University of Nantes
Faculty of Science and Technology
Master 2 Biology and Health. Speciality: Bioinformatics

Development of a short read alignment software using NCBI/BLAST

Aurélien GUY-DUCHÉ supervisor Dr. Pierre LINDENBAUM

Nantes, September 5, 2015

Abstract

Basic Local Alignment Search Tool (BLAST) has been widely used since 1990 to align DNA sequences. Since the Next-Generation Sequencing (NGS) revolution, sequencing data has been mapped with faster software like Burrows-Wheeler Aligner (BWA). In this project, we propose to see if BLAST can have a better accuracy and therefore can be used, in some applications, to complement existing aligners.

We created two programs, Fastq2Fasta and Blast2Bam, to enable the use of NGS data by BLAST. Fastq2Fasta transforms the NGS format fastQ into a fasta file so that the sequences can be loaded into BLAST. Blast2Bam extracts the alignment data from BLAST and outputs a file in the standard NGS format: Sequence Alignment/Map format (SAM). Blast2Bam is able to produce paired records if paired-end sequencing was performed. The output has passed validation tests.

We then evaluated on diverse parameters the differences between the results of BLAST–Blast2Bam and BWA. As expected, BWA proved to be much faster. However, BLAST–Blast2Bam demonstrated a better accuracy by displaying an interesting increase in the coverage of the sequencing.

In conclusion, we showed that BLAST–Blast2Bam may be useful in some NGS applications to confirm and/or to improve other aligners' results.

Keywords: BLAST, BWA, NGS, SAM.

Résumé

Basic Local Alignment Search Tool (BLAST) est couramment utilisé depuis 1990 pour aligner des séquences d'ADN. Depuis la révolution provoquée par l'arrivée du séquençage haut débit (NGS), les alignements de séquences sont faits par des logiciels plus rapides tels que Burrows-Wheeler Aligner (BWA). Dans ce projet, nous nous proposons de déterminer si BLAST peut être plus précis et donc s'il pourrait être utile dans certaines applications pour améliorer les résultats obtenus avec d'autres logiciels d'alignement.

Pour permettre l'utilisation de données NGS par BLAST, nous avons créé deux programmes : Fastq2Fasta et Blast2Bam. Fastq2Fasta transforme le format de fichier fastQ en fasta, permettant ainsi à BLAST de lire les séquences à aligner. À partir des résultats de BLAST, Blast2Bam produit un fichier au format NGS standard : Sequence Alignment/Map format (SAM). Dans le cas où le séquençage aurait été réalisé de façon *paired-end*, Blast2Bam conserve l'appariement des séquences afin d'améliorer l'alignement. Les logiciels ont passé les étapes de validation.

Nous avons ensuite évalué les différences d'exécution qui existent entre BLAST–Blast2Bam et BWA. Comme prévu, BWA s'est révélé beaucoup plus rapide que BLAST–Blast2Bam. Néanmoins, ce dernier a montré qu'il pouvait être plus précis en permettant l'obtention d'une couverture de séquençage plus importante.

En conclusion, nous avons montré que BLAST–Blast2Bam peut être utile pour confirmer et/ou améliorer les résultats d'autres logiciels d'alignement de séquences.

Mots-clés : BLAST, BWA, NGS, SAM.

Acknowledgments

Firstly, I would like to thank Dr. Richard Redon for welcoming me in his lab. It has allowed me to learn a lot about NGS data analysis along with the pleasure of working with a great team.

I wish to express my sincere gratitude to my supervisor, Dr. Pierre Lindenbaum for his guidance, his profound expertise, his constant availability and his great advices.

I would like to thank Dr. Solena Le Scouarnec, Anne-Sophie Denommé-Pichon and Cyrille Chopelet for the time they spent proofreading this report.

Finally, I would also like to thank all the other team members for the very enjoyable working environment they created.

Contents

List of Figures

Acronyms

API Application Programming Interface. 11

BLAST Basic Local Alignment Search Tool. 1, 2, 4–8, 10–15, 17–24

BWA Burrows-Wheeler Aligner. 4, 8, 14, 17–24

CIGAR Compact Idiosyncratic Gapped Alignment Report. 8, 14, 15, 18

DNA Deoxyribonucleic Acid. 1, 3, 5, 6, 17

DOM Document Object Model. 11

DTD Document Type Definition. 11

HIV Human Immunodeficiency Virus. 17, 18, 20

HSP High-scoring Segment Pair. 2, 11–15, 20

NCBI National Center for Biotechnology Information. 1, 5, 6, 11, 17, 24

NGS Next-Generation Sequencing. 1–5, 16, 17, 20

PCR Polymerase Chain Reaction. 3

SAM Sequence Alignment/Map format. 4, 7, 8, 10–19, 24

SAX Simple API for XML. 11

StAX Streaming API for XML. 11

XML Extensible Markup Language. 4, 6–8, 10–12

XSLT Extensible Stylesheet Language Transformations. 11

1 | Introduction

1.1 | Sequencing

Since 1869 and the discovery of "nuclein" by F. Miescher, we have come to understand that Deoxyribonucleic Acid (DNA) is the basis of every living organism on Earth. It is a code composed of four letters (nucleotides). In order to read this code and maybe be able to decipher it, new technologies had to be invented.

In the early 1970s, R. Wu, R. Padmanabhan and later F. Sanger created a process called sequencing to read the nucleotides in a DNA molecule. In 1977, F. Sanger and his team performed the first complete genome sequencing (bacteriophage φX174). In 2003, the International Human Genome Sequencing Consortium announced that the Human genome had been essentially sequenced.

In the next years appeared new technologies called Next-Generation Sequencing (NGS) that revolutionized genome research. These technologies, especially Illumina®, will be discussed in more detail in Section 1.3. They have allowed the researchers to obtain sequences much faster than what was possible before and at a fraction of the price. This has contributed to the more widespread use of DNA sequencing. Nowadays, medical practionners use it more and more commonly to clarify the diagnostic of genetic pathologies and/or to predict the risks of developing one.

Thanks to NGS, we are now able to "read the Human code" relatively easily; deciphering it however is still a work in progress. One of the processes used to understand it is to compare the genomes of different individuals. This involves aligning the sequences and identifying mutations that would explain the differences in the observed phenotypes. The alignment could theoretically be done by hand but as the Human genome is composed of around three billion base pairs, it would be a rather tedious task. And so, programs have been developed to perform this task.

1.2 | Basic Local Alignment Search Tool (BLAST)

BLAST is the most widely known sequence alignment software. It has been developed in 1990 by S. Altschul at the National Center for Biotechnology Information (NCBI) [1]. Two formats are available, as a web tool on the NCBI website and as a command-line tool in BLAST+ package. Basic Local Alignment Search Tool (BLAST), as its name suggests, performs alignments between two biological sequences (nucleotide or protein). The comparison can only be done on primary structures (raw sequences). There is always at least two sequences: a query and a reference. It can work with one or several query sequences and one or several references.

BLAST is a local aligner, meaning that it searches for sequence homology by matching the reference with small parts of the query instead of the whole sequence. Its process can be decomposed into six main steps:

1. The query sequence is cut into overlapping subsequences called "words" or "seeds". The cut is dependent on the length of the words generated (`word_size`). Example: if the query is `AATGCATCGC` and the `word_size` is 8, the resulting subsequences will be `AATGCATC`, `ATGCATCG` and `TGCATCGC`. All the words generated are recorded in a dictionary.

2. A neighborhood of potential words is created around each of the subsequences. This neighborhood is constituted to get around the fact that there may be mutations, and so mismatches, between the subsequence and the reference. BLAST generates a list of all the words possible considering the `word_size`. In our example, this would mean 4^8 words. Most of these words are irrelevant as the number of mutations needed to generate them would be too great. To filter them, BLAST uses a matrix of similarity in which a determined score is given for matches and mismatches. Each word in the list is compared with the subsequence generated in step 1 and, using the matrix, an homology score is generated for each. Only the words with a score higher than a given threshold are kept.

3. The words are tested for matches against the reference. BLAST tries to find a perfect match between each remaining word of the neighborhood and the reference.

