible to find a good solution. It is essential to discover and define how all the previous factors could influence and complicate the shape of the graph. This allows to recognize them during the scaffolding phase, and execute *ad hoc* operations for their simplification.

Problem classification

In Figure 3.23, there is a schematic overview of all the possible shapes assumed by the graph:

(a) *Linear path.* It is the ideal graph, composed by two external contigs, that delimit the internal ones, and it has an unique maximum linear path: the output scaffold.

(b) *Bubble.* In this situation, the path could diverge before the misassembled contigs, and converge again after them. This is a theoretical situation, that depends on the library insert size. In fact, if the deletion is inside of an extended contig, and the mate-pairs are not long enough, the blue dashed arrows do not exist, and only the red ones underline the problem. If even these last ones are not available, it is not possible to discover the deleted region. Anyway, this situation could happen even if not all connection information are available, and it is not well-known how order the contigs.

(c/d) *Repeats.* There are two possible cases: the repeats are in the same scaffold (c), and the other ones that are positioned on separated paths (d.1). The first group causes a loop inside the graph, and it could be solved duplicating the repeat in the correct places. In the second one, the path converges in the critical contig, and then it diverges again. In fact, the only shared region is the repeated one, otherwise the scaffolds would be separated. To untie the knot, the common region has to be duplicated and inserted in both the paths.

(d) *Frayed rope.* Three different kinds of misassembly induce the same graph structure: the above discussed repeats (d.1), the insertion (d.2) and the chimeric problems (d.3). The path converges in the critical region, and then it diverges again. In fact, in all the cases the behavior is the same: the paths should be separated, but they are linked together through the shared sequence. For the last two cases, the shared sequence should be further analyzed and deconvoluted in such a way to generate two correct scaffolds.

the wrong region has to be cut away, and insert in the correct scaffold. The information coming from the consistency analysis will suggest the nature of the misassembly, and how it is possible to solve it.

(e) *Short dead-end branch.* This last situation is caused by sequencing or assembly errors, that produce two similar but disjoint contigs. The

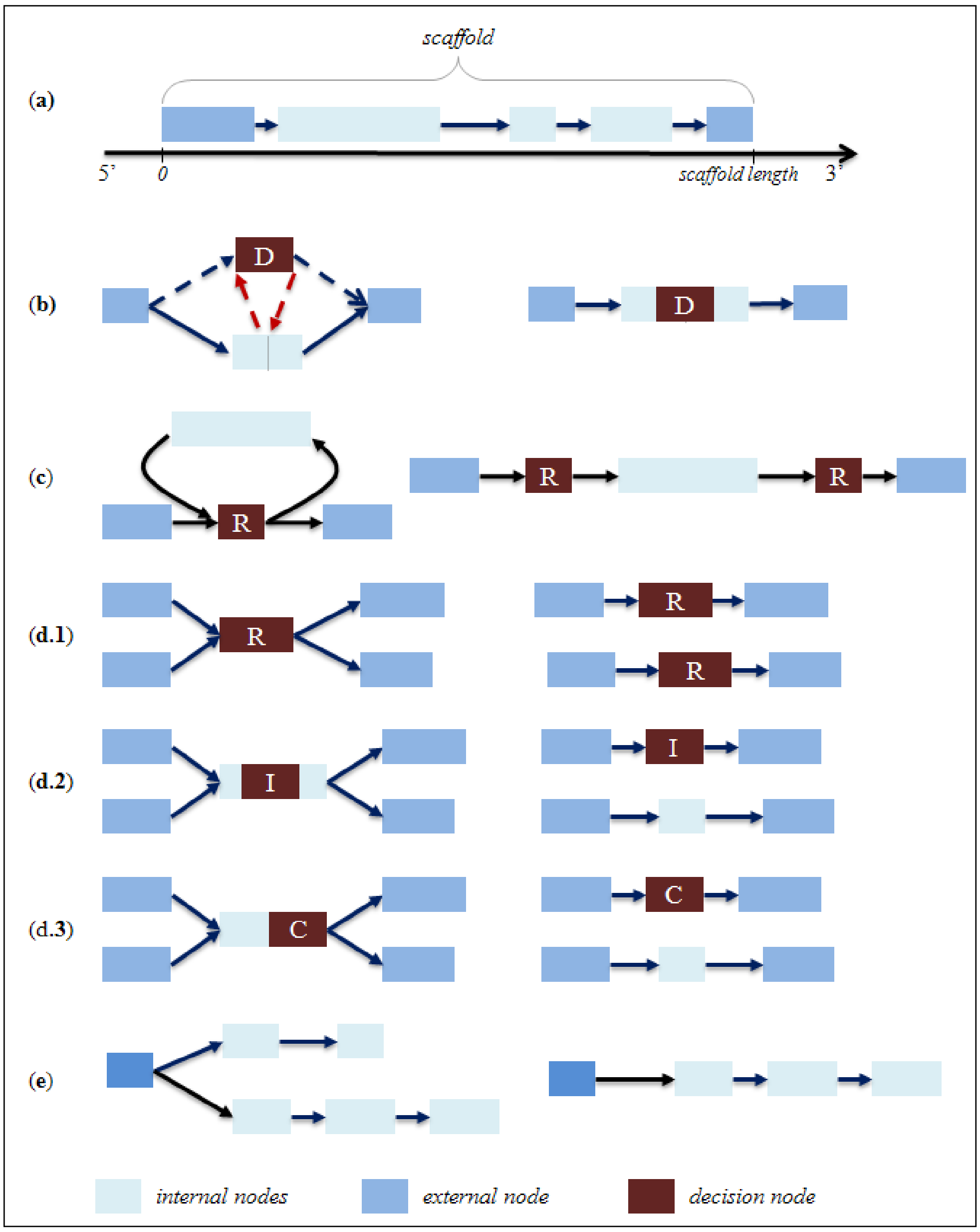


Figure 3.23: *In the figure, there are all the shape assumed by the graph, in presence of misassembled contigs, repeats or errors. The symbols stand for: I is the inserted region; D is the deletion region; C is a piece of the chimeric region and R is a repeated region.*

graph contains an alternative path, composed by a short one with the error inside, and a greater good one. With a similarity analysis between contigs, it is possible to solve the problem, and reconstruct the correct scaffold, deleting the wrong path.

ConSort is organized into two main steps: the pre-elaboration phase, on which the starting graph is created, and some local optimization are performed; the scaffolding phase, that tries to solve the ambiguous cases, and to transform the produced graph in a set of disjoint and linear components: the final set of scaffolds. In the next sections below, the two steps will be explained in more detail.

Scaffolding algorithm

Phase I: Graph creation and compression

The creation of the starting graph, with all the contigs and their connections, is a fundamental step to perform before the ambiguity resolution phase. In fact, the global achieved knowledge provides a problem overview that is very important for a complete analysis: a more restricted sight could penalize the scaffolding creation, falling in a local optimal solution.

Each node of the graph is characterized by the size of the represented contig and a list of all information about its internal consistency. Moreover, it has an associated state, used in the graph creation, that defines the level achieved in the process. There are three possible labels:

- *free*: it has not yet inserted in the graph, and consequently not yet explored;
- *pending*: it is connected to a previously inserted node;
- *inserted*: it has been added in the graph, with a fixed order and orientation.

Each arc is defined by the weight parameter, computed counting the number of mate-pairs that confirm the link: the higher is this value, the more probable is the connection. It has an associated length, that is equal to the size of the gap between the two adjacent nodes. Finally, it determines the type of link, specifying the order and orientation between contigs: 5'-3', 5'-5', 3'-5', 3'-3' (see Section 3.5.9). Each edge could be visited exactly once during the graph creation phase.

To create the graph, the following steps are applied:

Until there are free nodes in the component:

1. choose the next node to insert (**insertion policy**)
2. **insert** the node in the graph, in the **compressed** organization, setting all pending nodes;
3. if the inserted node was previously pending, perform the **close path** process, with optimization.

- *Insertion policy.*

  There are two possible insertion policy to use for the graph creation: "depth first search" (DFS) and "breadth first search" (BFS). The first approach expands each time the first inserted child, going deeper and deeper inside the graph, as suggested by the name. When the last visited node has no children, it comes back at the previous level, and continues the exploration with the next one. The other method visits all the nodes of a level before going down to the next deeper one.

