Structural Variant Detection

A Novel Approach

by

Sheetal Shetty

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2014 by the
Graduate Supervisory Committee:

Valentin Dinu, Chair
Matthew Scotch
Kimberly Bussey
Garrick Wallstrom

ARIZONA STATE UNIVERSITY

May 2014

ABSTRACT

Genomic structural variation (SV) is defined as gross alterations in the genome broadly classified as insertions/duplications, deletions inversions and translocations. DNA sequencing ushered structural variant discovery beyond laboratory detection techniques to high resolution informatics approaches. Bioinformatics tools for computational discovery of SVs however are still missing variants in the complex cancer genome. This study aimed to define genomic context leading to tool failure and design novel algorithm addressing this context. **Methods**: The study tested the widely held but unproven hypothesis that tools fail to detect variants which lie in repeat regions. Publicly available 1000-Genomes dataset with experimentally validated variants was tested with SVDetect-tool for presence of true positives (TP) SVs versus false negative (FN) SVs, expecting that FNs would be overrepresented in repeat regions. Further, the novel algorithm designed to informatically capture the biological etiology of translocations (non-allelic homologous recombination and 3-D placement of chromosomes in cells-**context**) was tested using simulated dataset. Translocations were created in known translocation hotspots and the novel-algorithm tool compared with SVDetect and BreakDancer. **Results:** 53% of false negative (FN) deletions were within repeat structure compared to 81% true positive (TP) deletions. Similarly, 33% FN insertions versus 42% TP, 26% FN duplication versus 57% TP and 54% FN novel sequences versus 62% TP were within repeats. Repeat structure was not driving the tool's inability to detect variants and could not be used as context. The novel algorithm with a redefined context, when tested against SVDetect and BreakDancer was able to detect 10/10 simulated translocations with 30X coverage dataset and 100% allele frequency, while SVDetect captured 4/10 and BreakDancer detected 6/10. For 15X coverage dataset with 100% allele frequency, novel algorithm was able to detect all

ten translocations albeit with fewer reads supporting the same. BreakDancer detected 4/10 and SVDetect detected 2/10 **Conclusion**: This study showed that presence of repetitive elements in general within a structural variant did not influence the tool's ability to capture it. This **context-based algorithm** proved better than current tools even with half the genome coverage than accepted protocol and provides an important first step for novel translocation discovery in cancer genome.

DEDICATION

This work is dedicated to my family without whose unwavering solid support I would not have tested the limits of by ability.

ACKNOWLEDGMENTS

I would like to thank Dr. Kimberly Bussey for directing and guiding the research from the first day. Without her support and guidance this work would not have been possible. I would also like to thank my committee for their very valuable inputs at various stages of research which shaped a better analysis.

Translational Genomics Institute (TGen), Phoenix, deserves special thanks for access to its bioinformatics resources and training I received as an intern and a graduate student. Dr. Waibhav Tembe, Dr. Christophe Legendre, and Dr. Raghu Metpally were all contributors and guides to my learning process at TGen with special mention for Dr. Tembe, who suggested the vigorous testing of the first hypothesis of this analysis.

Also a special thanks to Dr. Matthew Hayes from Case Western Reserve University for providing the Perl code which was used to create simulated translocations for testing purpose in this study.

This acknowledgment would be incomplete without special thanks to all blog posts on bioinformatics websites, specifically “seqanswers” and “biostars” whose precise and timely posts helped me solve many issues during the analysis process.

TABLE OF CONTENTS

Page

Page

CHAPTER

Page

CHAPTER

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

1. INTRODUCTION

Identifying differences at the genome level between a diseased and healthy individual has been the cornerstone of current medical genetic research with the purpose of identifying and targeting disease causing variants.

1.1 Overview

Structural variations in the human genome are changes in the genome, when compared to the reference human genome, that lead to gross aberrations in the physical structure of the genome. These typically include insertions, deletions, inversions, and translocations. Structural variations are typically studied by comparing the target DNA to the consensus reference genome. Insertions represent the inclusion of a segment of DNA sequence inserted into the target genome. Similarly a deletion represents a deleted segment, duplication represents a duplicated or repeat segment, and an inversion is a change in the orientation of the segment. A translocation is a change in the physical location of a segment of DNA normally present on the reference chromosome to another chromosomal location in the donor. Structural variations play a very important role in cancer development. Translocations were the earliest identified variants in cancer with the identification of Philadelphia chromosome (translocation between chromosomes 9 and 22 [t(9;22)]) as a hallmark of chronic myeloid leukemia (CML) (Nowell & Hungerford, 1960; Rowley, 1973). These gross variants in cancer produce fusion genes as BCR-ABL1, between a breakpoint cluster region on chromosome 22 and tyrosine kinase receptor gene chromosome 9 that is a recurrent phenomenon in CML and one of the diagnostic criteria for this disease. Another commonly occurring fusion gene between a transmembrane protease gene (TMPRSS2) and one of two transcription factors (ETV1 or ERG) was first reported in 23 of 29 prostate cancers (Tomlins et al., 2005).

These commonly occurring fusion genes are fast becoming the hallmark of cancer diagnostics with researchers now aiming to define the etiology of this recurrence.

1.2 Significance of the Problem

Detecting structural variations in the cancer genome is the first step towards extracting disease causing mutations. Current tools using well-examined bioinformatics approaches to variant detection are still missing variants in the cancer genome. The reasons for this could be due to allele frequency of the mutant genome captured in an experiment, due to tumor heterogeneity, mapping algorithm flaws or structural variant detection algorithm flaws. In addition, these tools have been designed without accounting for the very special case of cancer genome complexity. Understanding the biological processes leading to structural variation generation in the cancer genome and using this context to define the algorithm for variant detection is an overlooked novel approach explored in this study, specifically in the study of translocations.

Chromosomal translocations are the most obvious signature for many cancers and serve as a very important biomarker. Cancer genomes have inherent genetic instability either due to various micro-environmental factors leading to somatic mutations or inherent predisposition in the genome leading to germline mutations. Normal cellular machinery has checks to curb such major overhauls using DNA repair mechanism. The cancer genome however has circumvented this repair mechanism; most evident in the resulting fusion-genes which produce protein products which further disrupt the repair infrastructure.

DNA sequencing technologies aim to identify these variants at a faster rate and with more accuracy based on the sheer volume of data exiting the machines. Clinical translation of this information helps to design specific drug targets for these specific

translocation fusion gene products. Two major hurdles to achieve this purpose, namely the presence of the human reference genome and the reducing cost of sequencing the patient genome have now been surmounted. Bioinformatics challenges to map and align this large amount of data, though not fully conquered, have made significant progress. The challenge now facing the scientific community is how to make sense of the data by designing better algorithms for detecting variants.