4. The elongation of the seed: When a match is found, the word, now called a seed, is elongated in both direction. Each elongation operation modifies the homology score based on the matrix of similarity. The process continues until the score is lower than the given threshold. The alignment constituted is called a High-scoring Segment Pair (HSP).

5. E-value: BLAST then calculates the relevancy of each High-scoring Segment Pair (HSP) (E-value). If, given a threshold, the HSP is not found statistically significant, it will be discarded.

6. Output of the results. All the passing alignments are printed in a report.

1.3 | Next Generation Sequencing (NGS)

As we have seen in Section 1.1, around the mid-2000s, the second generation sequencers, commonly called NGS, have considerably increased the throughput compared with the first generation ones (Sanger sequencers). They allow the researchers to conceive large-scale projects and consider new applications. NGS comprises different technologies that share the same goal: to obtain a genome sequence fast, with as few errors as possible and for a low cost. These technologies are:

- Pyrosequencing (454 Life Sciences, Roche),
- Polymerase based sequence by synthesis (Illumina®),
- Ligation based sequencing (SOLiD) (Applied Biosystems, Thermo Fisher).

They all have their advantages and disadvantages in term of speed, accuracy and cost. Recently, new sequencers coined as "third generation sequencers" have appeared: PacBio® from Pacific Bioscience and MinION from Oxford Nanopore Technologies.

As the experiments proposed in this report are based on data acquired with Illumina® technology, we will now explain its basics.

Illumina®'s sequence by synthesis is currently the most used NGS technology worldwide. It is based upon the reading of fluorescent markers as the nucleotides on which they are fixed are used by the polymerase to synthesize a copy of the template. The first sequencer to use this technology was the Genome Analyzer produced by Solexa in 2006. Illumina®, now owner of the technology, commercializes it into four series of sequencers: MiSeq, NextSeq, HiSeq and HiSeq X.

The sequencing is composed of four main steps [2]:

- Library preparation: The DNA is cut into small fragments (200–500 base pairs) and adapters are fixed on their extremities.
- Clustering: Each fragment is hybridized onto the flow cell and amplified to form a cluster by a process called bridge amplification. To put it simply, it is a Polymerase Chain Reaction (PCR) that uses the special coating of the flow cell to make copies of the fragment and keep them attached to the surface. This way, the multiple copies of each fragment are located on a particular region of the cell. This is done in order to get a clear signal during the sequencing.
- Sequencing: First, a probe is fixed on the free extremity of each fragment. Then, fluorescently labeled nucleotides are added with the polymerase. Each nucleotide is assigned a specific wavelength. Each cycle consists in the extension, one nucleotide at a time, of the strand based on the probe. The process is performed as follows:
  - The polymerase puts the corresponding nucleotide at the end of the sequence.
  - The reaction is stopped due to the label.
  - The fluorescence is detected by a camera.
  - The label is cut. A new cycle can begin.

  Depending on the sequencer, there can be a maximum of 150 to 300 cycles, thus producing reads with a maximum length of 150 to 300 bases. The reads are then written into a fastQ file. We will talk more about this file format in Section 2.1.
- Data analysis: It consists in, firstly, the alignment of the reads to a reference and then the identification of the differences and their biological meaning.

The sequencing can be done in two different modes: single-end and paired-end. In single-end, each DNA fragment is read on one end. In paired-end, it is read on both ends, thus increasing the probability that the fragment will be well aligned on the reference. In that case, the clustering and sequencing steps are repeated for the complementary strand. A representation of the position of the reads in paired-end sequencing is shown on figure 1.1. In common usage, the read associated with the current read is called its "mate".

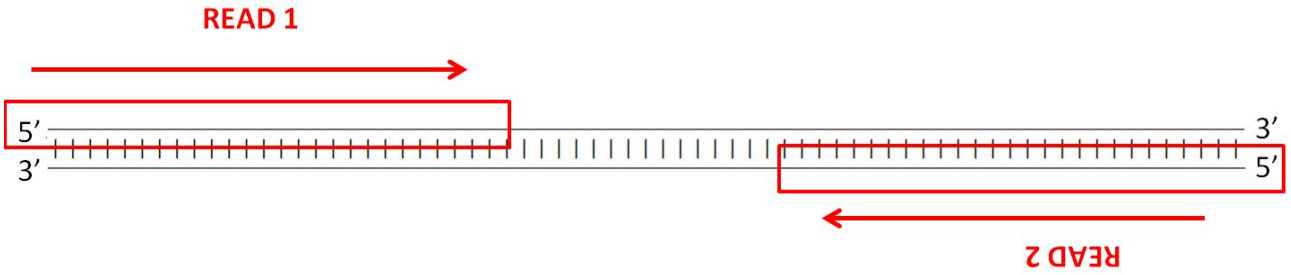


Figure 1.1: Representation of paired-end sequencing.

Illumina® issued a video explaining more precisely how the technology works: `https://www.youtube.com/watch?v=womKfikWlxM`.

Concerning the reads alignments, there are two possibilities depending on whether there is a reference sequence or not. In the first case, it is called mapping and in the last *de novo* assembly. In our project, we will focus only on mapping.

NGS technologies are able to generate millions of short-reads whereas Sanger sequencers produces a few long reads. In 2005, when NGS emerged, the need for mappers capable of handling this sort of data emerged also. In 2009, the Burrows-Wheeler Aligner (BWA) was created. It is currently the most used NGS mapper. It is able to fastly map millions of reads on genomes several billion base pairs long.

1.4 | The project

The goal of our project is to create a program able to use BLAST to map short-reads. As we have just seen, very efficient programs like BWA have already been developed to work with NGS data. In that case, why bother writing a software to adapt an old algorithm when good solutions already exist? Because BWA and the other new mappers are not perfect: their increased speed is often obtained at the expense of the accuracy. We know that BLAST is a lot slower than those mappers but it may be more sensitive and produce better alignments. Indeed, Heng Li, the creator of BWA, SAMtools and other highly used software, commented on Biostars (thread 145596):

> "Blast is in theory more sensitive to highly divergent hits IF you use the right parameters."

Apart from creating the software itself, this is what we propose to demonstrate with this project.

Due to the sluggishness of BLAST compared with newer mappers, the purpose is not to replace them but rather to complement them. For example, it could be useful to realign some small regions where we are not sure about the previous alignment. It could also be useful for the lab in case the quality of the sequencing data to analyze is poor; BWA for instance is not very performant when confronted with a lot of mismatches. Finally, it could be used on small genomes like viruses or bacteria.

To our knowledge, there are only two other tools able to read BLAST results and output a file that can be used for downstream analyses.

- blast2sam.pl: In the sources of SAMtools, a Perl script called blast2sam.pl can be found. This script translates BLAST output into a SAM file. There is no pairing, the sequencing quality is lost and it is reported not to work properly. Heng Li, the author, said about it on SEQanswers.com (thread 3759):

  > "BLAST support will be dropped unless someone want to maintain it. [...] I just thought this script may be useful to someone occasionally, but it is now causing more troubles than good. Sorry."

- jvarkit/blast2sam: Some time ago, Pierre Lindenbaum developed a tool written in Java called blast2sam for his jvarkit set of NGS tools. The program extracts the alignment data from BLAST XML output, analyzes it and then outputs a SAM file. It is able to work efficiently with paired-end data but like blast2sam.pl, it lacks the capability of retrieving the sequencing quality. Furthermore, it is slow and a new implementation would be beneficial.

As these tools did not give satisfactory results, we decided to go on with this project and create a new program.

2 | Program expectations

The goal of this project is to make a short read alignment software using NCBI/BLAST. Before writing it, we need to understand precisely what is expected of it. This is what we propose to do in this chapter.

2.1 | Short reads

NGS sequencers usually output their results in fastQ file(s). FastQs are text files containing records for each read generated by the sequencer.

Figure 2.1: A record in a fastQ file is composed of four lines:

- Read identification preceded by '@',
- DNA sequence,
- Separator (+),
- Read sequencing quality.

```
@sequenceID.1
ATATCGAT
+
!8ACC@FG
@sequenceID.2
(...)
```

The read quality is composed of symbols representing the sequencing quality of each base. This string and the DNA sequence should have the same length.