  For the insertion problem, it is used a revised and adapted version of the breadth first search method, that well models the scaffolding problem. Each arc is marked as already visited, once it has been used for the first time. This allows to avoid endless research, caused by multiple loops, introduced in the graph by repeats and errors. Moreover, the insertion policy selects the more probable connected node as the next one to expand. This corresponds to the link with the greater weight. Each time a node is linked to a previously inserted one, that presents another path starting or ending on that side, a fork is created and a new alternative path begins.

- *Graph compression.*

  The possible graph structures described above (see Figure 3.23), highlight the most interesting kind of nodes are the decision ones. They are the critical points, from which alternative paths diverge or converge, that have to be untied for the scaffolding solution. On the other hand, the easiest class is the internal one, that does not require any kind of decision. Consequently, a linear path could be represented by a unique node, which size is the sum of all the contained contig and gap lengths. The transformation simplifies the problem structure, compressing all the redundant information. The obtained graph is more essential, it requires less computational time to move on it, and it shows only the critical points.

- *Close path process.*

  Each time a node is inserted in the graph, all the other ones linked but yet free are set as pending. Whenever during the insertion a pending node is about to be expanded, a *close path* event occurs: a already visited walk is confirmed by a new evidence. This is a very important situation, in which it is possible to verify if all the information are consistent, or if there are some errors to investigate. Moreover, a local optimization step could be performed to adjust the path.

  There are two kinds of such events, pictured in Figure 3.24: the *arc*

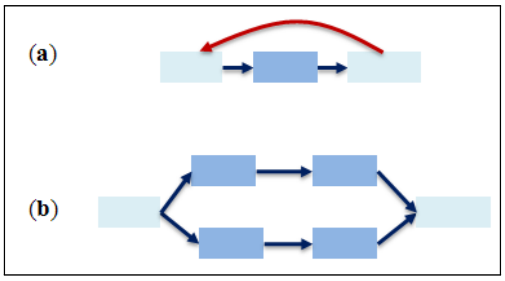


Figure 3.24: *In the figure, the are the two kinds of close path events: (a) arc bubble and (b) path bubble.*

and the *path bubble*. The first case is induced by a link: the last expanded node of a linear path is connected with a previously inserted one, closing the walk on it. The orientations and orders induced by all the involved arcs must be coherent, otherwise the nodes are labeled as problematic, and they will be solved in the further analysis. In case of consistency, the optimization process can be performed to adjust the gap length, as it will be soon described.
The second type produces a real bubble, in which there are two paths that diverge from a node, and then converge again in the same one. This could be due to a deletion problem, or to a lack of mate-pairs that has not allowed to insert the contig in a single path. The first situation is more complicated to resolve, and it will be left to the next phase. In the second case, the module controls if the two paths could be integrated in only one, satisfying the order, orientation and distance requirements. If this is possible, a new merged path is created, and an optimization step is performed; otherwise, the paths are marked as critical, and they will be studied in the second step of the algorithm.

- *Optimization phase.*

The sub-module performs a path length adjustment, integrating all the information coming from the mate-pairs. In an optimal situation, all the arcs would estimate the same gap size, and this no improvements should be required. In the reality, the probability at this event is very low, and it is important to find the best trade off among the distances suggested by all the involved edges.

The optimization problem is defined by the quadruplet $\Pi =< C, G, A, p >$, where:

- $C = \{c_1, ..., c_n\}$: is the set of n involved nodes, the contigs;

- $G = \{g_1, ..., g_{n-1}\}$: is the set of gaps among the nodes, the distances between a couple of contigs;
- $A = \{a_1, ..., a_m\}$: is the set of arcs that subtend the gaps;
- $p : A \times N \rightarrow R$: is the penalty function, that returns for each couple (*arc*, *length*) the associated score. This is proportional to the distance from the mean value defined by the library distribution: the greater is the value and the greater is the penalty.

Each configuration of G is defined by a $(n-1)$-upla of gap sizes $(s_j =< g_1^j, ..., g_{n-1}^j >)$, whose associated score is defined by the sum of all the penalties paid for the arcs, as follows:

$$score(s_j) = \sum_{i=1}^{m} p(a_i|s_j)$$

The set of all possible configuration $S = \{s_1, ..., s_M\}$ is called the *space of solutions*, and the optimization problem becomes the searching of the best configuration $\overline{s}$, with the minimum associated score:

$$\overline{s} = \arg \min_{s_j, j=1,...,M} score(s_j)$$

The minimization problem could require a lot of time in the research of the best solution, due to the large space dimension. For this reason, the application of an heuristic reduces the computational time, searching only into the region where it is more probable to find the best solution. These are the performed steps:

1. The algorithm starts with a local optimal configuration at time zero, $\overline{s}_0$. This has the greater number of arcs with zero associated score, and all the others ones (the remaining h) are to be improved. The optimal configuration is also the first selected one, from which starting the research.
2. At each time t, from the selected configuration other h are generated, avoiding repetition. Each one is obtained decreasing the score of an arc not yet optimal. The new found configurations are inserted in the set of not yet visited one, and the associated score is computed. If between them there is a (n-1)-upla with a better score than the current one, it is updated.
3. A pruning function is performed to discard all the new inserted configurations, from which it is not possible to derive a new one with a score better than the current optimal value.

4. The next node to expand is selected in and removed from the set of not yet visited configurations. This will have the best associated score.

5. The process is iterated from point (2), until there are nodes to visit; then, it terminates with the optimal value equal to the current one.

This first step requires a computational time proportional to $n \cdot \log n$ for the creation of the graph, plus the optimization phase execution, where n is the number of inserted contigs. The optimization phase is variable, and it depends on the nature of the the problem.

Phase II: problems resolution and scaffolding creation

This is the more complicated phase of the algorithm, where the ambiguities have to been solved to find consistent scaffolds. But, at the same time, it is not desirable to introduce new errors or artifacts.

All the critical nodes present in the compressed graph are to be analyzed and solved, as follows:

1. For each node, the adjacent ones and connected arcs are extracted.

2. If the extracted structure belongs to one previously defined (3.23)), it is solved using all the available consistency information and the defined rules.

   If the available information are not enough to solve the problem, the path is broken, and all the information and decisions are stored in the output file, otherwise the graph is adjust, correcting the nodes and connections, and performing optimization distances.

3. If the extracted structure is unknown, the path is broken and all the used decisions are stored in the output file.

The phase requires a computational time proportional to the number of critical points, that at least will be equal to the total number of contigs, plus the time required for the optimization phase. The allocated memory depends on the number of used contigs, arcs and information.

Input

The scaffolding modules returns the file with the assembly, and all information about the used evidences.

Type	Explanation	Format
contig list	a file with all contigs	multi-FASTA
arcs list	the file with all the arcs	txt
evidence list	the file with all information about inconsistent situations	GFF

Output

Type	Explanation	Format
scaffolding	a file with all scaffolds	multi-FASTA
decision list	the file with all taken decision during the tangle resolution, and information that have suggested them	GFF

3.5.11 Gap filling

The aim of this last module is the filling of the gaps between contigs. In fact, the scaffolding found the correct order and orientation among them, and it evaluates their distance, but it does not determine the sequence of the gap. The general idea is to substitute the stretch of N's with the correct sequence.

The module could be organized into a bioinformatical followed by a biological phase. The first one uses the unique single paired-ends, that have only one end aligned. This means that the other one can fall in the gap, for which the sequence is unknown, or it has been discarded during the alignment, because it requires too much mismatches to be mapped. The first case produces a set of reads that could align, but the

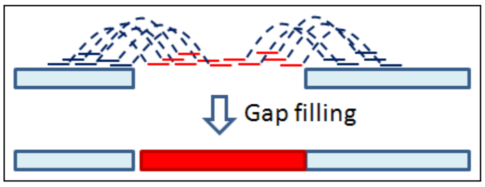


Figure 3.25: *In the figure, it is represented the gap filling process.*

corresponding genomic region is not assembled region and are anchored on the genome by the other side (see Figure 3.25). After the detection of all these paired-ends, they can be assembled creating the consensus sequence, reducing if possible the gap length or even close it. The process produces a set of new contigs, that can be added in the scaffolding. So, the module will perform an integration of the new obtained sequencing in the previous scaffolding.
At the end, the biological phase would be performed. It consists in a set of PCR, that should be suggested by the ConSort output, in term of primer design and length of the produced fragments, that will help in the definition of all not closed gap yet.