Every approach available in the literature has used some form of statistical/mathematical modeling or computational algorithm to solve this issue of detecting structural variants without paying much attention to the biology driving these variants. Computational methods to detect structural variants (SV) utilize a consistent algorithm (Tuzun et al., 2005) including: 1) creating a distribution of the reads length (insert size) to derive a mean and standard deviation (s.d) 2) defining SV signatures (e.g. insertion = reads mapping 3 s.d. outside the mean length), and 3) clustering all reads which support the same SV. The current study used a different approach using biological domain information which is already known and well documented in literature to help design a more effective and biologically plausible algorithm to detect translocations in the cancer genome.

1.3 Theoretical basis for the study

Cancer occurs due to genomic instability that leads to disruption of normal cell functions. These mutations and structural alterations can be germline or somatic. Genomic mutations in germline cells lead to their transmission to the next generation as a cancer susceptibility gene like BRCA1 (P. Kent et al., 1995). Mutations in somatic cells can cause cancer due to abnormal proliferation of somatic cells with aberrant genomic structure. Identifying these gross structural changes has been done traditionally using experimental technique like karyotyping. Karyotyping uses stains (Giemsa stains) to color the chromosome during mitosis and studying the

banding patterns (G-banding) to define structural abnormalities like duplication of a region of chromosome or presence or absence of chromosomes (McNeil, Montagna, Difilippantonio, & Ried, 2012). However the primary issue with karyotyping remains to be the resolution of the abnormal chromosomal region, which is 5-10 million base pairs (Mbps). A successful experiment is also dependent on quality of cell division rates in metaphase, which is when these chromosomes are captured (Bridge & Cushman-Vokoun, 2011).

One of the major advances in cytogenetic diagnosis was development of fluorescent in-situ hybridization (FISH) techniques and its various offshoots (Volpi & Bridger, 2008). The principle was based on hybridization of complementary DNA to specific fluorescent probes, designed for specific regions on the genome, and viewed through special cameras for detecting structural abnormalities (McNeil et al., 2012). These technologies have far reaching applications from pre-natal detection of structural abnormalities (Hastings, Nisbet, Waters, Spencer, & Chitty, 1999), profiling gene expression during meiosis in mammalian cells (Mahadevaiah, Costa, & Turner, 2009) to identifying novel fusion genes in leukemias (H. Lee et al., 2013). Further these techniques have been seminal in identifying specific recurrent translocations in tumors leading to targeted therapy for fusion genes (Buchdunger et al., 1996; Druker et al., 1996; Inokuchi, 2006; Lynch et al., 2004; Mathews et al., 2010; Niu et al., 1999; Paez et al., 2004). However the resolution of these methods can be about 1 kbps depending on the size of the probe, the target locus and type of FISH being used. A subsequent development, also based on hybridization, known as comparative genomic hybridization (CGH) (Kallioniemi et al., 1992) uses tumor and normal control samples to compare copy number variation between normal and tumor tissue and even detect gene fusions (Przybytkowski, Ferrario, & Basik, 2011). These methods have higher resolution of 500 bps to even 50 bps but can be

extremely sensitive to the quality of tumor samples and therefore have a high rate of errors (Fragouli et al., 2011). These methods are thus dependent on the quality of the probes and the genomic distance between the probes on the array. Furthermore, they can only detect aberrations that alter copy number; copy-number neutral events are missed.

With the advent of massively parallel high throughput sequencing technologies gaining ground due to reduction in cost, mutation definition is getting higher and higher resolution with the number of implicated oncogenes growing from 291 in 2004 (Futreal et al., 2004) to 384 in 2010 (Santarius, Shipley, Brewer, Stratton, & Cooper, 2010) and the most recent database of cancer mutations, 'Mitelmans Database of Gene Chromosome Aberrations and Gene Fusions in cancer', a National Cancer Institute (NCI) resource showing 2038 gene fusions(Mitelman, Johansson, & Mertens, 2014). These next generation sequencing methods have provided enormous amount of data and with it daunting bioinformatics challenges. Even with obvious computational challenges of analyzing large data, sequencing has identified novel causative mutations that were experimentally validated in diseases like mental retardation where there was no familial history (Vissers et al., 2010). Similar success was also seen in autism which is characterized as a multi spectrum disease. Exome-sequencing of 928 autistic individuals identified 279 novel coding mutations (Sanders et al., 2012), a feat if not unachievable would be highly labor intensive in an experimental procedure.

Sequencing methods currently produce paired-end, short reads (30 -100 base-pairs) such that both ends of a sequence segment are read with an intervening region of unread sequence in the middle. The two reads are paired such that they have the same identifier linking the two reads. Sequencing technologies today have adopted

paired end short read sequencing as the preferred method due to the relative low cost. The bioinformatics challenge is to map these short sequences correctly to the reference human genome and be able to discern variants. The algorithm developed for detecting variants was first laid down by Tuzun *et al*. (Tuzun et al., 2005) as:

- mapping the subject genome to the reference genome
- defining signatures for structural variation using the distribution based on size of the short reads (inserts)
- identifying the reads which support the structural variant as described by the signature and clustering all reads which support the same variant together, and finally
- removing false positives based on percent identity of the variants read with the reference genomes

Current tools use this basic model for defining structural variation with statistical adjustments for calculating sensitivity and specificity, and sometimes expanding the signatures for calling structural variants. These work very well in normal genomes and reasonably well in cancer genomes. However, cancer genomes are more complex as samples represent an admixture of normal and abnormal cells, the latter of which may differ genomically. The ability to detect somatic mutations in a given sample depends on tissue type, type of mutation (*i.e.* germline versus somatic), amount of intercellular heterogeneity in the tumor, the sequencing technology itself, and most importantly the algorithm used to detect variants. The choice of sequencing technology/platform also depends on the goal of the study. The comparison of sequencing platforms in metagenomic studies revealed that the use of short-read generating technology was better due to significantly increased number of

reads and therefore greater coverage of genomic regions (Mende et al., 2012). In contrast, a comparison of these sequencing technologies for low-coverage experiments in clinical setting to detect major copy number changes as part of prenatal diagnosis found that increased number of reads produced in one technological platform also made it more prone to GC-content bias due to greater number of PCR-cycles (S. Chen et al., 2014). After choosing the right platform for the study, the next challenge is to determine the appropriate tools to analyze the data and call variants. All major bioinformatics development has been open source, and tools such as National Center for Biotechnology Information, NCBI's BLAST (**B**asic **L**ocal **A**lignment **S**earch **T**ool), University of California Santa Cruz, UCSC's Genome Browser, and Burrow-Wheeler aligner (BWA) have become the de-facto standards for alignment and graphical viewing of the genome. However, new variant detection algorithms are constantly being designed to address specific problems. For example, a study comparing 12 algorithms for quantifying somatic copy number variation using whole genome sequencing data (WGS) found that there were significant differences in sensitivity and specificity of these algorithms (Alkodsi, Louhimo, & Hautaniemi, 2014). Sensitivity depended on the size of the variant being detected, while breakpoint detection accuracy was determined by the algorithmic approach of the tools. These studies demonstrate the bioinformatics challenges of analyzing DNA sequencing data, specifically variant detection. The multitude of data being generated by next generation sequencing and the analogous growth in bioinformatics tools to deal with this data shows the nascent state of the field in terms of standardized methodologies to analyze this data.