As we have seen in Section 1.3, sequencing can be done in a single-end or paired-end fashion. If the data is single-end, there should only be one fastQ file. If the data is paired-end, there are two possibilities:

- Two fastQ files are generated, one for each strand of the sequence;
- The sequences of the two strands are interleaved in one fastQ file.

An example of paired-end fastQ files is represented in figure 2.2.

The program is expected to use fastQ files as an input and analyze them properly according to the type of sequencing used (single-end, paired-end or paired-end with interleaved data).

FastQ1

```
@sequenceID.1
ATATCGAT
+
!8ACC@FG
@sequenceID.2
(...)
```

FastQ2

```
@sequenceID.1
AGATAGAG
+
!8A@CGGE
@sequenceID.2
(...)
```

Interleaved

```
@sequenceID.1
ATATCGAT
+
!8ACC@FG
@sequenceID.1
AGATAGAG
+
!8A@CGGE
@sequenceID.2
(...)
@sequenceID.2
(...)
```

Figure 2.2: Paired-end fastQ.

2.2 | BLAST

The project states that the alignment must be done with NCBI/BLAST. Since we want the results to be directly available as an input of the program, we have decided to use the local version of BLAST (BLAST+). BLAST+ is composed of different pieces of software, each able to align a specific type of sequence. Our goal being the alignment of short reads sequences against a DNA reference, we chose the tool blastn (Nucleotide BLAST).

2.2.1 | BLAST input

BLAST cannot process fastQ files, the input sequences must be in fasta format. Fastas are text files containing records for each sequence.

Figure 2.3: A record in a fasta file is composed of two lines:

- Sequence identification preceded by '>',
- DNA sequence.

```
>sequenceID.1
ATATCGAT
>sequenceID.2
(...)
```

The software is expected to convert and merge fastQ files into a fasta file for BLAST input.

2.2.2 | BLAST output

Since the last version of BLAST+ (2.2.31), BLAST has 14 different outputs. There is also a 15th undocumented one we will discuss in the general conclusion (page 24). In order to get the maximum of information about the alignments, Extensible Markup Language (XML) output was chosen (`outfmt 5`).

Figure 2.4 shows an example of a partial BLAST XML output.

The software is expected to extract the data from a BLAST XML output file.

```
(...)
<Iteration>
  <Iteration_iter-num>3</Iteration_iter-num>
  <Iteration_query-ID>Query_3</Iteration_query-ID>
  <Iteration_query-def>ERR656485.2</Iteration_query-def>
  <Iteration_query-len>300</Iteration_query-len>
<Iteration_hits>
<Hit>
  <Hit_num>1</Hit_num>
  <Hit_id>gnl|BL_ORD_ID|0</Hit_id>
  <Hit_def>gi|9629357|ref|NC_001802.1| Human immunodeficiency virus 1, complete
      genome</Hit_def>
  <Hit_accession>0</Hit_accession>
  <Hit_len>9181</Hit_len>
  <Hit_hsps>
    <Hsp>
      <Hsp_num>1</Hsp_num>
      <Hsp_bit-score>148.852</Hsp_bit-score>
      <Hsp_score>80</Hsp_score>
      <Hsp_evalue>4.07002e-39</Hsp_evalue>
      <Hsp_query-from>2</Hsp_query-from>
      <Hsp_query-to>120</Hsp_query-to>
      <Hsp_hit-from>833</Hsp_hit-from>
      <Hsp_hit-to>715</Hsp_hit-to>
      <Hsp_query-frame>1</Hsp_query-frame>
      <Hsp_hit-frame>-1</Hsp_hit-frame>
      <Hsp_identity>106</Hsp_identity>
      <Hsp_positive>106</Hsp_positive>
      <Hsp_gaps>0</Hsp_gaps>
      <Hsp_align-len>119</Hsp_align-len>
      <Hsp_qseq>TGGGCTAAAGGCCTTTTCCTCTATTACTTTTACCCATGCATTTAAAGTTCTAGGTGACATGGCCTGGTG
          TACCATTTGCCCTTGGAGATTTTGCACTATAGGATAATTTTGACTGACCT</Hsp_qseq>
      <Hsp_hseq>TGGGCTGAAAGCCTTCTCTTCTACTACTTTTACCCATGCATTTAAAGTTCTAGGTGATATGGCCTGATG
          TACCATTTGCCCCTGGATGTTCTGCACTATAGGGTAATTTTGGCTGACCT</Hsp_hseq>
      <Hsp_midline>|||||| || ||||| || |||| ||||||||||||||||||||||||||||||||| ||||||||
           ||||||||||||||| ||||  || |||||||||| |||||||| |||||||
      </Hsp_midline>
    </Hsp>
(...)
```

Figure 2.4: BLAST XML output.

2.3 | Output

Popular downstream analysis software such as SAMtools, GATK or IGV use the standard Sequence Alignment/Map format (SAM) or its binary counterpart (BAM). It is defined in the SAM format specification [3]. SAM is a text-based file containing two parts described below.

- Header section:
  - @HD: Header line: SAM format version and sorting order of alignments;
  - @SQ: Reference sequence dictionary: reference sequence name and its length;
  - @RG: Read group: read group identifier and sample name;
  - @PG: Program: aligner identifier, name, version and command-line used.

- Alignment section, composed of at least 11 elements:
    1. QNAME: Read name;
    2. FLAG: Bitwise flag used to categorize the read according to its alignment on the reference;
    3. RNAME: Name of the reference on which the read is mapped;
    4. POS: Position on the reference at which the read is mapped;
    5. MAPQ: Mapping quality of the read;
    6. CIGAR: Compact Idiosyncratic Gapped Alignment Report (CIGAR) is string representation of the alignment. '=' represents a match, 'X' a mismatch, 'I' an insertion, 'D' a deletion, 'N' a long deletion, 'S' a soft-clipped base, 'H' a hard-clipped one. 'M' can be used to represent either a match or a mismatch. A base is soft-clipped if it does not take part in the alignment but the base is still shown in the sequence. Hard-clipped bases differ by the fact that they are not shown in the sequence. In both cases, they can only be encountered at the ends of the reads. In the CIGAR string, the number before the symbol represents the number of times the symbol is encountered before a different one occurs; Example: `1S3=2X1I` means that the alignment begins with a soft-clipped base, followed by three matches, two mismatches and an insertion.
    7. RNEXT: Name of the reference on which the read's mate is mapped;
    8. PNEXT: Position on the reference at which the read's mate is mapped;
    9. TLEN: Distance on the reference between the 5' end of the read and the 5' end of its mate;
    10. SEQ: Sequence of the read;
    11. QUAL: Sequencing quality of the read;
    12. Metadata: Optional supplementary information on the read.

Figure 4.3 shows an example of a SAM file generated by BWA.

The software is expected to produce a SAM file compatible with downstream softwares.

2.4 | Implementation language

BLAST is known to be slower than more recent aligners. To avoid slowing down further the mapping, an implementation in a compiled language would be preferred. The program is expected to be able to read gzipped formatted files (fastQ files are usually compressed) and to parse an XML file. Very well known C libraries can be used to that goal: zlib and libxml2. To maximize speed, the program will be implemented in C.

Based on these expectations, we devised a general outline of the program to implement (fig 2.5).