Input

Type	**Explanation**	**Format**
genome sequence	the set of all produced scaffolds, with N's	multi-FASTA
pairing output	all the files produced in the pairing phase	GFF

Output

Type	**Explanation**	**Format**
genome sequence	the set of all produced scaffolds, with less N's, if possible	multi-FASTA

3.6 Testing ConSort: the results

ConSort has been tested on the tomato (*Solanum lycopersicum*) genome, which has been obtained in the "International Tomato Genome Sequencing Project" [4]. The project began in 2004, and it is an international consortium including participants from Korea, China, the United Kingdom, India, the Netherlands, France, Japan, Spain, Italy and the United States. The first tomato genome draft is going to be released. I used all the produced data to

[4]`http://solgenomics.net/genomes/Solanum_lycopersicum/index.pl`

study, test and improve ConSort, and a lot of work has been done in this direction.

The test has been performed on a 8 dual core system with 72GB of RAM, with Linux openSUSE operating system platform. With this computating power, it is possible to execute the whole ConSort pipeline in about a week, parallelizing the independent operations. The program starts with a set of assembled contigs and mate-pair libraries, and it returns the output scaffolding.

454 input data

The project was performed using the GS-FLX Titanium Roche/454 sequencer to create the starting set of contigs. The consortium produced about a 25 x coverage of whole genome shotgun (WGS) fragments and paired-end libraries of ∼350 bp. The last one were divided into: 3kb and 20kb (LPE). All the data have been incrementally assembled using Newbler 2.3, achieving the following results:

Scaffold Metrics		**Large Contig Metrics**	
Measure	*Value*	*Measure*	*Value*
Number of Scaffolds	7,409	Number of Contigs	62,716
Number of Bases	794,608,225	Number of Bases	748,398,241
Average Scaffold size	107,249 bp	Average Contig size	11,933 bp
N50 Scaffold size	4,487,776 bp	N50 Contig size	48,653 bp
Larger Scaffold size	20,687,090 bp	Larger Contig size	575,502 bp

All Contig Metrics

Number of Contigs	118,692	Number of Bases	762,497,151

SOLiD input data

Two mate-pair libraries were used for the scaffolding:

- *2kb insert length*: about 800 millions of reads of 25 + 25 bp long were produced using SOLiD 3;
- *10kb insert length*: about 1,200 millions of reads of 50 + 50 bp were produced, using SOLID 3.

In the below table, there is a detailed report about all produced data.

	2kb mate-pair library		**10kb mate-pair library**	
	F3	*R3*	*F3*	*R3*
RUN_1	196,585,507	196,698,072	312,753,851	313,076,508
RUN_2	211,325,578	211,960,463	316,643,202	317,395,412

where RUN_1 and RUN_2 indicate the sequencing two of the two slides of the same library, while F3 and R3 represent the name of the two mate-pair reads, in SOLiD notation.

To have a general idea about the quantity of available data, the achieved sequence and physical coverage was computed. The first value measures the number of times a base is covered on average, determined by the formula:

$$\textit{sequence coverage} = \frac{r_l \cdot n_r}{L}$$

where r_l is the read length, n_r is the total number of reads, and L is the genome length. The greater is this value, and the higher is the probability that each base would be covered enough to have a good confidence on the information hold by the reads that map there.
The second quantity estimates on average how many times a base is spanned by the mate-pairs, as defined in the formula:

$$\textit{physical coverage} = \frac{i_l \cdot n_c}{L}$$

where i_l is the experimentally estimated library insert length, and n_c is the average value of potential couples.

The used assembly and the starting mate-pair library achieve good coverage values: 26.8 x sequence and 1070 x physical coverage for the 2kb library, and 82.6 x sequence and 8254 x physical coverage for the 10kb one. The parameter were set to:

- r_l = 25bp and 50bp;
- i_l = 2kb and 10kb;
- n_r is given by the sum of all the reads, $816,569,620$ and $1,259,868,973$;
- n_c = is the average number of couples for both runs, $407,911,085$ and $629,397,053$;

- $L =$ is the number of total bases contained in all the contigs, $762,497,151$.

Originally, other two libraries were produced respectively of 6kb and 7kb size. The analysis showed that the first one was very redundant (about 90% of redundancy), and so it was discarded. While, during the creation of the second one, some sequencing problems brought to the production of one only end of the mate-pair, and consequently, it was not possible to use it for the pairing aim.

Spectral correction

The first step of the pipeline applied the error correction to the SOLiD data. This phase has been performed in four hours of 10 cpu work for each set of reads, and it did not require a considerable quantity of RAM memory.

Pre-pairing

The module divides the paired-ends into the two groups: the paired-ends that are made of two reads, and the one that are composed by a single read. This last kind of data is useless for the future analysis, that is based on paired-reads. The number of reads that are not coupled is a small percentage, less than 0.5%, as it could be seen in the Table 3.1.

Trimming and alignment

The trimming and alignment phase were executed in 10 hours of computation time for each mate-pair class, using 16 cpu and about 30GB of RAM.

The trimming was performed using the following parameters: $w = 2$ (external window size), $t = 7$ (associated quality threshold) and $l = 20$ (minimum read length) for the 2kb library; $w = 2$, $t = 17$ and $l = 20$ for the 10kb one. The alignment was run with the same parameter values for the two libraries, set to: at most 90% of identity for each accepted alignment, zero gaps and the best-hit options (only the best alignment are returned).

In the table below, the obtained results are shown. They have been divided per library and RUN. The row meaning is:

- *single*: number of unpaired reads;
- *filter*: number of reads that have been discarded by the trimming step;
- *no_filter*: number of not filtered reads;
- *total* = the total number of input reads, equal to the sum of the above rows (single + filter + no_filter);

- *aligned* = number of not filtered reads, that have been aligned.

		2kb mate-pair library		**10kb mate-pair library**	
		F3	*R3*	*F3*	*R3*
	single	396,470	509,035	105,405	428,062
	filter	40,321,545	39,212,063	69,896,969	99,678,523
1	no_filter	155,867,492	156,976,974	242,751,477	212,969,923
	total	196,585,507	196,698,072	312,753,851	313,076,508
	aligned	121,170,782	128,758,109	176,665,593	181,509,260
	single	271,611	906,496	370,049	1,122,259
	filter	48,620,011	47,494,896	36,084,069	39,556,655
2	no_filter	162,433,956	163,559,071	280,189,084	276,716,498
	total	211,325,578	211,960,463	316,643,202	317,395,412
	aligned	122,982,237	135,601,290	185,574,495	177,448,682

Table 3.1: *The table contains the pairing, trimming and alignment results. The data have been divided per library, RUN, and type of reads.*

The Table 3.2 shows that good results have been obtained with the input data elaboration. Only few reads have been discarded, about 10-20% for all the cases, and about 60-70% of reads were successfully aligned. This means that the starting libraries have a good quality, which can be obtained during the spectral correction and trimming phases.

Pairing

The pairing was performed computing the average insert size μ and standard deviation σ of each library. After that, all the mate-pairs were classified as explained in Section 3.5.6. The accepted distance length was defined considering the interval $\mu \pm 3\sigma$, who are around 99.7% of all the data. All the links, whose distance fell in the range, were classified as unique pair, while all the other were put in the unique pair wrong distance. This last group is populated by background noise, or data that map on misassembled contigs, and consequently their length has been changed. At the end of this process, the following results were obtained:

As it was expected, the shorter library has a lower standard deviation than the greater one, ant it is more precise in the mate-pair length evaluation.