1.4 Variables used in the study

Taking the computational modeling for detection of SVs first presented by Tuzun *et al.* (Tuzun et al., 2005) further, Lee *et al.* (S. Lee, Cheran, & Brudno, 2008)

proposed a probabilistic method for variant calling and controlling the false discovery rates. Many currently popular tools use this methodology with modifications. For example, SVDetect (Zeitouni et al., 2010) uses a windowing strategy in the paired end library together with the clustering mechanism explained above to define variants. BreakDancer (K. Chen et al., 2009) uses a more stringent method for defining probabilities and therefore reducing the output of false positives. CREST (Wang et al., 2011) on the other hand used another signature called soft clipping[1] to extract anomalous reads, build contigs[2] of soft-clipped reads and subsequently define variants based on similar probability calculations discussed above. Thus, all of these tools work off the alignments produced by the mapping tool. However, this can be a problem when there are multiple mapping regions with similar identity in the BWA mapping tool algorithm (Li & Durbin, 2009). During these instances the tool will randomly assign the read to any location. This is a characteristic of the human genome: there are regions with considerable sequence homology and repeats such that the mapping tool fails to align sequences at a unique region on the genome. The variant detection tools completely ignore the fact that the alignment reported by the mapping tool may be a random position.

These repeat regions comprise up to 50% of the human genome (Smit, Hubley, & Green, 2014) and play a very significant role in development of these structural variants due to errors in the DNA repair process (G. McVean, 2010). McVean has reviewed these recombination events that lead to mutation and identified hotspots in the genome that were more prone to rearrangement breakages. Further, these

[1] Soft clipping: Paired end read mapped to the reference genome where one end mapped globally an d the other mapped partially, may need the unaligned ends of the read ‘clipped’ to achieve mapping. The read is labeled as ‘S’ in its CIGAR string (Section 4.2.1).

[2] Contig: Creating a longer sequence of DNA from overlapping smaller subsequences.

regions tended to be closer to promotor regions but not in the transcribed regions and were driven by a particular class of repair system called non-allelic homologous recombination (G. A. McVean et al., 2004). Non-allelic homologous recombination events (NAHR) are the result of errors in DNA repair involving large chromosomal regions characterized by low copy repeats and up to 95% sequence identity (Stankiewicz & Lupski, 2002), (Colnaghi, Carpenter, Volker, & O'Driscoll, 2011a). Thus, there was information within the genome with a specific signature that was leading to error prone repair. Further, Ou *et al*. (Ou et al., 2011) were able to specifically prove NAHR as the cause of same unbalanced translocation in four unrelated families. The group was also able to map these 'NAHR' regions on the human genome based on the signature of low copy repeat regions with greater than 94% sequence identity. They were also able to find validated translocations in the database in these predicted rearrangement hotspots.

Current tools ignore this very relevant information driving these structural variation events in the genome. There is a general consensus in the scientific community that variant calling tools fail in repetitive regions (non-unique regions) of the genome because the mapping algorithms cannot reliably map these regions. This leads to a decrease in the signal-to-noise ratio, and thus the number of reads supportive of an aberration. While this is true, there is enough information known about these regions to account for them algorithmically in a variant detection tool.

DNA damage is acquired primarily during replication process of the cell. Cells have checks and balances to control for errors during repair and a breakdown of these repair mechanism leads to non-allelic homologous regions of the genome undergoing recombination and going unchecked. A low copy repeat region will share homology to many regions on the genome and even share significant identity with these regions.

The 3-D structural organization of the genome within the cell, as studied by Hi-C experiments showed that regions of chromosome within close proximity to each other were more likely to interact compared to regions that were far away (Lieberman-Aiden et al., 2009). Another study probing the effect of DNA replication timing on generation of copy number variations in the cancer genome also found the correlation of spatial organization of the chromosomes within the genome affecting the occurrence of mutations at specific locations (De & Michor, 2011). They went further with the hypothesis proving that those regions that were closer to each other spatially in the cell were also likely to have similar replication timing during cell division.

1.5 Problem Statement

The purpose of this research was to use the known genomic context driving the formation of chromosomal aberrations, both in terms of repeat structure and 3D spatial organization, to design a biologically sound computational algorithm to detect translocations in the cancer genome.

1.6 Research Question and Hypothesis

1. Are the tools able to detect validated structural variants from a known dataset?

2. Will the tools fail in regions of repeats?

3. Is the ability of the tool to detect variants driven primarily by the presence of these variants in high complexity (unique) regions?

Research Hypothesis

1. Tools fail when the variants fall in the repeat regions. Variants in unique regions are more likely to be detected by the tools compared to variants in repeat regions.

2. A novel algorithm designed to take into account the genomic context that drives the formation of chromosomal alterations (De & Michor, 2011; Lieberman-Aiden et al., 2009) as well as the genomic architecture of cancers (regions of the genome more susceptible to structural variant formation as seen in recurrent cancers) is likely to perform better than current tools that do not include this information.

2. BACKGROUND

2.1 DNA Sequencing

DNA sequencing is the decoding of genetic information locked in the DNA and is the machine translation of the nucleotide sequence that makes up the three billion bases of human genetic code. Fred Sanger introduced the chain-termination method for base determination (Sanger, Nicklen, & Coulson, 1977) (Figure 2.1 (Estevezj., 2012)) which gained wide acceptance as the preferred method for sequencing. Although another method by Maxam-Gilbert (Maxam & Gilbert, 1977) was introduced at the same time and used base-specific chemical degradation, the Sanger method became more popular due to its ease of use (Nunnally, 2005).

DNA sequencing can be broadly divided into four steps (Nunnally, 2005):

- Reaction
- Separation
- Detection
- Data analysis

The reaction step is specific to the type of method being used. Broadly, double-stranded DNA is broken mechanically or chemically into single-stranded DNA, mixed with a DNA polymerase[3], DNA primer[4], the four deoxynucleotide bases (adenine, guanine, tyrosine and cytosine) and one dideoxynucleotide (ddNTP) corresponding each of the bases that when incorporated during the polymerase reaction stops the lengthening of the DNA chain. Thus, the reaction yields a multiplicity of different

[3] Polymerase: Enzyme present in the cells, used during DNA replication process for synthesizing a new strand of DNA from a copy/template.

[4] Primer: Short DNA segment of known sequence which attached to the DNA strand at its end

sized fragments where the length of the fragment depends on the chance incorporation of a ddNTP into the chain. Each type of radioactively or fluorescently labeled ddNTP (ddATP, ddTTP, ddCTP, ddGTP) is aliquotted into a separate reaction, and all four reactions are required to generate a sequence.

The separation step involves separating the DNA fragments obtained from the reaction step based on size. Earlier methods used polyacrylamide gel electrophoresis, in which the DNA fragments travelled vertically through the gel under a steady current for a set period of time. The distance traveled was dependent on size, with smaller fragments migrating more quickly than larger ones. Thus sequence was read from the bottom of the gel up. Currently, this process is done by the capillary based system where the sample are run through a very fine capillary and can be read simultaneously by the detector as the DNA sample is travelling through the capillary.