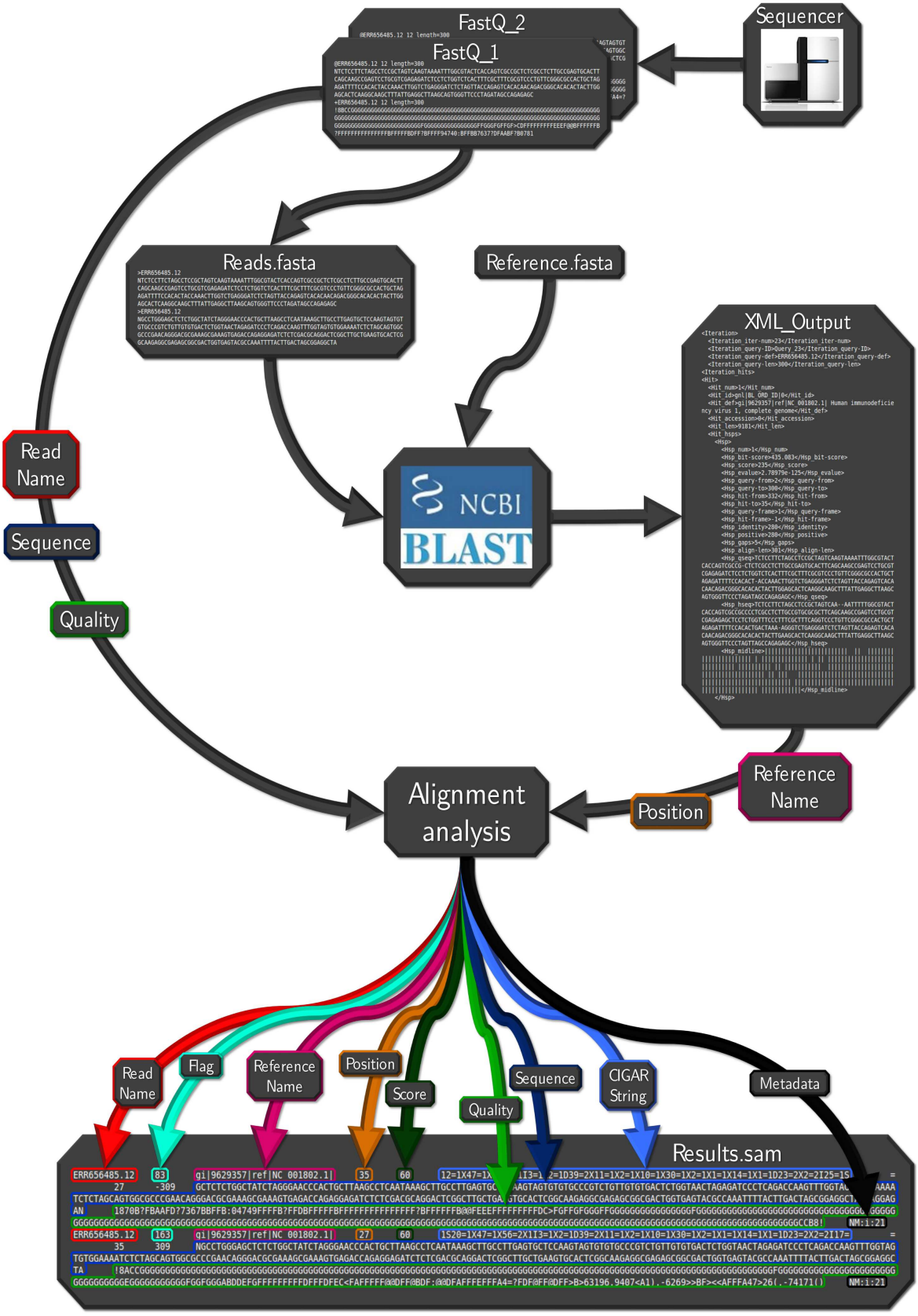


Figure 2.5: General outline of the program.

3 | Implementation

To conform to the expectations, the program was built in several parts. We will now see how it was done.

All the sources, tests and manual can be found on GitHub: `https://github.com/guyduche/Blast2Bam`.

3.1 | Fastq2Fasta

As we have seen in Subsection 2.2.1, BLAST cannot process fastQ files. FastQs must be transformed into fasta files.

This can be done with bash commands but we thought it would be more user-friendly to integrate it in a simple tool. That is why we created Fastq2Fasta. Fastq2Fasta reads the fastQ file record by record (four lines at a time). It then extracts the first two lines of each record (read name and sequence) and writes it in the fasta file. In the process, the '@' at the beginning of the first line is replaced with a '>' to match fasta format.

3.2 | Blast2Bam

Blast2Bam is the name we gave to the program we created to go from a BLAST XML output to a fully compatible SAM file. As we have seen in Section 2.3, a SAM file is composed of two parts: the header section and the alignment section. We will now see how Blast2Bam is able to create them.

3.2.1 | Header section

As we have seen previously (Section 2.3), a SAM header is composed of mainly four types of line: *@HD, @SQ, @RG and @PG.*

- *@HD* is created automatically by SAMtools when the SAM file is transformed into BAM (binary version of SAM). Thus, there is no need for Blast2Bam to create this line.
- *@SQ* contains the reference name and its length. There must be an *@SQ* line for each reference on which the reads are tested for alignment. As we can see in figure 2.4, the reference name and its length are available respectively under tags `Hit_def` and `Hit_len` of BLAST XML output. However, those tags will only exist if the reads are mapped on the reference. A reference must have an *@SQ* line even if no read is mapped on it. Therefore, the information must come from an external source. Blast2Bam extracts the names and calculates the lengths of the sequences from the reference file. It then outputs a formatted *@SQ* line for each sequence.

- *@RG*: The read group line is optional and must come from user input. This can be passed to Blast2Bam as a string via the `-R` option. This string must follow a specific syntax in order to be accepted: '`@RG\tID:foo\t...`'.
- *@PG* contains the program identification, name, version and the command-line used (CL). The CL tag is obtained by perusing through the arguments of the *main()* function of Blast2Bam.

We will now see how the alignment section was done.

3.2.2 | XML parsing

The first challenge was to parse BLAST XML output. Parsing is the process of reading data stored in a file (in our case an XML file) and storing it into structures we can easily manipulate. To help us, we chose to use libxml2, a software library for XML parsing. Three ways of parsing are available in this Application Programming Interface (API):

- Document Object Model (DOM): Tree-based parsing. The whole file is loaded in memory as a tree.
- Simple API for XML (SAX): Event-based push parsing. The parsing is done in streaming using callback functions.
- Streaming API for XML (StAX): Event-based pull parsing. Simpler event-based parsing.

The size of BLAST XML output files can be huge so DOM approach cannot be used in this application. Besides, event-based parsers can extract data from 'stdin', which means that they would enable us to integrate BLAST and Blast2Bam in a pipeline. Two event-based parsers, SAX and StAX (xmlreader), are available in libxml2. The pull parsing option is simpler and allows more control over the parsing. Daniel Veillard, the conceptor of libxml2, wrote on xmlreader webpage [4]:

> "In a nutshell the XmlTextReader API provides a simpler, more standard and more extensible interface to handle large documents than the existing SAX version."

Consequently, we chose to use the xmlreader (StAX) approach to parse the XML BLAST output.

In order to avoid writing redundant pieces of code, we created an XSLT transformation file. Extensible Stylesheet Language Transformations (XSLT) is an XML specification used to transform an XML file into another type of document. In our case, we use it to write a text file that will contain the C source code necessary to parse the BLAST XML output.

First, we created an XML document that contains all the different tags that can be found in BLAST output. This document is based on the BLAST output Document Type Definition (DTD) issued by the NCBI. The XSLT code looks into the XML document and, for each tag, writes the C code that creates the structures and substructures corresponding to the tags and then the code that will be used to extract the data from the BLAST output and put it into the structures. This manner of implementation will allow the parser to be more adaptable in case the NCBI issued an update of the XML output.

For each iteration of BLAST, the parser creates data structures adapted to the information contained in the XML file (number of hits, number of HSP) and records it for further use by the rest of the program.

3.2.3 | FastQ parsing

If we compare figure 2.4 and figure 4.3, we can see that BLAST XML output is lacking some information needed to write the SAM file.

- BLAST cannot process fastQ files and so its output does not display the sequencing quality of the reads.
- The read's sequence needed in column 10 of the SAM file should be the whole sequence of the read. Sequences in BLAST output are modified. They are hard-clipped (only the part of the sequence needed for the alignment is displayed) and '-' are added in the query sequence to represent deletions.

This information can be extracted from the original fastQ(s). Consequently, Blast2Bam parses the fastQ and writes its contents into the SAM file. This parsing is done essentially the same way it was done by Fastq2Fasta (Section 3.1), apart from the fact that, in this implementation, the fourth line (sequencing quality) is kept and there is no fasta file writing.

3.2.4 | Read pairing

BLAST does not recognize paired-end data. It maps a single sequence with the reference and outputs the alignment results. As we can see in Figure 4.3, the read and its mate should be paired in SAM output. It is essential to obtain the FLAG, RNEXT, PNEXT and TLEN columns.