		2kb mate-pair library		**10kb mate-pair library**	
		F3	*R3*	*F3*	*R3*
1	single	0.20	0.26	0.03	0.14
	filter	20.51	19.94	22.35	31.84
	no_filter	79.29	79.81	77.62	68.02
	aligned	61.64	65.46	56.49	57.98
2	single	0.14	0.46	0.12	0.36
	filter	24.73	24.15	11.54	12.63
	no_filter	82.63	83.15	89.59	88.39
	aligned	62.56	68.94	59.34	56.68

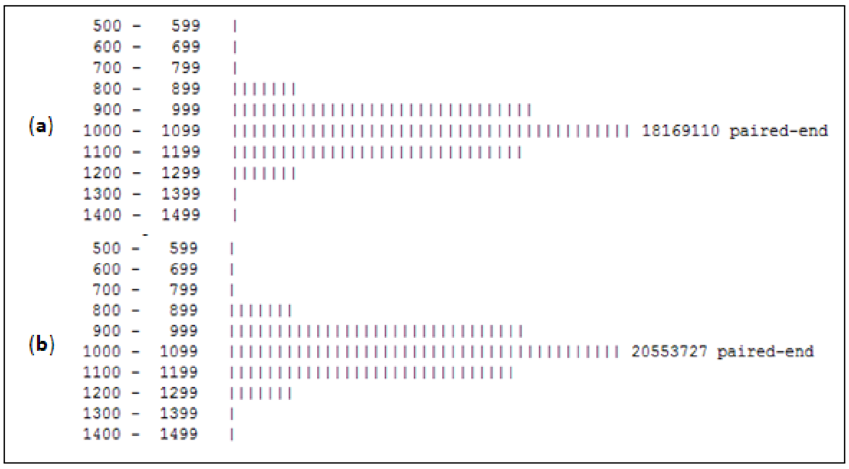
Table 3.2: *The table contains the percentage of the unpaired, discarded and aligned reads in respect to initial total number of reads.*

	2kb mate-pair library		**10kb mate-pair library**	
	RUN_1	*RUN_2*	*RUN_1*	*RUN_2*
Mean value	1,048	1,049	8,155	8,152
Std. deviation	97	97	955	955
unique pair	51,564,576	58,316,542	68,820,506	67,540,962
unique single	34,190,792	37,094,222	78,974,981	93,858,722
unique wrong distance	1,731,500	1,954,277	1,808,897	1,695,186
unique pair out	14,488,566	11,988,744	40,470,074	39,597,626

This is very useful for the gap estimation the: lower is the standard deviation value, and the higher is the accuracy of the evaluation. This is shown in the Figure 3.26 and 3.27: the 2kb library has a more tight curve, than the 10kb one.
Moreover for both libraries, the cardinality of the unique pair is greater than the unique pair out, but evaluating the percentage of the unique pair with respect to the unique pair out, this is greater for the 2kb library than the 10kb one. Even this result was expected, because the short library can be

more easily maps inside contigs than the greater one.

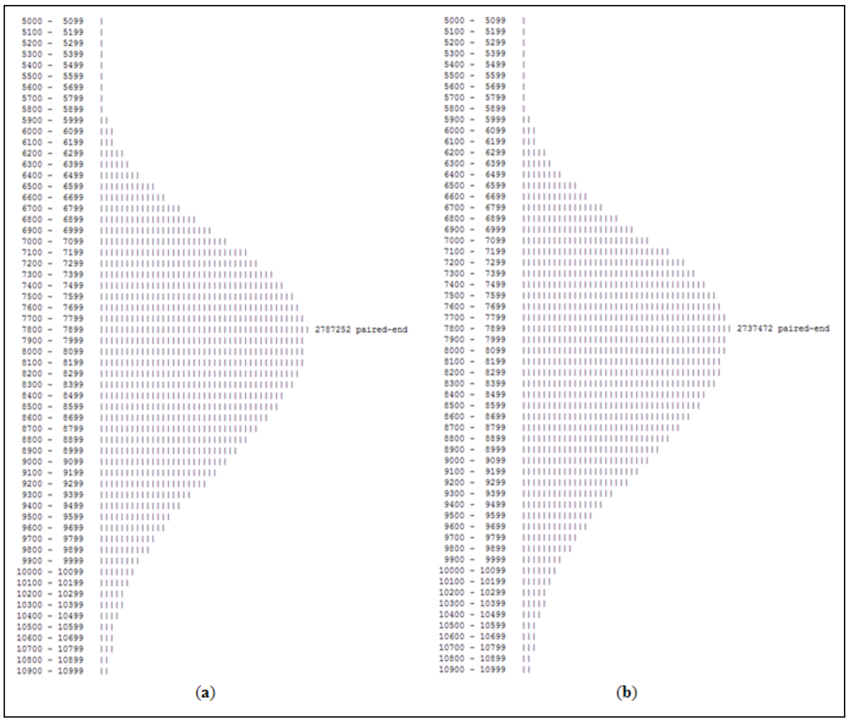
The library distribution was computed with all the mate-pairs that map on the same contig. For each library and each run, the corresponding curves were plotted, as shown in Figure 3.26 and 3.27.



(a)
```
  500 -   599  |
  600 -   699  |
  700 -   799  |
  800 -   899  |||||||
  900 -   999  |||||||||||||||||||||||||||||||
 1000 -  1099  ||||||||||||||||||||||||||||||||||||||||| 18169110 paired-end
 1100 -  1199  ||||||||||||||||||||||||||||||
 1200 -  1299  |||||||
 1300 -  1399  |
 1400 -  1499  |
```

(b)
```
  500 -   599  |
  600 -   699  |
  700 -   799  |
  800 -   899  |||||||
  900 -   999  |||||||||||||||||||||||||||||||
 1000 -  1099  ||||||||||||||||||||||||||||||||||||||||| 20553727 paired-end
 1100 -  1199  ||||||||||||||||||||||||||||||
 1200 -  1299  |||||||
 1300 -  1399  |
 1400 -  1499  |
```

Figure 3.26: *The distribution of the two run of the 2kb library.*



```
 5000 -  5099  |
 5100 -  5199  |
 5200 -  5299  |
 5300 -  5399  |
 5400 -  5499  |
 5500 -  5599  |
 5600 -  5699  |
 5700 -  5799  |
 5800 -  5899  |
 5900 -  5999  ||
 6000 -  6099  |||
 6100 -  6199  |||
 6200 -  6299  |||||
 6300 -  6399  ||||||
 6400 -  6499  ||||||||
 6500 -  6599  |||||||||||
 6600 -  6699  |||||||||||||
 6700 -  6799  ||||||||||||||||
 6800 -  6899  |||||||||||||||||||
 6900 -  6999  ||||||||||||||||||||||
 7000 -  7099  |||||||||||||||||||||||||
 7100 -  7199  |||||||||||||||||||||||||||||
 7200 -  7299  |||||||||||||||||||||||||||||||
 7300 -  7399  ||||||||||||||||||||||||||||||||||
 7400 -  7499  ||||||||||||||||||||||||||||||||||||
 7500 -  7599  ||||||||||||||||||||||||||||||||||||||
 7600 -  7699  |||||||||||||||||||||||||||||||||||||||
 7700 -  7799  ||||||||||||||||||||||||||||||||||||||||
 7800 -  7899  ||||||||||||||||||||||||||||||||||||||||| 2787252 paired-end
 7900 -  7999  ||||||||||||||||||||||||||||||||||||||||
 8000 -  8099  ||||||||||||||||||||||||||||||||||||||||
 8100 -  8199  ||||||||||||||||||||||||||||||||||||||||
 8200 -  8299  |||||||||||||||||||||||||||||||||||||||
 8300 -  8399  ||||||||||||||||||||||||||||||||||||||
 8400 -  8499  ||||||||||||||||||||||||||||||||||||
 8500 -  8599  |||||||||||||||||||||||||||||||||||
 8600 -  8699  |||||||||||||||||||||||||||||||||
 8700 -  8799  |||||||||||||||||||||||||||||||
 8800 -  8899  |||||||||||||||||||||||||||||
 8900 -  8999  |||||||||||||||||||||||||||
 9000 -  9099  |||||||||||||||||||||||||
 9100 -  9199  |||||||||||||||||||||||
 9200 -  9299  |||||||||||||||||||||
 9300 -  9399  ||||||||||||||||||
 9400 -  9499  ||||||||||||||||
 9500 -  9599  ||||||||||||||
 9600 -  9699  |||||||||||||
 9700 -  9799  |||||||||||
 9800 -  9899  ||||||||||
 9900 -  9999  ||||||||
10000 - 10099  |||||||
10100 - 10199  ||||||
10200 - 10299  |||||
10300 - 10399  |||||
10400 - 10499  ||||
10500 - 10599  |||
10600 - 10699  |||
10700 - 10799  |||
10800 - 10899  ||
10900 - 10999  ||
```