The detection of the separated fragments involves exposing the separated sample to X-ray film for radioactive labeling. Once the film was developed, the sequence of the DNA could be read from the bottom up by recording in which lane the smallest fragment appeared, followed by the next smallest, and so on. However, this method is not routinely used anymore. It has been replaced by the use of fluorescently labeled ddNTPs exposed to laser light that is simultaneously detected by the detecting machine. In the analysis step, all the data is compiled into a single continuous sequence.

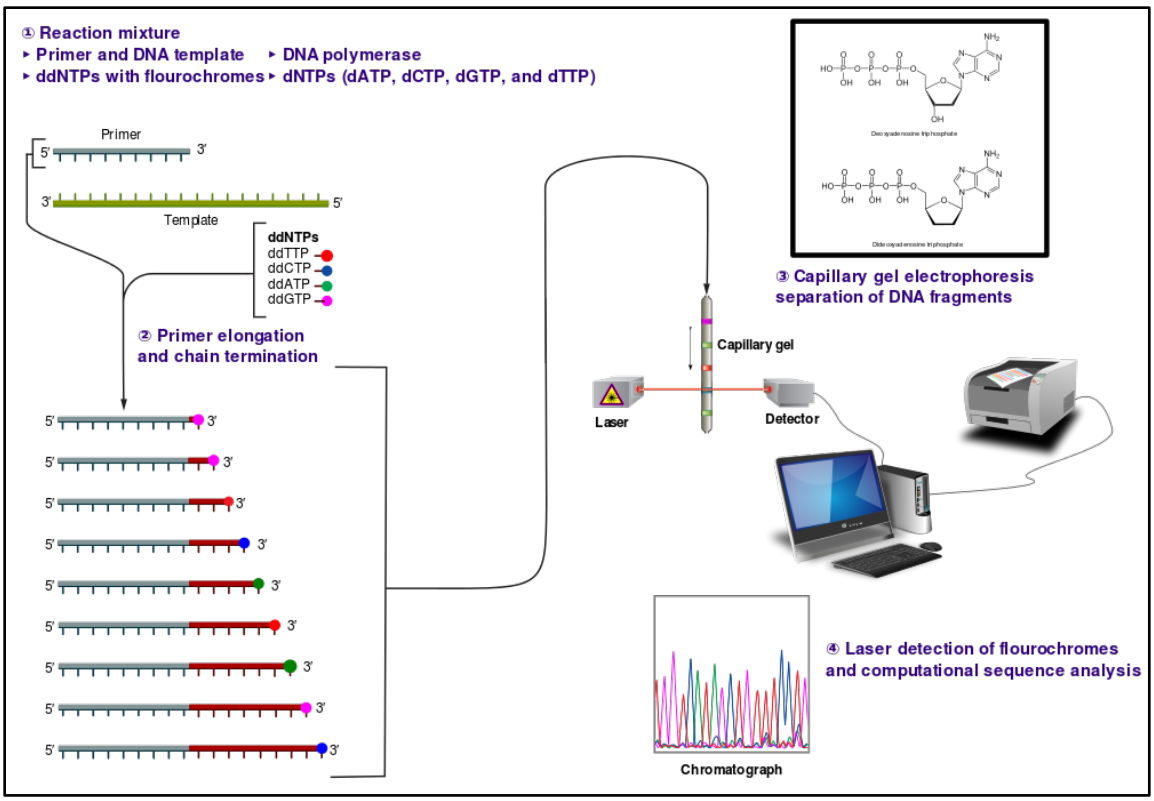


Figure 2.1. Sanger DNA sequencing method. This figure explains the steps in modern Sanger technology. Adapted from Sanger Sequencing, by Estevezj, Retrieved March 27, 2009, from http://commons.wikimedia.org/wiki/File:Sanger-sequencing.svg. Copyright 2012 by Estevezj. Reprinted under Creative Commons Attribution-Share Alike 3.0 Unported license.

The automation of the Sanger-sequencing method led to the human genome project with the aim of sequencing the entire human genome and was completed in 2001 (Venter et al., 2001). The project was completed much earlier than expected using the above method together with shotgun assembly. In shotgun assembly process the entire genome is broken into random smaller pieces, these pieces are amplified by first cloning in bacterial cell (plasmids/bacterial artificial clones) and then through PCR[5]. These amplified PCR products have known fragment sizes (also known as insert size). The major contribution of this group was the development of mate-pair

[5] PCR: Polychromase chain reaction, a procedure to create multiple copies of the DNA using DNA polymerase

reading methods. Different sized libraries of these inserts were created such that multiple clonally amplified fragments of circularized DNA sequences from each library are cut into linear pieces and read from both ends, put together to form contigs with unread gaps and finally contigs assembled together bioinformatically into scaffolds in the analysis step (Istrail et al., 2004). The size of the gap can be estimated with reasonable confidence based on the segment length (*i.e.* insert size) of the DNA library. This mate-pair sequencing and assembling method was a major step in the direction of high throughput analysis. The completion of the human genome assembly was a significant achievement as the scientific community now had the entire human genome decoded and publicly available through NCBI.

Technology for faster sequencing has improved over the years with a corresponding decrease in the cost due to the development of these new sequencing methods. These newer-generation sequencing technologies are now collectively called next-generation (next-gen) sequencing. The most commonly used next-gen applications include Roche/454, Illumina/Solexa, Life/APG and Helicos BioSciences (Metzker, 2010). The flow of steps more or less is the same as Sanger sequencing, including template preparation, sequencing, viewing and data analysis, with major improvements in template preparation and sequencing.

Template preparation has seen major advances in next generation sequencing methods with shift from bacterial artificial chromosomes (BACs) (Monaco & Larin, 1994) (Shizuya & Kouros-Mehr, 2001) due to inherent problems with BAC procedures, including loss of genomic material in the BAC during cell replication and introduction of replication errors as human errors during the mapping process. The two major types of template preparation in NGS technology are:

1. Clonally amplified templates

2. Single-molecule templates (Metzker, 2010)

Clonally amplified templates use single stranded DNA molecule with universal primer attached to beads (Figure 2.2)

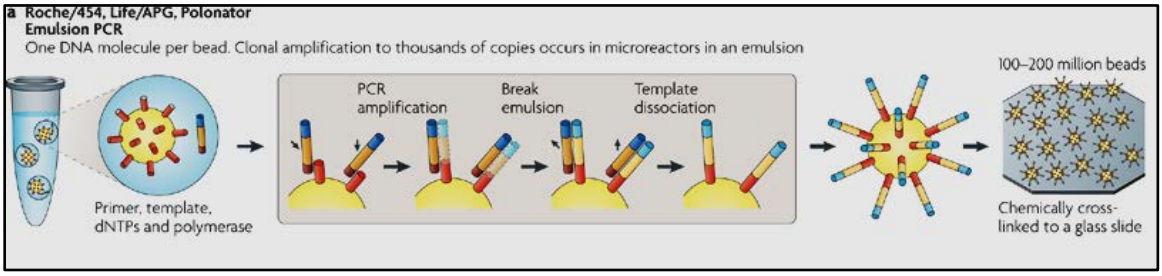


Figure 2.2. Emulsion bead template preparation for next generation sequencing technology. Adapted by permission from Macmillan Publishers Ltd, "Sequencing technologies - the next generation", by M.L.Metzker, 2010, Nature reviews.Genetics, 11, p.33, copyright 2010.