If the sequencing data is single-end, no pairing is possible. In that case, FLAG does not display any pairing component, RNEXT shows the indefinite character '*' and PNEXT and TLEN are nullified.

The pairing must be done at several levels:

- BLAST should process the sequence of the mate just after its corresponding read. We have seen in Section 2.1 that paired-end data can come in two flavors: two fastQs or a single interleaved one. In case of interleaved data, the read and its mate will already be processed one after the other. In case of two fastQs, Fastq2Fasta is instructed to read alternatively from the first and second file, thus writing an interleaved fasta file.
- The BLAST output is parsed two iterations at a time (BLAST does one iteration for each read).
- Blast2Bam parses the fastQ files or interleaved fastQ similarly to Fastq2Fasta. It keeps reads' records two at a time to match BLAST iterations.
- After XML and fastQ parsing, Blast2Bam pairs the data by recording it into a series of structures.
  - *IterationSam*: global structure containing all the information concerning the alignments of the two reads. It contains an indefinite number of *SamHits*. There is always only one *IterationSam* at a time as it is deleted and reconstituted for each double iteration in BLAST output.
  - *SamHit*: structure containing the alignments of the two reads on a single reference. It contains an indefinite number of *RecordSam*. There is as much *SamHits* in *IterationSam* as there are references on which the two reads are tested for mapping.
  - *RecordSam*: structure containing a paired alignment record. In BLAST output, there is an HSP for each alignment found. Blast2Bam creates a paired record for each combination of HSP possible between the two reads. For example, if three possible alignments are found for a read and two for its mate, six records will be created. As the two reads of a pair should not be mapped to different references, only the HSP concerning the same reference are considered for pairing. A *RecordSam* structure contains two *SamOutput* structures, one for each read.
  - *SamOutput*: structure containing the read informations from the fastQ, the name of the reference on which it is mapped and the corresponding HSP from BLAST output.

- As we will see in the next subsection, the pairing process continues with the filtering of the records and then with the creation of the FLAG, RNEXT, PNEXT and TLEN columns of the SAM file.

3.2.5 | Record filtering

As we have seen in the last subsection, a record (*RecordSam*) is created for each possible combination of HSP between the two reads. Some of the combinations created may be good pairs but others may not. Blast2Bam needs to sort them out and output only the relevant ones in the SAM file. This filtering is done based on the distance between the two reads' alignments on the reference. This distance is called template length and will be displayed in the SAM file under TLEN field (9^{th} column). In SAM format specification [3], TLEN is partially defined as such:

> "If all segments are mapped to the same reference, the unsigned observed template length equals the number of bases from the leftmost mapped base to the rightmost mapped base."

Considering how paired-end sequencing works (see figure 1.1), in a pair, the reads (segments) should be 'facing' each others; the leftmost mapped base should be the 5' end of one of the read and the rightmost, the 5' end of its mate. In other words, unsigned TLEN should be the distance on the reference between the 5' ends of the reads. In BLAST output, the 5' end of a read's alignment is available under `hsp->hsp_hit_from` tag. And so, unsigned TLEN equals to 1 + the absolute value of the difference between the position of the two 5' ends.

To filter the records, we arbitrarily decided that TLEN should not be greater than three times the size on the reference of the longest alignment between the two reads. If a record does not agree with that rule, it will be marked and so will not be printed in the SAM file. If in an *IterationSam*, no record passes that test, the two reads will be considered unmapped.

3.2.6 | Selection of the best record

Along with the filtering, the best record must be chosen. This is necessary for the FLAG creation as we will see in the next subsection. To determine which paired record is the best, a score must be calculated. Each alignment possesses an e-value (`hsp->hsp_evalue`); the smaller, the better. The score of the record is the sum of the e-values of its two alignments. If one of the two reads is unmapped, an arbitrary penalty of +0.5 is applied. The paired record with the smallest score is considered the best and marked as such.

3.2.7 | Sequence Alignment/Map format (SAM) alignment section creation

After input parsing, read pairing, filtering and best record selection, all the necessary elements of the SAM file are recorded, organised and ready to be printed.

As we have seen in Section 2.3, a SAM record is composed of 11 mandatory columns and an optional one. We will now see how Blast2Bam creates them.

1. QNAME: The name of the read is taken from the *SamOutput* structure: `samOutput->query->name`.
2. FLAG:
   - `0x1` is set when the sequencing data is paired-end. If it is unset, `0x2`, `0x8`, `0x20`, `0x40` and `0x80` cannot be set.

- `0x2` is set when the pair of reads is considered properly aligned. As the paired records have already been filtered (see Subsection 3.2.5), every mapped reads coming from paired-end data is flagged `0x2`.
- `0x4` is set when the read is unmapped (no HSP in the record).
- `0x8` is set when the read's mate is unmapped.
- `0x10` is set when the read is mapped on the reverse strand. For the read's HSP, it means that `hsp->hsp_hit_to < hsp->hsp_hit_from`.
- `0x20` is set when the read's mate is mapped on the reverse strand.
- `0x40` is set if the read is the first of its pair. If there are two fastQs, it means the read comes from the first file.
- `0x80` is set if the read is the second if its pair.
- `0x100` is set if the record is a secondary alignment. If after the alignment filtering, the read still possesses more than one record; the best one is selected; the others are flagged secondary (`0x100`) (see subsections 3.2.5 and 3.2.6).
- `0x200` and `0x400` cannot be set by Blast2Bam.
- `0x800` is set if the record is part of a chimeric alignment. For the time being, chimeric alignments cannot be processed by Blast2Bam.

3. RNAME: The name of the reference on which the read is mapped is taken from the *SamOutput* structure: `samOutput->rname`.
4. POS: The position at which the read is mapped is taken from the *SamOutput* structure. The SAM format specification [3] stipulates that POS should be the leftmost position of the alignment on the reference. Depending on which is the leftmost, POS is chosen between `samOutput->hsp->hsp_hit_from` and `samOutput->hsp->hsp_hit_to`.
5. MAPQ: In order to be compatible with downstream software, the mapping quality is set to 60 for mapped and 0 for unmapped reads. The score of the alignment calculated by BLAST can be found in the metadata tags.
6. CIGAR: The CIGAR string is done by comparison between the read and reference sequences displayed in BLAST output (`hsp->hsp_qseq` and `hsp->hsp_hseq`). Unlike other mappers such as BWA, we chose to differentiate matches and mismatches and so display '=' and 'X' and no 'M'.
7. RNEXT: The name of the mate's reference is taken from its *SamOutput* structure in the same manner RNAME was taken for the read. If RNAME and RNEXT are the same, RNEXT is noted '='.
8. PNEXT: The position at which the mate is mapped is taken from its *SamOutput* structure in the same manner POS was taken for the read.
9. TLEN: The template length is taken from the *RecordSam* structure: `recordSam->tlen`. If the read is mapped on the reverse strand, a minus sign is added.
10. SEQ: The Sequence of the read is taken from the *SamOutput* structure: `samOutput->query->seq`. As specified [3], if the read is mapped on the reverse strand, its sequence is displayed reverse-complemented.

11. QUAL: The sequencing quality of the read is taken from the *SamOutput* structure: `samOutput->query->qual`. Similarly to the SEQ column, if the read is mapped on the reverse strand, its sequencing quality string will be reversed.

12. Metadata: The data displayed in this column is optional. This is what Blast2Bam can show:

    - NM corresponds to the number of differences between the read sequence and the reference in the alignment. Its calculation is based on the CIGAR string, one point for every mismatch and every base inserted or deleted (X, I, D or N).
    - RG corresponds to the read group identifier as described in the *@RG* line of the header section.
    - AS is the alignment score. It is the HSP score of the read: `samOutput->hsp->hsp_score`.
    - XB is the alignment bitscore of the HSP: `samOutput->hsp->hsp_bit_score`.
    - XE is the e-value of the alignment: `samOutput->hsp->hsp_evalue`.

3.3 | Upgrades

In order to improve Blast2Bam and make it more user-friendly, some upgrades were made.

3.3.1 | Options

An option system was created to allow the user to tune Blast2Bam to his needs.