(a)

```
 5000 -  5099  |
 5100 -  5199  |
 5200 -  5299  |
 5300 -  5399  |
 5400 -  5499  |
 5500 -  5599  |
 5600 -  5699  |
 5700 -  5799  |
 5800 -  5899  |
 5900 -  5999  ||
 6000 -  6099  |||
 6100 -  6199  |||
 6200 -  6299  |||||
 6300 -  6399  ||||||
 6400 -  6499  ||||||||
 6500 -  6599  |||||||||||
 6600 -  6699  |||||||||||||
 6700 -  6799  ||||||||||||||||
 6800 -  6899  |||||||||||||||||||
 6900 -  6999  ||||||||||||||||||||||
 7000 -  7099  |||||||||||||||||||||||||
 7100 -  7199  |||||||||||||||||||||||||||||
 7200 -  7299  ||||||||||||||||||||||||||||||||
 7300 -  7399  ||||||||||||||||||||||||||||||||||
 7400 -  7499  ||||||||||||||||||||||||||||||||||||
 7500 -  7599  ||||||||||||||||||||||||||||||||||||||
 7600 -  7699  |||||||||||||||||||||||||||||||||||||||
 7700 -  7799  ||||||||||||||||||||||||||||||||||||||||
 7800 -  7899  ||||||||||||||||||||||||||||||||||||||||| 2737472 paired-end
 7900 -  7999  ||||||||||||||||||||||||||||||||||||||||
 8000 -  8099  ||||||||||||||||||||||||||||||||||||||||
 8100 -  8199  |||||||||||||||||||||||||||||||||||||||
 8200 -  8299  |||||||||||||||||||||||||||||||||||||||
 8300 -  8399  ||||||||||||||||||||||||||||||||||||||
 8400 -  8499  ||||||||||||||||||||||||||||||||||||
 8500 -  8599  |||||||||||||||||||||||||||||||||||
 8600 -  8699  |||||||||||||||||||||||||||||||||
 8700 -  8799  |||||||||||||||||||||||||||||||
 8800 -  8899  |||||||||||||||||||||||||||||
 8900 -  8999  |||||||||||||||||||||||||||
 9000 -  9099  |||||||||||||||||||||||||
 9100 -  9199  |||||||||||||||||||||||
 9200 -  9299  |||||||||||||||||||||
 9300 -  9399  ||||||||||||||||||
 9400 -  9499  ||||||||||||||||
 9500 -  9599  ||||||||||||||
 9600 -  9699  |||||||||||||
 9700 -  9799  |||||||||||
 9800 -  9899  ||||||||||
 9900 -  9999  ||||||||
10000 - 10099  |||||||
10100 - 10199  ||||||
10200 - 10299  |||||
10300 - 10399  |||||
10400 - 10499  ||||
10500 - 10599  |||
10600 - 10699  |||
10700 - 10799  |||
10800 - 10899  ||
10900 - 10999  ||
```

(b)

Figure 3.27: *The distribution of the two run of the 10kb library.*

The step required about four hours per run, using few memory and only a cpu.

Redundancy

The redundancy was performed merging together the corresponding files of the two runs for each library. It was executed on the unique pair and unique pair out file, and the output results show that the 2kb and 10kb library were respectively about 20% and 40%redundant.
With the performed data elaboration, all data have been cleaned and filtered, and only the good ones have be used for further analysis. The real coverages values that have been computed are: 79,27 x physical and 3,90 x sequence for the 2kb library, and 273,87 x physical and 3,88 x sequence for the 10kb one.
The module runs in two hours per file library, requiring about 30% of memory to allocate the whole genome and all the other useful information. At the end, it produced the not redundant files, one for each type of class and library.

Consistency

The module was firstly proved on simulated data, using the physical coverage track of the 2kb library, to demonstrate that it is able to recognize the misassembled contigs.
First of all, the data set was created, selecting some well assembled contigs, and introducing a fixed number of bases inside them. After that, the lengths of the insertion were selected dependently to the library average size of the library (about 1,000 bp) and the standard deviation , equal to 97. In Figure 3.28, the obtained results are shown. The enclosed region shows how the physical coverage decreases to zero, with the increase of the length of the contig.

- **insert length = 300 bp**. A small insertion minor or equal to 3σ, is not easily identified by noise of variation of the library length. In fact, as it could see in the Figure 3.28, the signal variation with respect to the original one could be confused with the normal fluctuation of the curve. This is due to the borderline situation, in which the mate-pair could still fall in the unique pair class, even if not all fallen.

- **insert length = 500 bp**. In this case, the insertion is greater than the allowed range for the mate-pair value, and the variation could be easily detected.

- **insert length = 1000 bp**. This is equal to the previous situation.

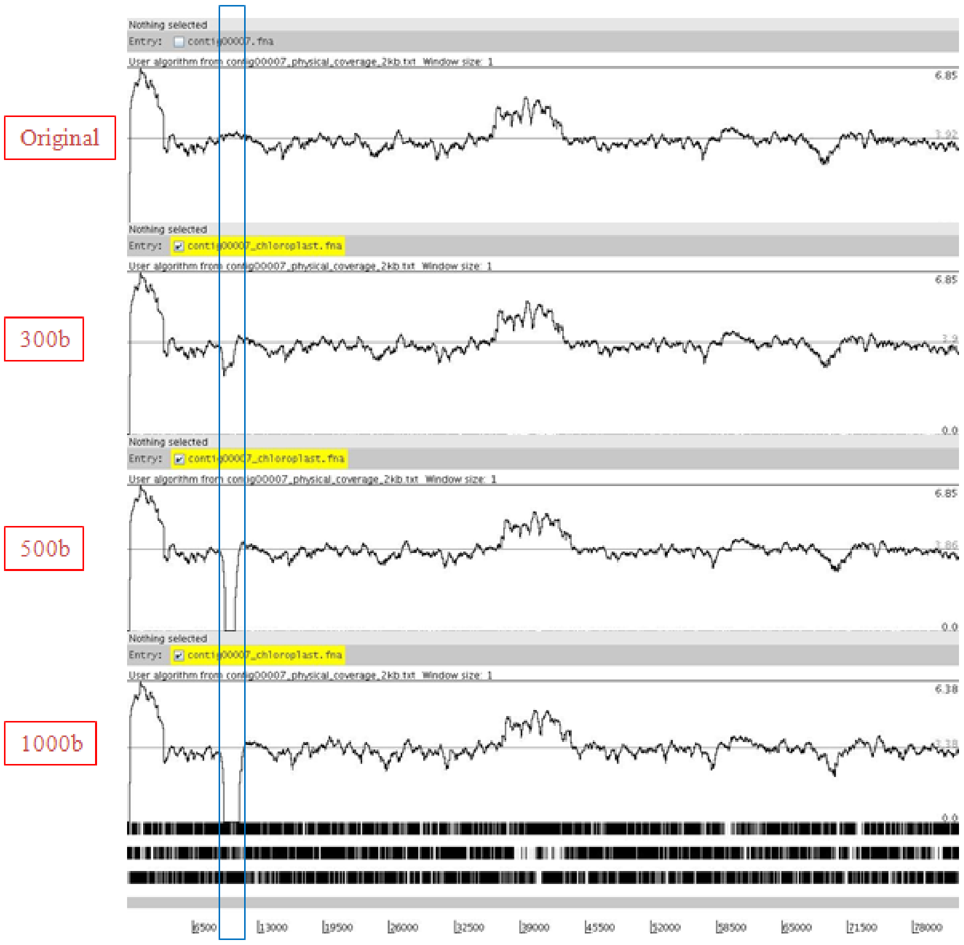


Figure 3.28: *In this figure is shown the physical coverage behavior for the original contig (the upper one), and the modified one (the following ones), with the inserted bases of 300 bp, 500 bp and 1,000 bp.*

The insertion size is still too small to allow the mapping of the mate-pairs on it.

The first test demonstrated that the strategy applied for the consistency analysis works well, and it was possible to continue the controls. The next validation phase was performed on a real test set. In this case, it was found a contig labeled as misassembled by the module, and it was manually controlled. The Figure 3.29 is a screenshot of the GBrowse centered in the region. It is composed by: the signal with all contigs, their position, length, order and orientation, the physical coverage (2 and 10 kb), the outer pairing (2 and 10 kb), the low complexity and repetitive (RAP) indexes.