These PCR-amplified beads can then be placed on a glass slide for the NGS sequence reading to be performed.

Solid phase amplification or bridge amplification can also be performed (Figure 2.3), where the primers are attached to glass slide and DNA fragments along with polymerase are added to the glass slide to produce spatially-separated clones of amplified DNA fragments.

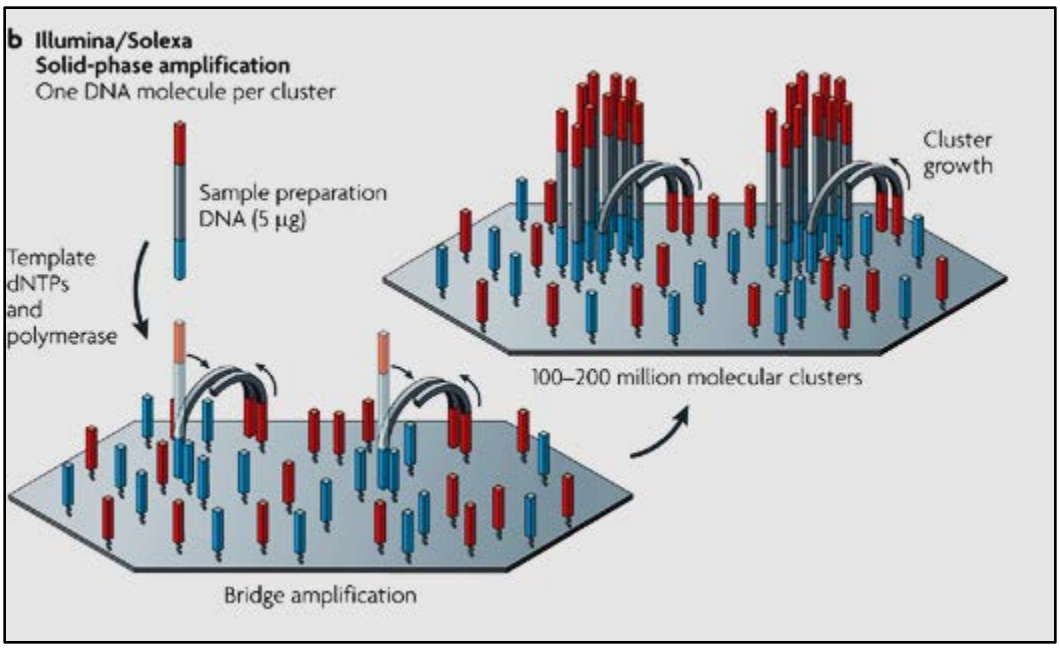


Figure 2.3. Bridge amplification template preparation for next generation sequencing technology. Adapted by permission from Macmillan Publishers Ltd, "Sequencing technologies - the next generation", by M.L.Metzker, 2010, Nature reviews.Genetics, 11, p.33, copyright 2010.

A single molecule template has a major advantage of not requiring PCR amplification and therefore reducing errors. Thus there is no need for clonal amplification in this method and it is believed to be more representative of the original sample. This is achieved by either attaching primers to the glass slide and adding single-stranded DNA molecules to these immobilized primers (Figure 2.4) or attaching the single-stranded DNA molecules to the glass slide and then adding the primers to these immobilized DNA strands (Figure 2.4). A new approach uses the DNA polymerase bound to a glass slide and the single-stranded DNA template can be introduced to this polymerase and read in real-time (Figure 2.5).

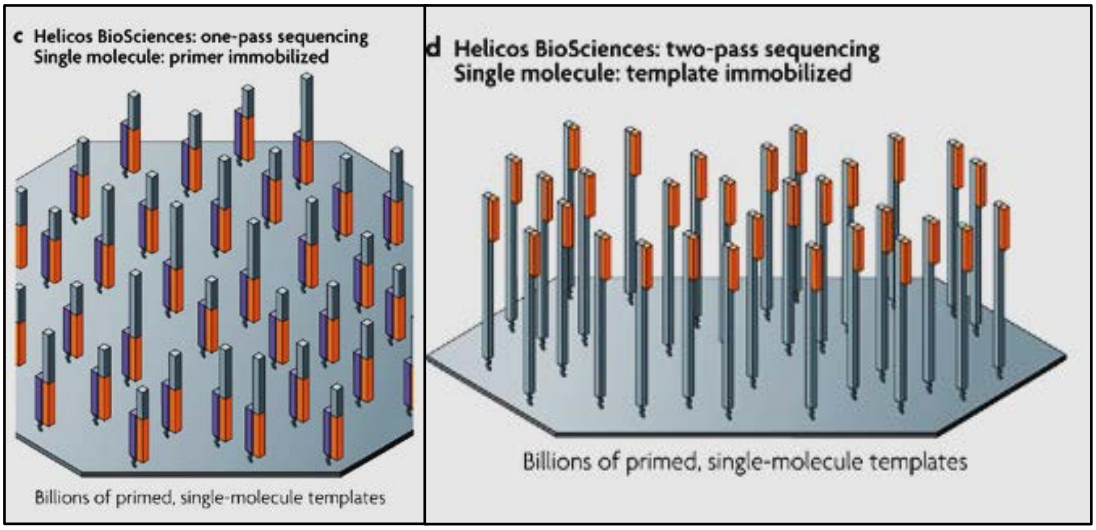


Figure 2.4. Single molecule template preparation for next generation sequencing technology. Adapted by permission from Macmillan Publishers Ltd, "Sequencing technologies - the next generation", by M.L.Metzker, 2010, Nature reviews.Genetics, 11, p.33, copyright 2010.