- **-o** *file*: modification of the program output. To enable the use of Blast2Bam in a pipeline, its default output is 'stdout' (standard output). This option allows the user to write the SAM output into a file. This option is also available for Fastq2Fasta.
- **-p**: interleaved option. If two fastQ files are given to Blast2Bam, the program will enter automatically in paired-end mode. But if there is only one fastQ, it cannot know if the data is single-end or interleaved paired-end. By default, it will consider it to be single-end. If this option is activated, the program will consider that the given fastQ file is interleaved.
- **-R** *str*: read group. This option associates a read group to all the reads analyzed. The read group will appear in the header section of the SAM file and its ID tag will be written under the RG tag of the metadata field in the alignment section (see 3.2.1).
- **-W** *int*: minimum alignment length. Depending upon its seed length, BLAST can generate small alignments. The user may consider that those alignments are not relevant but still want to keep a small seed length to have the best accuracy possible. This option allows the user to impose a cut-off on the alignment length. If the alignment length is inferior to the given number, the alignment will not be displayed in the SAM file. If, for a given read, no alignment conforms to this rule, the read will be considered unmapped and printed as such. This option has been improved further with a default action of which we will discuss in Subsection 3.3.2.
- **-z**: adjusted position. By default, the position of the alignment (POS) is its absolute position in the reference, 1 being the first position. If the reference is part of a bigger sequence, the user may want POS to show the position in the whole sequence rather than in the smaller fragment. In that case, the position of the reference in the bigger sequence is often displayed in its name (example: chr2:12752–24782). This option makes use of that and adjusts the position displayed

to the position of the reference. Example: an alignment beginning at position 5 of the previous reference will have a POS noted 12757 with this option set.

3.3.2 | Automatic filtering

As we will see in Chapter 4, in the same experiment, the reads can have very different lengths. In that case, a static cut-off to filter the records, like the one we have seen in Subsection 3.3.1, may not be the best solution. For example, a threshold of 40 does not mean much for a read 10000 bases long but it can be very stringent if the read is only 50 bases long. Although this example is a bit extreme, it would be interesting to adjust the filter to the reads' length. We estimated that the size of the alignments should be equal to at least 20% of the length of the read, with a minimum of 20 bases. As this threshold is arbitrary, the user may choose a different one via option `-W` if he sees fit.

3.3.3 | Fasta

Istvan Albert of Pennsylvania State University commented on Biostars (thread 53434):

> "There are so many BAM file visualizers out there and none (that I know of) for BLAST output."

This made us think that some people may have a use of Blast2Bam outside of NGS. And so, it would maybe be interesting for the user to be able to use fasta files instead of fastQs as an input (see figures 2.1 and 2.3 for a reminder of the differences between the two formats).

To allow the use of fasta files, two modifications were necessary:

- The fastQ parser has been rewritten to be able to detect whether the input record is in fastQ or fasta format. If it is in fastQ format, the third and fourth lines will be analyzed. Otherwise, the next line is considered to be a new sequence. As fasta sequences are often written on multiple lines, the parser had to be modified further to accommodate to that characteristic.
- The SAM alignment section creation has also been modified to allow the possibility that there may not be a sequencing quality string to write in the QUAL field. In that case, the indefinite character '*' is used instead.

4 | Results

The goal of this project was to make a short-read alignment program using NCBI/BLAST. To determine whether the results matched our expectations or not, we tested the program on different sets of data and compared the results with BWA. We have chosen BWA as a gold standard because it is the most commonly used software to align NGS DNA data. Furthermore, BWA and our software have the same input and output, making the comparison easier.

The data chosen comes from the Sequence Read Archive (`http://www.ncbi.nlm.nih.gov/sra`):

- ERR656485: Sequencing of Human Immunodeficiency Virus (HIV) performed with Illumina® technology in paired-end. All the reads are 300 bases long.
- ERR732557: Sequencing of phage M13mp18 performed with Oxford Nanopore MinION technology (single-end). The length of the reads is very variable: from 215 to 11213 bases long; average read size: 3909.
- SRR1536433: Sequencing of Escherichia coli, MG1655 strain, performed with PacBio® technology (single-end). The length of the reads is very variable: from 1 to 10001 bases long; average read size: 5142.

The following tests have been performed on the three different sets of data. In order to avoid confusion, we will only present here the results obtained with the HIV data. The results obtained with the other sets are available on GitHub: `https://github.com/guyduche/Blast2Bam/tree/master/test`.

4.1 | Proof of concept

First, we wanted to know if the software worked and if it produced a proper SAM file. To obtain the results, we created a Makefile (Figure 4.1). The program "`makeblastdb`" makes an index of the reference. BLAST needs it to map the sequences. The command "`samtools view -Sb -`" creates a BAM file from a SAM input. The option "`-F 0xF00`" is used to remove the secondary and supplementary alignments (see FLAG in Subsection 3.2.7). We added the SAMtools part to see if the SAM file produced was proper, i.e. compatible with downstream softwares.

Figures 4.2 and 4.3 show a partial representation of the results obtained. These figures were obtained by using `samtools view -h` to make the BAM files readable.

Figure 4.2 first tells us that Blast2Bam is working: there is an output, and this output is accepted by SAMtools. If we compare figures 4.2 and 4.3, we can see that both outputs seems to be similar:

- The flags are correctly formed and are the same as BWA's.
- POS and PNEXT correspond to each other.

```
.SHELL = /bin/bash
FASTQ = ERR656485_1.fastq.gz ERR656485_2.fastq.gz
DB = hiv.fasta

all: results.bam resultsBWA.bam

# BLAST
results.bam: ${FASTQ} $(addsuffix .nin, ${DB})
        fastq2fasta ${FASTQ} | blastn -db ${DB} -outfmt 5 | \
        blast2bam - ${DB} ${FASTQ} | samtools view -Sb -F 0xF00 - > $@

$(addsuffix .nin, ${DB}): ${DB}
        makeblastdb -in $< -dbtype 'nucl'

# BWA
resultsBWA.bam: ${FASTQ} $(addsuffix .bwt, ${DB})
        bwa mem ${DB} ${FASTQ} | samtools view -Sb -F 0xF00 - > $@

$(addsuffix .bwt, ${DB}): ${DB}
        bwa index ${DB}
```

Figure 4.1: Makefile used for the HIV dataset.

- SEQ and QUAL are correctly modified to fit the fact that the first read is mapped on the reverse strand (SEQ reverse complemented and QUAL reversed).

There are however some differences:

- The CIGAR strings are dissimilar. This can be explained by the fact that, as we have seen in Subsection 3.2.7, BWA does not display differently matches and mismatches (`M`).

- TLEN: 120 for BWA and 119 for Blast2Bam. As, in this case, BLAST and BWA have created the same alignment, the two TLEN should be the same. However, if we look at the sequences, we can see that they begin with an 'N'. BLAST does not consider it as a nucleotide and so does not consider it for alignment. This can be seen in the CIGAR string of the first read: it ends with '1S'. When faced with an 'N', BWA replaces it with one of the four nucleotides chosen randomly. This explains that BWA's CIGAR string does not end with '1S' and that its TLEN and NM tag are one base longer.

- The metadata is also different. Apart from the *NM* tag, the others are specific to the aligner used and so should not be considered to compare the outputs.

To confirm that the output conforms to SAM specification [3], we tested it with `ValidateSamFile` from Picard-tools. No error was found.