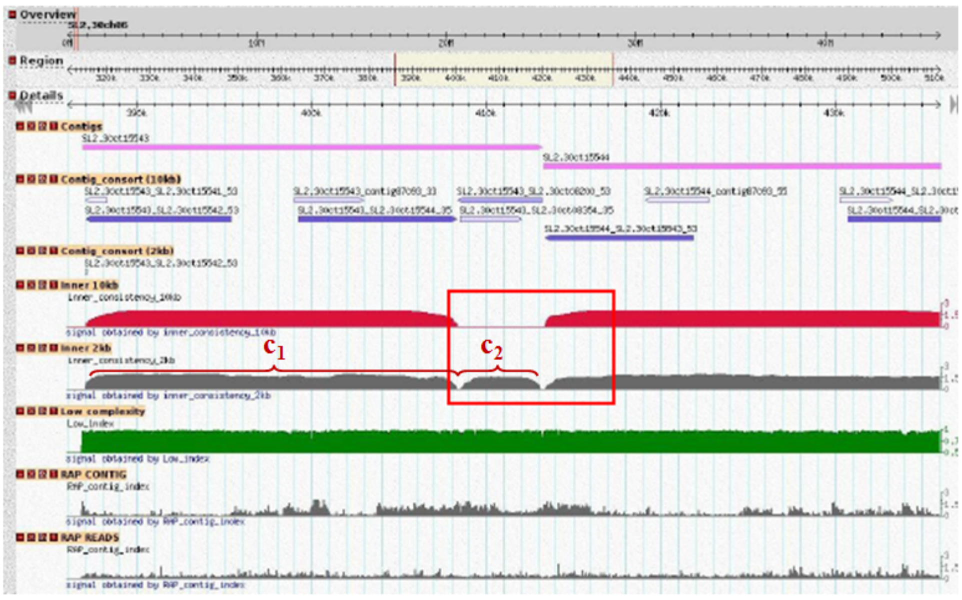


Figure 3.29: *The GBrowse screenshot shows the misassembled contig. The red rectangle encloses the region in which the physical coverage falls to zero, highlighting the potential misassembly.*

The rectangle encloses the region of zero physical coverage for the 2 and 10 kb library. The first one shows an evident decreasing of the signal, and an unique critical point, suggesting that this could be a deletion or chimeric situation. The starting contig is divided into two segments: the left part C_1 of about 210 kb, and the right one C_2 of about 4-5 kb. The second fragment is smaller than the longer insert library, and this explains why the physical coverage is not detected. The outer pairings analysis shows that C_1 and C_2 are connected to other disjoint contigs, confirming the hypothesis of a misassembled contig. Besides, these links involved different contigs, and consequently the misassembly could be a chimeric region, rather than a deletion problem. Finally, the repetitive index shows the presence of a degenerate stretch of bases, around the critical point.

Scaffolding

It is not easy to test the performance of the scaffolding module on a new genome. In fact, it does not exist a genome to use as reference one. A simple solution would be a comparative test, in which the obtained sequences are controlled with the output of other assemblers. Unfortunately, this is not a feasible idea to evaluate the real performances of the program, because they are not able to correctly assembly the whole genome, creating a valid test set.

In the beginning of the tomato genome project, a set of BAC clones were sequenced with the Sanger method, and were used to construct a draft of euchromatic regions of the genome. This is a good test set to use for the validation. From the output of ConSort and Newbler were extracted the contigs that mapped on a BAC-based already sequenced region of around 500kbp, and the estimated distanced were compared. The obtained results are displayed in the Table 3.3.

Name of the ordered contigs	**Real gap size**	**Gap size (1kb)**	**Mate-pair number (1kb)**	**Gap size (8kb)**	**Mate-pair number (8kb)**	**Newbler gap**
53074-53073	4401	ND	0	4173	28	3732
53073-53072	195	158	31	-12	94	663
53072-53071	-50	-89	16	-495	48	538
53071-24639	-25	-58	15	-464	9	/
24639-53070	-39	-15	28	27	12	/
53071-53070	2042	ND	0	1924	38	917

Table 3.3: *In the table there is a brief test set on which the performance on gap estimation of ConSort and Newbler have been tested.*

The results show that ConSort works bitter well than Newbler, in fact it is always able to give a better evaluation of the gap size. Moreover, Newbler loses some contigs, creating bigger gaps, that are filled by small ones instead recovered by ConSort . It can be said, the 2kb library is more accurate than 10kb one, due to its lower standard deviation.

3.7 System considerations and Conclusions

As could be seen in the previous sections, ConSort is a complete and efficient program both at functional and at executional level. It orders and orients all the contigs, trying to solve the ambiguous and very complicated situations that could occur. At the same time, it is very conservative, because it does not introduce unwanted errors and artifacts. Moreover, it records all data used in the taken decisions, to allow an easy and manually check-up of the obtained scaffolding. It performs all these operations in a reasonable time and with few memory allocation. Below, it will be briefly summarized the weaknesses, strength, properties and characteristics of the method.

- **ConSort functions: consistency and finishing**.
  ConSort has been integrated with two useful functions: consistency analysis and finishing process. The first one corrects all the errors present in the starting assembly, taking advantage from all the available data, and not requiring further ones. This feature is very important because it revises the existing genome sequence, solving possibly problems, and overcoming the limits of the sequencers and assemblers that have produced it.

  The second skill is the gap closure execution: it reconstructs the correct sequence and defines the real distance between contigs where and when it is possible. The process is very interesting, because it allows to reduce the number of N's present in the sequence, and to reconstruct the original genome.

- **Output scaffolding quality**.
  The results of the produced scaffolding depend very strictly on the goodness of the starting set of contigs and of the assembler used to produce them. This is due to the nature of ConSort, that based its analysis on already existing assembly. If the produced 454 coverage is too low, the created fragments will be very short, and not all the genome could be covered with a consequent high level of large gaps. ConSort could not resolve this problem, because if there are few input data, reasonably it is not able to create something out of nothing. On the other hand, if the used assembler is not able to correctly create the consensus, it would introduce a lot of errors, reducing the final quality of the assembly. If there are enough mate-pair libraries of different length, its consistency function is able to solve misassembly present in the genome, and to improve the quality of the final sequence.

- **Number of produced scaffolds**.
  The output number of produced scaffolds derives from the goodness of the starting assembly, the number of mate-pair libraries, and the variability of their insert size. It is not easy to determine the best library value, because this depends on the average gap length, that is initially unknow. If the genome is composed by very spread contigs, with great distance between them, it is required a set of mate-pairs with very long insert, to connect them. As a matter of fact, if the library length is lower than the smallest gap, the final number of scaffolds

would be equal to the starting number of contigs: ConSort is not able to create any connections.

- **ConSort vs. Newbler**.
  ConSort scaffolding is more accurate than Newbler one, in fact it is able to well estimate the gap length, and to resolve inconsistency. However, even this performance closely depends on the used mate-pair libraries: if they have a large standard deviation error, the gap evaluation is worst than the value achieved with more precise libraries, as could be seen in the comparison between 2kb and 10kb ones on the tomato genome test.

- **Repeat problem**.
  The new developed tool is subjected to error in the solution of very complicated repeats. Sometimes, with the mate-pairs information, it is able to determine some repeated regions and find the correct solution. Very often it is forced to stop the scaffold, and start a new one, avoiding the repeated sequence.

- **Code maintenance**
  ConSort could be easily maintained, adding, deleting or changing each module any time, without a complete code re-implementation, thanks to its modular organization. Each module performs a specific function, and it shares well defined data with the other ones. This is very useful both to update existing module with more efficient techniques, and to introduce new features. Besides, this allows an easy program updating, that could evolve with the sequencing technologies. Finally, it does not require too computational time or memory.

In conclusion, by now ConSort is strictly dependent on the performance, quantity and quality of the used data, contigs and mate-pair libraries. However, it is able to overcome some limits of the technologies on which it is based, as the inconsistency or the gaps, improving the assembly quality.

For the future, a lot of work has still to be done. First of all, the scaffolding module has to be expanded to manage repeats and ambiguous situations. Then, the gap filling function is to be implemented and tested. Finally, with the boost of the throughput produced by the sequencers and their more sophisticated techniques, it will soon require an integration of the new kind of data into the pipeline, to increase the achieved performance.

A File Format

In the appendix, there is a more detailed explanation about all the file format used for each module in the ConSort pipeline.

A.1 FASTA and multi-FASTA

A FASTA format is a text-based format for representing either nucleotide sequences or peptide sequences, in which base pairs or amino acids are represented using single-letter codes. A sequence in FASTA format begins with a single-line description, followed by lines of sequence data. The line description generally starts with a ">" (greater-than), and it contains the sequence name (contig in this case), sequence length and other possible information.