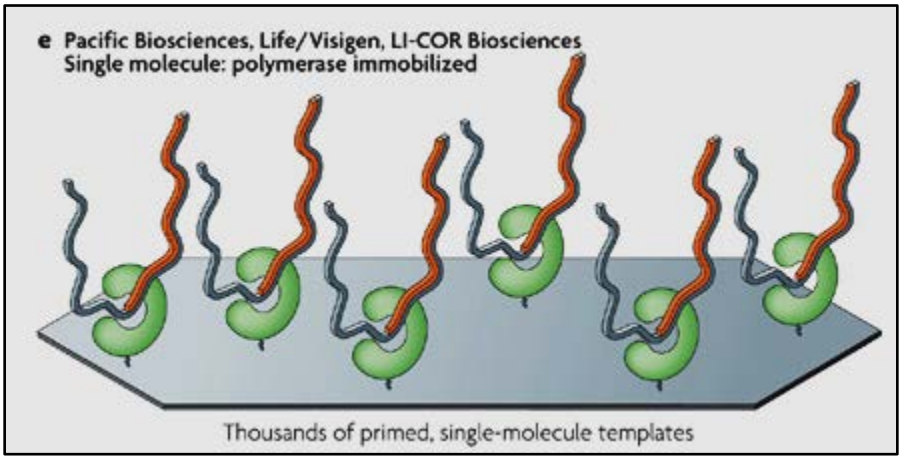


Figure 2.5. Real-time template preparation for next generation sequencing technology. Adapted by permission from Macmillan Publishers Ltd, "Sequencing technologies - the next generation", by M.L.Metzker, 2010, Nature reviews.Genetics, 11, p.33, copyright 2010.

The next step in the process is sequencing the single-stranded DNA and reading it using imaging of fluorescence probes attached to the nucleotides. Currently there are four methods used for sequencing (Metzker, 2010):

1. Cyclic reversible termination
2. Sequencing by ligation
3. Single-nucleotide addition/ pyrosequencing
4. Real-time sequencing

Cyclic reversible termination (Figure 2.6) uses the cyclical process of incorporating fluorescent nucleotides, imaging, and termination using ddNTPs. Each step of a single nucleotide incorporation, termination, imaging, and washing is repeated until the entire template is read. This is most commonly used in clonally amplified templates. The method relies on the use of modified ddNTPs with more efficient cleavage of the fluorescent labels compared to Sanger sequencing.

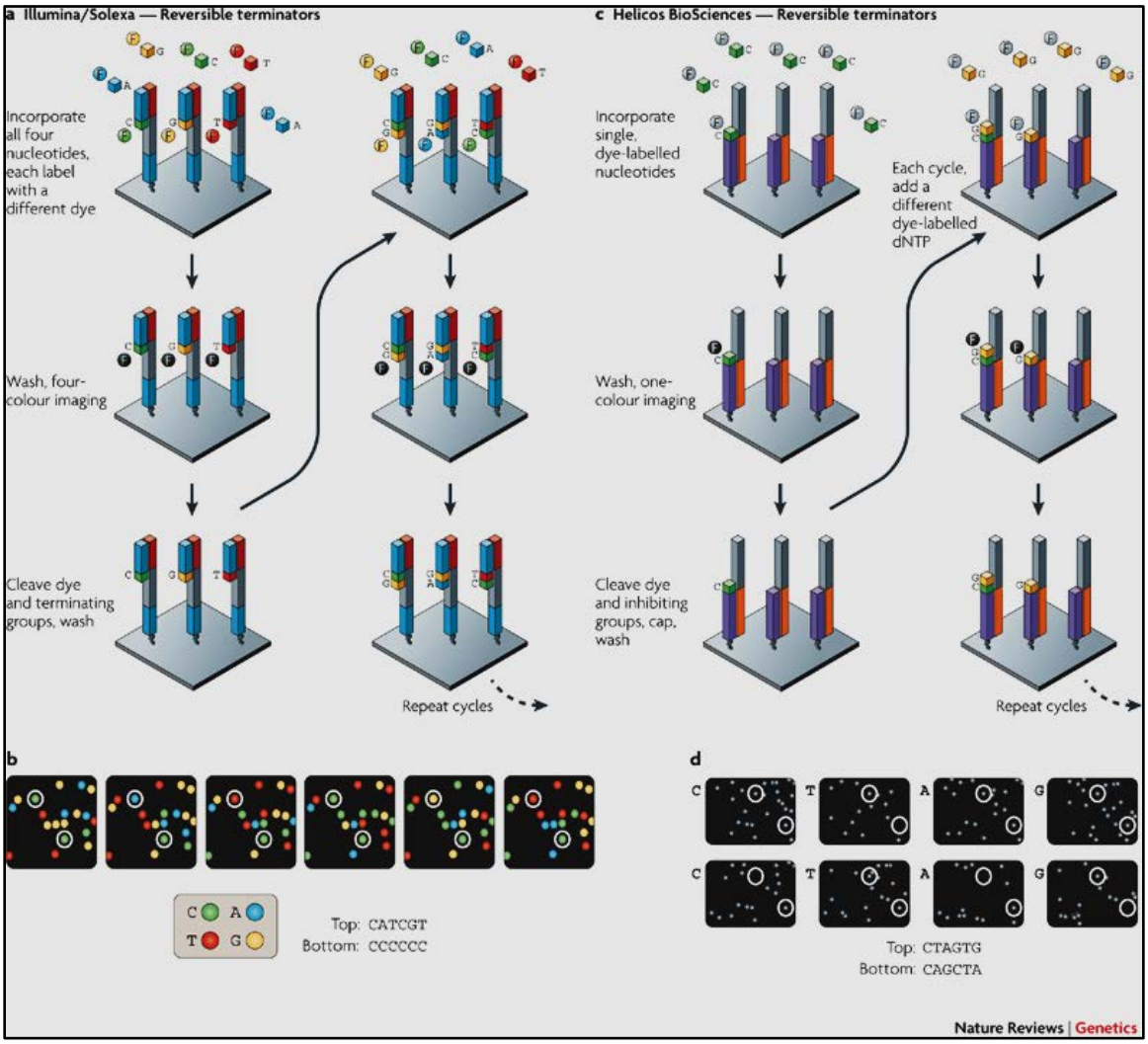


Figure 2.6. Cyclic reversible termination method in next generation sequencing technology. Adapted by permission from Macmillan Publishers Ltd, "Sequencing technologies - the next generation", by M.L.Metzker, 2010, Nature reviews.Genetics, 11, p.36, copyright 2010.

Sequencing by ligation uses DNA ligase[6] to attach to a fluorescent dye-labeled probe, washing extra probes away and read by imaging in a cyclical way (Figure 2.7). Pyrosequencing uses sulphurylase and luciferase to detect bioluminescence (Figure 2.7) instead of fluorescently labeled nucleotides. Real-time sequencing is the

[6] Ligase: cellular enzyme which catalyzes the formation of bonds between two DNA strands.

most advanced of these technologies not requiring any terminators to stop the process of sequencing. DNA polymerases are attached to the glass slides and therefore sequencing can be performed without the need to terminate and the release of fluorescence from the nucleotide read in real-time by imaging.