As the software seemed to work properly, we thought it would be interesting to see if the user could benefit or not from the use of BLAST instead or in complement of BWA.

```
@SQ     SN:gi|9629357|ref|NC_001802.1|     LN:9181
(...)
ERR656485.2     83     gi|9629357|ref|NC_001802.1|     715     60
    180S7=1X8=1X11=1X2=2X4=1X14=1X8=1X33=1X4=1X2=1X5=1X2=1X6=1S     =     715
    -119     CCTAGTGTTGCTTGCTTTTCTTCTTTTTTTTTTCAAGCAGAAGACGGCATACGAGATCCTCTATCGAGA
    TCGGTCTCGGCATTCCTGCTGAACCGCTCTTCCGATCTAAGATAGAGGAAGAACAAAACAAATGTCAGCAAAGTCAG
    CAAAAGACACAGCAGGAAAAAGGGGCTGACGGGAAGGTCAGTCAAAATTATCCTATAGTGCAAAATCTCCAAGGGCA
    AATGGTACACCAGGCCATGTCACCTAGAACTTTAAATGCATGGGTAAAAGTAATAGAGGAAAAGGCCTTTAGCCCAN
    (),.((((,((((((,((.((.-(>69>20E>6/=>5EC@9-52?BEE::2951.)74B64=B==FFAF=A??59:>F
    FFDF:55GGFGF?DFGGFE868>GGGFGGGGED;FGFFGGGGGGGGGGGGEFFGE9GGGGFGGGGGGGGDGECGGFGG
    GGGGGGGGFGGGGGEGGFGGGGGGFFGGGGGFF?EGGFFFEGGGGGGGGFEGGGEGGGFEGGGGGGGGGGDGFFCEG
    FGGGGGGGGGGGFFECFGGGGFGGGGGGGGGGGFCGGGGGGGGGGGGGGGGGGFGGGGGGGGF@CCA8!
    NM:i:13     AS:i:80     XB:f:148.852     XE:Z:4.07e-39
ERR656485.2     163     gi|9629357|ref|NC_001802.1|     715     60
    73S7=1X8=1X11=1X2=2X4=1X14=1X8=1X33=1X4=1X2=1X5=1X2=1X8=106S     =     715
    119     NAGATAGAGGAAGAACAAAACAAATGTCAGCAAAGTCAGCAAAAGACACAGCAGGAAAAAGGGGCTGACG
    GGAAGGTCAGTCAAAATTATCCTATAGTGCAAAATCTCCAAGGGCAAATGGTACACCAGGCCATGTCACCTAGAACT
    TTAAATGCATGGGTAAAAGTAATAGAGGAAAAGGCCTTTAGCCCAGAGATCGGAAGAGCGTCGTGTAGGGAAAGAGT
    GTAGATCTCGGTGGTCGCCGTCTCATTACAAAAAAAACATACACAATAAATGATATAAGCGGAATCAACAGCATGA
    !8A@CGGEFGFGCDFGGGGGGGGGGGGGGFGGGGGFGFGGGGGGGGGGGGGGGGGGGGGGGEGGGGGGGGGGGGGG
    FGGFGGGGGGGGGEGFGFGGGFFGGGGGGGGFGGGGGGGGGGGGFFFFGGGGGG=FFGGFFDGGGGGGGG8FGFGGG
    GGGGGGFGGGGGGGGGGFDGGFGGFGGGFFFGFF8DFDFDFFFFFFFFFBCDB<@EAFB@ABAC@CDFF?4>EEFE<
    *>BDAFB@FFBFF>((6<5CC.;C;=D9106(.))).)-46<<)))))))))))(((,(-)))(((((
    NM:i:13     AS:i:82     XB:f:152.546     XE:Z:3.15e-40
(...)
```

Figure 4.2: Excerpt from the SAM file produced by BLAST–Blast2Bam.

```
@SQ     SN:gi|9629357|ref|NC_001802.1|     LN:9181
(...)
ERR656485.2     83     gi|9629357|ref|NC_001802.1|     715     60
    180S120M     =     715     -120     CCTAGTGTTGCTTGCTTTTCTTCTTTTTTTTTTCAAGCAGAAGAC
    GGCATACGAGATCCTCTATCGAGATCGGTCTCGGCATTCCTGCTGAACCGCTCTTCCGATCTAAGATAGAGGAAGAA
    CAAAACAAATGTCAGCAAAGTCAGCAAAAGACACAGCAGGAAAAAGGGGCTGACGGGAAGGTCAGTCAAAATTATCC
    TATAGTGCAAAATCTCCAAGGGCAAATGGTACACCAGGCCATGTCACCTAGAACTTTAAATGCATGGGTAAAAGTAA
    TAGAGGAAAAGGCCTTTAGCCCAN     (),.((((,((((((,((.((.-(>69>20E>6/=>5EC@9-52?BEE::
    2951.)74B64=B==FFAF=A??59:>FFFDF:55GGFGF?DFGGFE868>GGGFGGGGED;FGFFGGGGGGGGGGG
    EFFGE9GGGGFGGGGGGGGDGECGGFGGGGGGGGGGFGGGGGEGGFGGGGGGFFGGGGGFF?EGGFFFEGGGGGGGG
    FEGGGEGGGFEGGGGGGGGGGDGFFCEGFGGGGGGGGGGGFFECFGGGGFGGGGGGGGGGGFCGGGGGGGGGGGGGG
    GGGGFGGGGGGGGF@CCA8!     NM:i:14     AS:i:55     XS:i:0
ERR656485.2     163     gi|9629357|ref|NC_001802.1|     715     60
    73S121M106S     =     715     120     NAGATAGAGGAAGAACAAAACAAATGTCAGCAAAGTCAGCAAA
    AGACACAGCAGGAAAAAGGGGCTGACGGGAAGGTCAGTCAAAATTATCCTATAGTGCAAAATCTCCAAGGGCAAATG
    GTACACCAGGCCATGTCACCTAGAACTTTAAATGCATGGGTAAAAGTAATAGAGGAAAAGGCCTTTAGCCCAGAGAT
    CGGAAGAGCGTCGTGTAGGGAAAGAGTGTAGATCTCGGTGGTCGCCGTCTCATTACAAAAAAAACATACACAATAAA
    TGATATAAGCGGAATCAACAGCATGA     !8A@CGGEFGFGCDFGGGGGGGGGGGGGGFGGGGGFGFGGGGGGGGG
    GGGGGGGGGGGGGGGEGGGGGGGGGGGGGGFGGFGGGGGGGGGEGFGFGGGFFGGGGGGGGFGGGGGGGGGGGGFFF
    FGGGGGG=FFGGFFDGGGGGGGG8FGFGGGGGGGGGFGGGGGGGGGGFDGGFGGFGGGFFFGFF8DFDFDFFFFFFF
    FFBCDB<@EAFB@ABAC@CDFF?4>EEFE<*>BDAFB@FFBFF>((6<5CC.;C;=D9106(.))).)-46<<))))
    ))))))((,(-)))(((((     NM:i:13     AS:i:56     XS:i:0
(...)
```

Figure 4.3: Excerpt from the SAM file produced by BWA.

4.2 | BLAST–Blast2Bam vs. BWA

As we have seen in Section 1.2, BLAST has not been developed to map NGS reads. Therefore, its parameters may need to be optimized. The efficiency of BLAST is largely dependent upon its seed length (option `-word_size`). The default `word_size` of BLAST is 28, which means that in order to align a read, BLAST needs to find at least a 28 bases long match. It may not be judicious to keep this default behaviour when working with short-reads. Therefore, when comparing BLAST and BWA, we needed to test different `word_sizes`. This comparison has been done on several parameters.

4.2.1 | Time

First, we decided to evaluate the time spent by both softwares to align the reads. It was obtained using `GNU time`. On our computer, 51 seconds were necessary for BWA to output the results. The results concerning BLAST–Blast2Bam can be seen on figure 4.4a.

We can see that the smaller the seed length is, the slower BLAST–Blast2Bam is. A small seed length means that BLAST initially researches smaller combinations (small perfect alignments) between the reads and the reference. Smaller combinations implies that BLAST will have a higher probability of finding hits. The time necessary for BLAST to elongate the hits to form HSP is proportional to the number of hits [1]. And so, the smaller the seed length is, the more time BLAST will need to align the sequences.
With a `word_size` of 8 (minimum), 1182 seconds (19 min 42 s) are needed to output the results.
With a `word_size` of 28 (maximum), 910 seconds (15 min 10 s) are needed to output the results.

If we compare the results from both softwares, BWA is unquestionably much faster than BLAST–Blast2Bam. As we have seen in Section 1.3, part of why BWA was created was to make up for the lack of speed of the previous generation aligners in order to be able to work with whole genome data. Knowing this, it seems coherent that, in our tests, BWA has been found to be faster than BLAST.