For example, Newbler uses this format:

```
>contig00001  length=2856   numreads=56
TTGACAAGTCAACCCATCACCAATCCCATCACTTCCAAACATATACCACAAGGAAATACA
AAAAGGTGTGAATGGAAAATGAAGAGTTGTGACTTCTATTTAAAGTTGTGACTTTTGTGA
....
>contig00002  length=1348   numreads=140
CGCATTCCATACACTTAACAAAAGTTTAGAAATCCACTTGCCTCATTTAGTCGAACAATC
AAAGTTGTGAATTTTATGAAAGGTGACAACCTTTCTGAAAGACTGTGATTTTTCCAAAGG
...
```

The first field after the ">" symbol (*contig00001* and *contig00002*in the example) is the name of the produced contig, followed by the length of the contig (*length=2856*), and the number of reads (*numreads=56*) that have been used to assembled the contig. The comment line is followed by the lines with the sequence (TTGAC...). After that, there is the second contig, with the same format, and so on.

A.2 GFF alignment

PASS stored the alignment in a GFF (General Feature Format) file format [1]. It is a tabular file, composed by nine fields:

```
 <seqname> <source> <feature> <start> <end> <score> <strand>
 <frame> [attributes] [comments]
```

It was developed to be a standard format in the feature information storing, that allows people to maintain all the data in a well-organized and easy to parse file.
PASS adapts this to its output, and it creates for each alignment a line with the nine fields, as it is shown in the example:

```
SL2.30ct18209   pass   match   4057   4102   44      -       .
ID=2339196:0:0;Name=1_1000_1017_R3 S1E45A18L45;P="46-1";
Q="27 23 16 9 12 24 21 12 10 8 17 23 18 17 23 26 22 16
17 23 24 13 12 12 14 22 22 23 21 24 26 19 12 17 23 25 19
 12 17 21 21 22 18 17 20 23 ";Note="M:2 -> 4/4 ?/A 10/10
 %?/T,G:0 ->";BS=1;Hits=1;
```

The meaning of its fields is:

1. *The name of the sequence*: in this case it is the name of the contig that contains the alignment.
2. *The source of the feature*: it is the name of the program that has produced or the source from which it has been extracted the information.
3. *The feature type name*: for the alignment its value is “match”, to identify the type of the information. It is constant for each line.
4. *start and end*: are respectively the initial and the final points of the alignment. The first value must be less than or equal the second one.
5. *score*: it specifies the number of bases that have been aligned.
6. *strand*: one of ‘+’, ‘-’ or ‘.’. The first and second symbols are used to define an alignment respectively in the positive and negative strand, while the last one is used when the information lack.

[1] http://www.sanger.ac.uk/resources/software/gff/spec.html

7. *frame*: one of '0', '1', '2' or '.'. It indicates the frame of the used region, in this case it is used the last symbol, because it is not avoidable.

8. *attribute*: in the last field all the comments and features that are not stored in the previous fields. PASS inserts the fields:

    - a line identifier value: that has to be unique for all the file;
    - the name of the reads;
    - trimming information (e.g. "S1E45A18L45"): S and E are respectively the first and last not trimmed base of the reads, A is its the average quality and L is its the final length;
    - quality base (e.g. Q="27 23 16 9..."): for each base it reports the quality of its sequencing;
    - matching (e.g. "Note="M:2 -> 4/4 ?/A 10/10 %?/T,G:0 ->"": the number of mismatches (M:2) and it defines the position, the wrong base and its quality;
    - number of alignment.

A.3 Pairing

PASS_pair uses a modified version of the GFF format used by PASS. The meaning of the fields are the same, but the lines are not independent. For example, for the UNIQUE_PAIR file, that stores the information about each paired-end, each couple of lines contains respectively one read of the pair. For a more detailed explanation about the program and its output format, see `http://pass.cribi.unipd.it/cgi-bin/pass.pl`.

A.4 Make Arcs

The "Make arcs" module produces the output in a tabular file, as seen in the example below:

```
{}
contig00011     contig00012     3       5       288     26      14
ok_counter="7 0 0 0 0 0 0 7";  ok_distance="282 0 0 0 0 0 0 294";
ko_counter="0 0 0 0 0 0 0 0"; ko_distance="0 0 0 0 0 0 0 0"; ";
```

The meaning of its fields is:

- contig00011: the name of the first contig that composes the arc;

- contig00011: the name of the second contig that composes the arc;
- 3 5: the two ends that look each other in the connection;
- 288: the gap length;
- 26: the standard deviation for the arc;
- 14: the number of mate-pairs that create the arcs;
- comment field: it contains the classification of the mate-pairs in the eighth counters, described in the Section 3.5.9. This is divided into mate-pairs that respect the library length constraint or not.

Bibliography

[1] F Sanger, S Nicklen, and A R Coulson. Dna sequencing with chain-terminating inhibitors. 1977. *Biotechnology*, 24:104–108, 1992.

[2] A M Maxam and W Gilbert. A new method for sequencing dna. 1977. *Biotechnology*, 24:99–103, 1992.

[3] D R Bentley. Whole-genome re-sequencing. *Curr Opin Genet Dev*, 16(6):545–552, Dec 2006.

[4] J D McPherson et al. (International Human Genome Mapping Consortium). A physical map of the human genome. *Nature*, 409(6822):934–941, Feb 2001.

[5] N. Whiteford, N. Haslam, G. Weber, A. Prugel-Bennett, J. W. Essex, P. L. Roach, M. Bradley, and C. Neylon. An analysis of the feasibility of short read sequencing. *Nucleic Acids Res.*, 33:e171, 2005.

[6] J Shendure and H Ji. Next-generation dna sequencing. *Nat Biotechnol*, 26(10):1135–1145, Oct 2008.

[7] Elaine R. Mardis. Next-generation DNA sequencing methods. *Annual review of genomics and human genetics*, 9(1):387–402, June 2008.

[8] E R Mardis. The impact of next-generation sequencing technology on genetics. *Trends Genet*, 24(3):133–141, Mar 2008.

[9] M J Fullwood, C L Wei, E T Liu, and Y Ruan. Next-generation dna sequencing of paired-end tags (pet) for transcriptome and genome analyses. *Genome Res*, 19(4):521–532, Apr 2009.

[10] C Burge and S Karlin. Prediction of complete gene structures in human genomic dna. *J Mol Biol*, 268(1):78–94, Apr 1997.

[11] M Stanke and S Waack. Gene prediction with a hidden markov model and a new intron submodel. *Bioinformatics*, 19 Suppl 2:215–225, Oct 2003.

[12] A Krogh. Two methods for improving performance of an hmm and their application for gene finding. *Proc Int Conf Intell Syst Mol Biol*, 5:179–186, 1997.

[13] G Parra, E Blanco, and R Guigó. Geneid in drosophila. *Genome Res*, 10(4):511–515, Apr 2000.

[14] E Birney and R Durbin. Using genewise in the drosophila annotation experiment. *Genome Res*, 10(4):547–548, Apr 2000.

[15] C. B. Burge and S. Karlin. Finding the genes in genomic DNA. *Curr. Opin. Struct. Biol.*, 8:346–354, Jun 1998.

[16] David Haussler. Computational genefinding. *Trends in Biotechnology*, 16(Supplement 1):12 – 15, 1998.

[17] M Pertea, X Lin, and S L Salzberg. Genesplicer: a new computational method for splice site prediction. *Nucleic Acids Res*, 29(5):1185–1190, Mar 2001.

[18] I.B. Rogozin and L. Milanesi. Analysis of donor splice signals in different organisms. *J. Mol. Evol.*, 45:50–59, 1997.

[19] J W Fickett and C S Tung. Assessment of protein coding measures. *Nucleic Acids Res*, 20(24):6441–6450, Dec 1992.

[20] G B Hutchinson and M R Hayden. The prediction of exons through an analysis of spliceable open reading frames. *Nucleic Acids Res*, 20(13):3453–3462, Jul 1992.

[21] S L Salzberg, A L Delcher, S Kasif, and O White. Microbial gene identification using interpolated markov models. *Nucleic Acids Res*, 26(2):544–548, Jan 1998.

[22] J. H. Do and D. K. Choi. Computational approaches to gene prediction. *J Microbiol*, 44(2):137–144, April 2006.

[23] R F Yeh, L P Lim, and C B Burge. Computational inference of homologous gene structures in the human genome. *Genome Res*, 11(5):803–816, May 2001.

[24] A Krogh. Using database matches with for hmmgene for automated gene detection in drosophila. *Genome Res*, 10(4):523–528, Apr 2000.

[25] J E Allen, M Pertea, and S L Salzberg. Computational gene prediction using multiple sources of evidence. *Genome Res*, 14(1):142–148, Jan 2004.

[26] J E Allen and S L Salzberg. JIGSAW: integration of multiple sources of evidence for gene prediction. *Bioinformatics*, 21(18):3596–3603, Sep 2005.