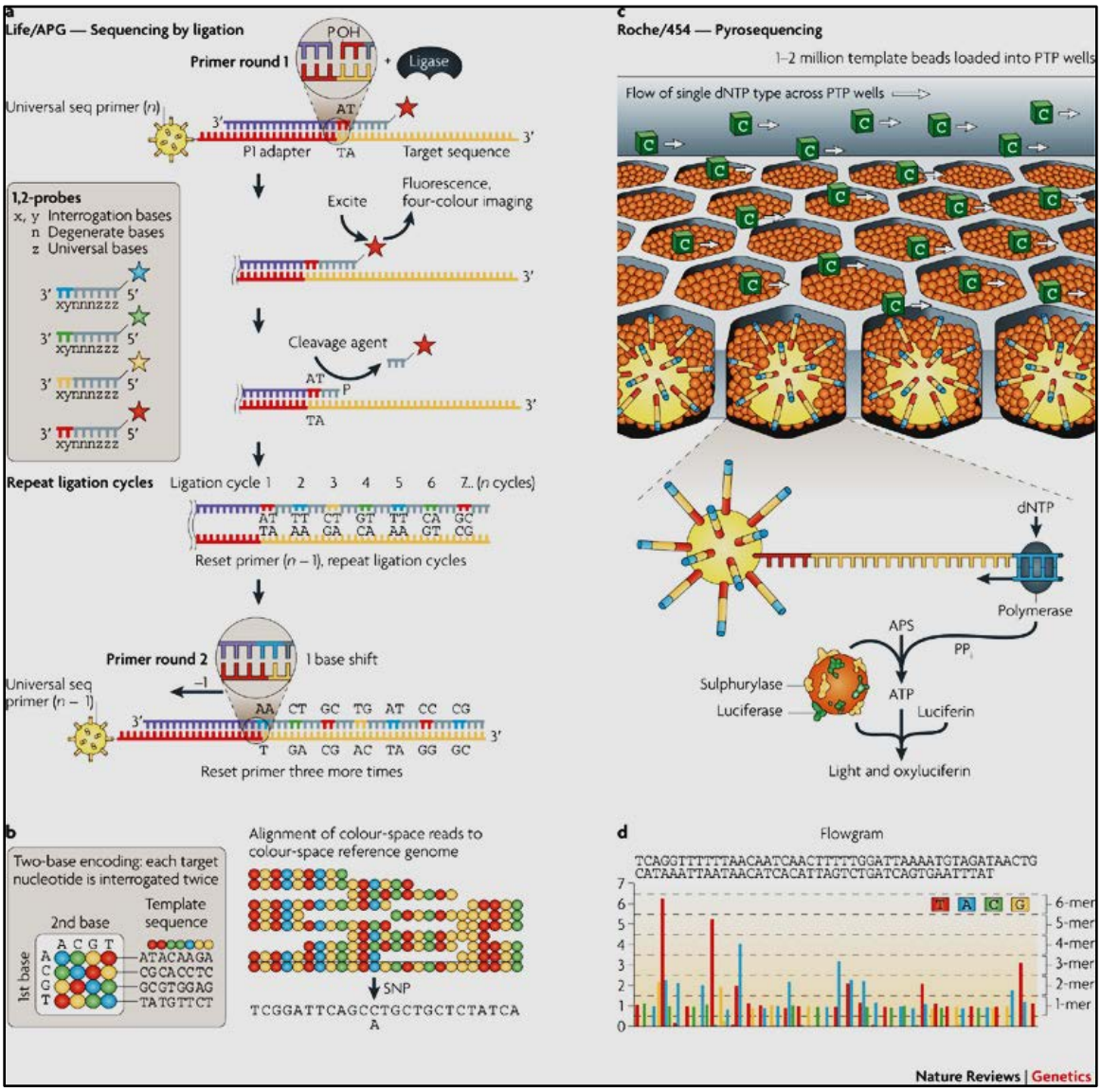


Figure 2.7. Sequencing by ligation and pyrosequencing method in next generation sequencing technology. Adapted by permission from Macmillan Publishers Ltd, “Sequencing technologies - the next generation”, by M.L.Metzker, 2010, Nature reviews.Genetics, 11, p.36, copyright 2010.

Technological advances will continue to improve these NGS technologies with the purpose of reducing sequencing errors. One of the first challenges of parallel sequencing was developing efficient algorithms for fast and accurate mapping of this data to the reference genome. Big data however brought with itself bioinformatics analysis challenges, the first being mapping the target genome to the reference.

2.2 Mapping Algorithms

The earliest efforts in algorithm development for mapping was done for discovering sequence homology that examines the relatedness of two sequences (protein-coding DNA sequences) in different species in order to understand its function (Pevsner, 2009). Understanding homology was important to define relatedness of proteins through evolution and define changes that occurred through speciation. The same principles were applied to DNA sequence mapping. Reads to be compared are placed along the x-axis and y-axis and scored +1 for each match, -2 for each mismatch and gap. This method was first developed by Needleman and Wunsch and is known as the Needleman-Wunsch algorithm (Needleman & Wunsch, 1970). At the heart of this algorithm is dynamic programming (Eddy, 2004) which starts with laying out the matrix, deciding on scoring method and finally recursively finding the best path with the best optimal score. This algorithm is dynamic because it finds the best optimal score and keeps it in memory in order to avoid recalculation of scores that happens in a recursive process (Figure 2.8).

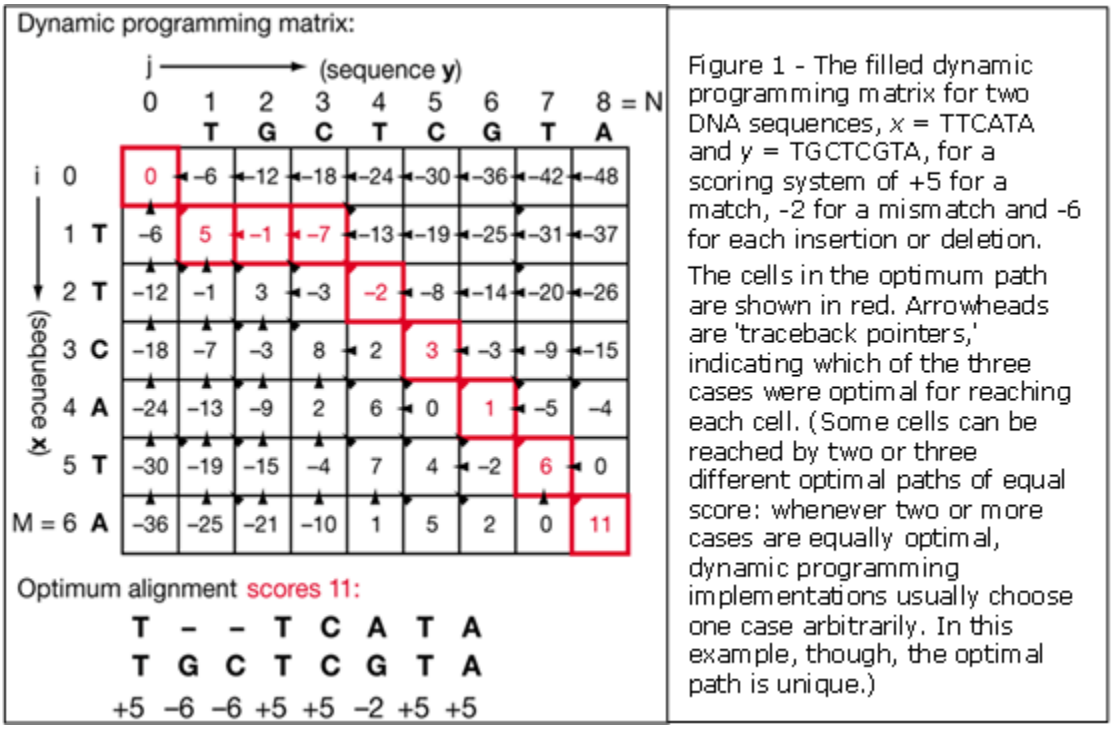


Figure 2.8. Dynamic programming matrix. Reprinted by permission from Macmillan Publishers Ltd, "What is dynamic programming", by S.R.Eddy, 2004, Nature biotechnology, 22, p.909, copyright 2004.