4.2.2 | Number of reads mapped

Next, we wanted to see whether BLAST–Blast2Bam could atone for its sluggishness with a larger quantity of reads mapped to the reference. To obtain the results, we used `samtools idxstat`. The total number of reads for the HIV dataset is 822480 (411240 for each fastQ). Of them, 554181 are mapped by BWA (67.38%). The results obtained with BLAST–Blast2Bam are exposed in figure 4.4b. As expected, we can see that as the `word_size` increases, the number of reads mapped decreases.
With a `word_size` of 8 (minimum), 547233 reads are mapped (66.53%).
With a `word_size` of 28 (maximum), 369912 reads are mapped (44.98%).

If we compare the results obtained with BLAST–Blast2Bam and BWA, we can see that even at its best, BLAST does not map as much reads as BWA. However, this may be due to Blast2Bam record filtering. As we have seen in Subsection 3.3.2, Blast2Bam filters its results based on their alignment length. If we deactivate the filtering (`-W 0`), the results are completely different: 788364 mapped (95.85%) with `word_size 8`.

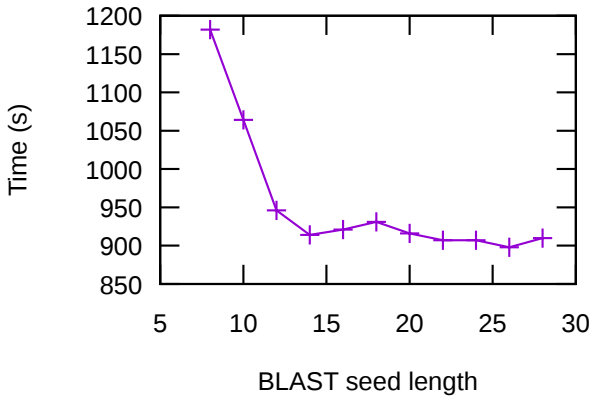
With the filtering, the aim of BLAST–Blast2Bam is not to have the most reads mapped but to output alignments with a better quality.

4.2.3 | Coverage

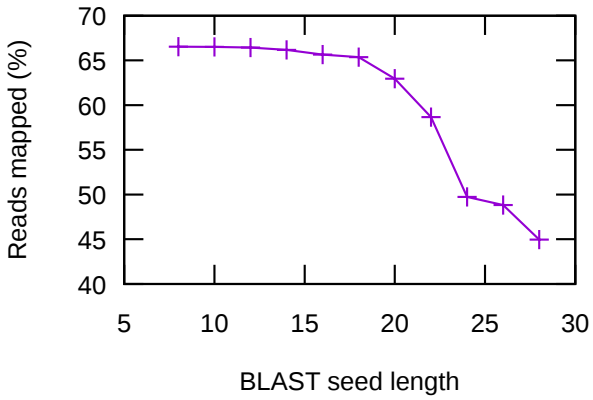
To assess if BLAST–Blast2Bam could procure overall better alignments than BWA, we decided to compare their results' coverage. The coverage corresponds to the fraction of the reference mapped by the reads. Ideally, the reads would cover 100% of the reference.

`samtools depth` computes the depth of the sequencing (number of reads aligned for each reference base). From that, we can extract the number of reference bases used in an alignment and then calculate the coverage with the total number of bases in the reference. For BWA, we obtained a coverage of 7747/9181 bases (84.38% of bases covered). BLAST–Blast2Bam results are exposed on figure 4.4c. Similarly to what we observed for the number of reads, we can see that as the `word_size` increases, the coverage decreases.
With a `word_size` of 8 (minimum), the coverage is equal to 8707/9181 bases (94.84% of bases covered). With a `word_size` of 28 (maximum), the coverage is equal to 7704/9181 bases (83.91% of bases covered).

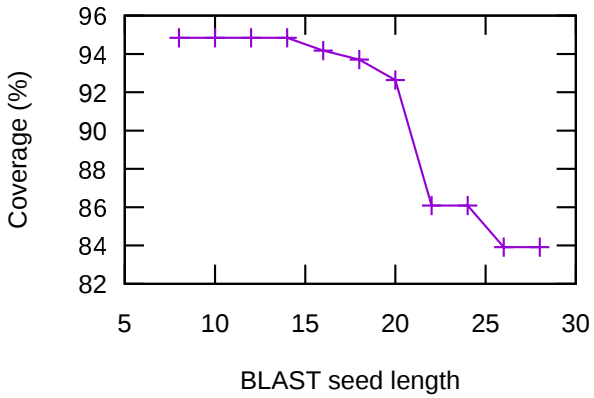
Interestingly, we can see that, for a `word_size` below 26, BLAST can, with this dataset, provide a better coverage than BWA. To see it more clearly, we did a graphical representation of the alignments with IGV. Figure 4.5 shows an excerpt from the representations of the alignments obtained with BLAST–Blast2Bam (`word_size 8`) and BWA. The upper part of the picture represents BLAST–Blast2Bam results and the lower part, the alignments from BWA. We can see that, in that case, BLAST–Blast2Bam seems to be able to align reads in regions where BWA has difficulties mapping them.



(a) Time needed by BLAST–Blast2Bam.



(b) Percentage of reads mapped.



(c) Coverage.

Results obtained with BWA:

- Time: 51 seconds
- Reads mapped: 67.38%
- Coverage: 84.38%

(d) BWA results

Figure 4.4: Results obtained with BLAST–Blast2Bam (a,b,c) and BWA (d). (a) Time needed by BLAST–Blast2Bam to output the results. (b) Percentage of reads mapped. (c) Coverage. (a,b,c) are expressed as functions of BLAST seed length. (d) BWA results.

Figure 4.5: IGV representation of the alignments obtained with BLAST–Blast2Bam (`-word_size 8`) and with BWA. Upper part: BLAST–Blast2Bam. Lower part: BWA.

5 | Conclusion

The aim of this project was to develop a software able to align short-reads using NCBI/BLAST. Following these objectives, we created two programs:

- Fastq2Fasta to allow the use of short-reads by BLAST;
- Blast2Bam to extract the results from BLAST, pair them if need be, analyze them and finally output them in a format which will allow for further analyses by downstream software (SAM).

After implementation, we used it on real data and tested if the output was conform to the format specification and if the results were comparable with those obtained with another aligner. Blast2Bam passed the tests. A number of upgrades were then issued to make the programs better, able to work with different types of data and more suited to the user's needs.

We then wondered if BLAST–Blast2Bam could be better, in certain circumstances, than the current gold standard, BWA. Both programs were evaluated on different parameters: time spent, number of reads mapped and coverage. BLAST–Blast2Bam was definitely slower than BWA but seems to be better in terms of coverage. More tests would be necessary to confirm it.

Although Fastq2Fasta and Blast2Bam are working properly, there is still room for improvement. For example, nothing was done, yet, concerning chimeric alignments, i.e. a read of which two different parts have been found to be matched at different positions on the reference. Example: the first half of the read is mapped at a position on the reference and the second half is mapped at another place on the reference. Currently, Blast2Bam considers both but will output the one with the lowest score as the best (see Subsection 3.2.6) and the other one as secondary.

As reported by Peter Cock on http://blastedbio.blogspot.fr, the NCBI is working on a SAM output for BLAST. It is present in the current version of BLAST+ (2.2.31) as an undocumented option (`-outfmt 15`). Like blast2sam.pl, it seems to be a converter. As of now, it does not produce a proper header, the read names are changed, there is no pairing and the sequence quality is lost. However, it seems to work also with protein sequences. Although Blast2Bam has been developed with genetic data in mind, it might be interesting to adapt it to enable the use of protein sequences.

To make our software available to the community, we pushed all the source code, tests and manual, including the code of this master's thesis on GitHub at `https://github.com/guyduche/Blast2Bam` where, to our delight, it seems to have been appreciated.

Bibliography

[1] Stephen F. Altschul et al. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990. PMID: 2231712.

[2] David R. Bentley et al. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456:53–59, 2008. PMID: 18987734.

[3] The SAM/BAM Format Specification Working Group. *Sequence Alignment/Map Format Specification*. Url: `https://samtools.github.io/hts-specs/SAMv1.pdf`.

[4] Daniel Veillard. Libxml2 xmltextreader interface tutorial.
Url: `http://www.xmlsoft.org/xmlreader.html`.

The electronic version of this report contains hyperlinks with additional references.