[27] D Campagna, A Albiero, A Bilardi, E Caniato, C Forcato, S Manavski, N Vitulo, and G Valle. Pass: a program to align short sequences. *Bioinformatics*, 25(7):967–968, Apr 2009.

[28] I Birol, S D Jackman, C B Nielsen, J Q Qian, R Varhol, G Stazyk, R D Morin, Y Zhao, M Hirst, J E Schein, D E Horsman, J M Connors, R D Gascoyne, M A Marra, and S J Jones. De novo transcriptome assembly with abyss. *Bioinformatics*, 25(21):2872–2877, Nov 2009.

[29] S Kumar and M L Blaxter. Comparing de novo assemblers for 454 transcriptome data. *BMC Genomics*, 11:571–571, 2010.

[30] M Pop. Genome assembly reborn: recent computational challenges. *Brief Bioinform*, 10(4):354–366, Jul 2009.

[31] E W Myers. Toward simplifying and accurately formulating fragment assembly. *J Comput Biol*, 2(2):275–290, 1995.

[32] M C Schatz, A L Delcher, and S L Salzberg. Assembly of large genomes using second-generation sequencing. *Genome Res*, 20(9):1165–1173, Sep 2010.

[33] R D Fleischmann, M D Adams, O White, R A Clayton, E F Kirkness, A R Kerlavage, C J Bult, J F Tomb, B A Dougherty, and J M Merrick. Whole-genome random sequencing and assembly of haemophilus influenzae rd. *Science*, 269(5223):496–512, Jul 1995.

[34] J C Venter, M D Adams, E W Myers, P W Li, and Mural *et al.* The sequence of the human genome. *Science*, 291(5507):1304–1351, Feb 2001.

[35] D A Wheeler, M Srinivasan, M Egholm, Y Shen, L Chen, and *et al.* The complete genome of an individual by massively parallel dna sequencing. *Nature*, 452(7189):872–876, Apr 2008.

[36] D R Bentley, S Balasubramanian, H P Swerdlow, G P Smith, J Milton, and *et al.* Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456(7218):53–59, Nov 2008.

[37] J Wang, W Wang, R Li, Y Li, G Tian, and *et al.* The diploid genome sequence of an asian individual. *Nature*, 456(7218):60–65, Nov 2008.

[38] J I Kim, Y S Ju, H Park, S Kim, and *et al.* A highly annotated whole-genome sequence of a korean individual. *Nature*, 460(7258):1011–1015, Aug 2009.

[39] R Velasco, A Zharkikh, M Troggio, D A Cartwright, and A *et al.* Cestaro. A high quality draft consensus sequence of the genome of a heterozygous grapevine variety. *PLoS One*, 2(12), 2007.

[40] S Huang, R Li, Z Zhang, L Li, and *et al.* The genome of the cucumber, cucumis sativus l. *Nat Genet*, 41(12):1275–1281, Dec 2009.

[41] R Li, W Fan, G Tian, H Zhu, and *et al.* The sequence and de novo assembly of the giant panda genome. *Nature*, 463(7279):311–317, Jan 2010.

[42] V Shulaev, D J Sargent, R N Crowhurst, T C Mockler, O Folkerts, and *et al.* The genome of woodland strawberry (fragaria vesca). *Nat Genet*, Dec 2010.

[43] X Argout, J Salse, J M Aury, M J Guiltinan, G Droc, and *et al.* The genome of theobroma cacao. *Nat Genet*, Dec 2010.

[44] R L Warren, G G Sutton, S J Jones, and R A Holt. Assembling millions of short dna sequences using ssake. *Bioinformatics*, 23(4):500–501, Feb 2007.

[45] W R Jeck, J A Reinhardt, D A Baltrus, M T Hickenbotham, V Magrini, E R Mardis, J L Dangl, and C D Jones. Extending assembly of short dna sequences to handle error. *Bioinformatics*, 23(21):2942–2944, Nov 2007.

[46] J C Dohm, C Lottaz, T Borodina, and H Himmelbauer. Sharcgs, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Res*, 17(11):1697–1706, Nov 2007.

[47] E W Myers, G G Sutton, A L Delcher, I M Dew, D P Fasulo, and *et al.* A whole-genome assembly of drosophila. *Science*, 287(5461):2196–2204, Mar 2000.

[48] S Batzoglou, D B Jaffe, K Stanley, J Butler, S Gnerre, E Mauceli, B Berger, J P Mesirov, and E S Lander. Arachne: a whole-genome shotgun assembler. *Genome Res*, 12(1):177–189, Jan 2002.

[49] D B Jaffe, J Butler, S Gnerre, E Mauceli, K Lindblad-Toh, J P Mesirov, M C Zody, and E S Lander. Whole-genome sequence assembly for mammalian genomes: Arachne 2. *Genome Res*, 13(1):91–96, Jan 2003.

[50] X Huang and S P Yang. Generating a genome assembly with pcap. *Curr Protoc Bioinformatics*, Chapter 11, Oct 2005.

[51] J R Miller, S Koren, and G Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, Jun 2010.

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

[53] S Koren, J R Miller, B P Walenz, and G Sutton. An algorithm for automated closure during assembly. *BMC Bioinformatics*, 11:457–457, 2010.

[54] D Hernandez, P Francois, L Farinelli, M Osteras, and J Schrenzel. De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. *Genome Res*, 18(5):802–809, May 2008.

[55] P A Pevzner, P A Pevzner, H Tang, and G Tesler. De novo repeat classification and fragment assembly. *Genome Res*, 14(9):1786–1796, Sep 2004.

[56] P A Pevzner, H Tang, and M S Waterman. An eulerian path approach to dna fragment assembly. *Proc Natl Acad Sci U S A*, 98(17):9748–9753, Aug 2001.

[57] P A Pevzner and H Tang. Fragment assembly with double-barreled data. *Bioinformatics*, 17 Suppl 1:225–233, 2001.

[58] M J Chaisson and P A Pevzner. Short read fragment assembly of bacterial genomes. *Genome Res*, 18(2):324–330, Feb 2008.

[59] M J Chaisson, D Brinza, and P A Pevzner. De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Res*, 19(2):336–346, Feb 2009.

[60] D R Zerbino and E Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Res*, 18(5):821–829, May 2008.

[61] J T Simpson, K Wong, S D Jackman, J E Schein, S J Jones, and I Birol. Abyss: a parallel assembler for short read sequence data. *Genome Res*, 19(6):1117–1123, Jun 2009.

[62] J Butler, I MacCallum, M Kleber, I A Shlyakhter, M K Belmonte, E S Lander, C Nusbaum, and D B Jaffe. Allpaths: de novo assembly of whole-genome shotgun microreads. *Genome Res*, 18(5):810–820, May 2008.

[63] I Maccallum, D Przybylski, S Gnerre, J Burton, I Shlyakhter, A Gnirke, J Malek, K McKernan, S Ranade, T P Shea, L Williams, S Young, C Nusbaum, and D B Jaffe. Allpaths 2: small genomes assembled accurately and with high continuity from short paired reads. *Genome Biol*, 10(10), 2009.

[64] R Li, H Zhu, J Ruan, W Qian, X Fang, Z Shi, Y Li, S Li, G Shan, K Kristiansen, S Li, H Yang, J Wang, and J Wang. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res*, 20(2):265–272, Feb 2010.

[65] M Pop, D S Kosack, and S L Salzberg. Hierarchical scaffolding with bambus. *Genome Res*, 14(1):149–159, Jan 2004.

[66] A Dayarian, T P Michael, and A M Sengupta. Sopra: Scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinformatics*, 11:345–345, 2010.

[67] W Huang and G Marth. Eagleview: a genome assembly viewer for next-generation sequencing technologies. *Genome Res*, 18(9):1538–1543, Sep 2008.

[68] D Gordon, C Abajian, and P Green. Consed: a graphical tool for sequence finishing. *Genome Res*, 8(3):195–202, Mar 1998.

[69] J Nijkamp, W Winterbach, M van den Broek, J M Daran, M Reinders, and D de Ridder. Integrating genome assemblies with maia. *Bioinformatics*, 26(18):433–439, Sep 2010.

[70] D. Campagna, C. Romualdi, N. Vitulo, M. Del Favero, M. Lexa, N. Cannata, and G. Valle. RAP: a new computer program for de novo identification of repeated sequences in whole genomes. *Bioinformatics*, 21:582–588, Mar 2005.