The Needleman-Wunsch algorithm was used for global alignment and when adapted to local alignment as done by Smith-Waterman algorithm (Smith, Waterman, & Fitch, 1981), proved even more useful in finding DNA sequences of local identity (Pevsner, 2009). The Smith-Waterman method uses one more row (m+1) and column (n+1) for two sequences of length m and n and each subsequent score in the matrix is incremented from the value in the preceding diagonal cell, with a match getting a score of +1, -0.3 for a mismatch, and -1.3 for a gap. The Smith-Waterman algorithm does not allow for any negative scores. The highest score in the matrix signifies the end of alignment. This method allows for getting local alignments from a long sequence instead of global alignment of Needleman-Wunsch that is computationally intensive due to its recursive calculations. Local alignment is often

used for quick database searches and serves as the foundation for most population database search mapping tools like BLAST and BLAT.

2.3 Structural Variant Detection Tools

Tuzun *et al.* were the first to describe a computational framework for structural variant (SV) detection (Tuzun et al., 2005). The most gain in information while defining SVs comes from alignment of paired end reads *i.e*. single fragment of DNA read from both ends with an unread segment in between. The length of the original fragment is known and can further be empirically determined based on the insert size distribution of the experiment. Their approach is still the basic framework for SV detection and is as follows: 1) define probability distribution of insert sizes (length of fragment of DNA sequenced), 2) define discordant reads as those which lie at least 2-4 standard deviations outside this distribution, 3) cluster all reads that support the same SV such that at least 2 discordant reads identify the same SV, 4) identify the SV at the location using percent identity of the discordant reads and number of supporting reads.

Lee *et al.* proposed a more robust probabilistic method for SV detection (S. Lee et al., 2008). The key methodology proposed was:

1. Defining a probabilistic framework: All reads generated (paired end reads: short fragments of DNA sequenced from both sides with an area of un-sequenced portion in between) are assumed to be independent of each other. The set of same type of structural variants is represented by C. Using the independence assumption each mapped read *A* and *B* which are part of the same cluster and therefore explaining the same SV will have a joined probability of belonging to the same cluster *C* defined as $P(A,B|C)=P(A|C).P(B|C)$. The mean insert size length *s* is known from library generation for

sequencing experiment and also can be derived computationally from experiment output. The probability distribution gives the standard deviation of insert size. The signature for insertions and deletions is as follows: a length of *r* represents the length of insertion or deletion and *s* represents the correct mapping length of the insert in the reference. Therefore an insertion event is *s+r* and deletion event is *s-r*. Using this signature we define the probability of read *A* and read B being in the insert size distribution of *s+r* (or *s-r)* paired-end reads by using a maximizing function of the joint probability of A and B belonging to the same variant cluster. It thus calculated the probability that a read is produced from the genome sequenced. Similarly, in defining the signature for inversion and translocation we determine the joint probability of observing the 2 mate pairs in the same cluster. The higher this probability the more likely a read belongs to the cluster.

2. Defining the SV: The structural variant is defined using three features
    i. Percent similarity of the mate pair sequence to the reference
    ii. Product of the probabilities calculated above for a cluster. The larger the probability of a cluster the more it is reliable.
    iii. Number of mate pairs in a cluster

Each SV can map to various clusters and therefore using the above three features we need to find the configuration which maximizes each of the features mentioned above. Using this configuration we can assign each SV to just a single cluster.

The primary challenge of the probabilistic model is that the distributions are based on insert sizes and are dependent on intrinsic information output by the mapping tool. If the mapping tool does not know where to place the read, the read can be either disregarded or placed randomly based on all possible best matches. Lee *et al.*

assigned confidence scores (p-values) to insertion/deletion signatures, but this could not be done to inversions and translocations. If there were systematic errors in sequencing or in alignment algorithms, these will not be picked up using the probability calculation for inversions and translocations and will therefore be ignored. However with the current NGS platforms these errors have been considerably reduced due to higher coverage. Current tools built up on this method with SVDetect, BreakDancer and CREST discussed below.

The BreakDancerMax (K. Chen et al., 2009) algorithm uses the distance and alignment of paired end reads along with the distribution of reads which map to a certain location on the reference to define a SV. Using the SV signature, the program then searches for regions on the reference genome which anchor more SV's than expected. These regions form the putative breakpoints. For a particular region, whichever is the 'dominant' SV signature is used to identify the SV at the location *i.e.* clustering of all SV's for a particular location and choosing the SV. Further, it uses a confidence score to assign the probability of observing a SV at the location which is higher than chance using a chi-squared statistic with the cutoff p-value <0.0001. This tool assumes a Poisson distribution for the clustered variants. Using an analytical model for detecting true positive SV rate in a simulated dataset, they estimated that with an average insert size of 200 bps, insertions and deletions shorter than 40 bps would be difficult to detect.

BreakDancerMini tries to overcome the dependence on the insert size by using a sliding window test. The algorithm defines a window for the reference that is the mean of the insert library of a confidently mapped region+3 s.d - 2(average length of read). Using this window, the frame is shifted 1 bp at a time and the probability that the read lies in this window is calculated. Once again this method works well for insertions and deletions. BreakDancer Max and Mini together did a better job at indel

detection (experimentally validated 110 of 167 indels called by BreakDancer) and to some extent inversions (validated 4 of 13 inversions) but did not do so well with intra-chromosomal events (validated 2 of 6 intra-chromosomal translocations). This is primarily because it uses only paired end data where both read ends map within a defined distance. If only one end of the pair maps then this read is not considered. Translocations will show this pattern where one end of a paired end read mapped to one chromosome and the other end mapped on another chromosome. This information may be lost in the alignment process as the aligner aligned just one end of the read and information on the paired end is lost. Therefore, this tool is most likely to miss this inter-chromosomal event. Thus BreakDancer performs well with indels but does not have a very strong method for detecting translocations.

SVDetect (Zeitouni et al., 2010) on the other hand uses a simplified strategy for clustering which is non-probabilistic. Clusters are formed from paired end reads which had incorrect distance (2-3 s.d.) and/or orientation from the reference genome. SVDetect then divides the reference genome into overlapping windows of fixed size and groups all the paired end reads which map to the same overlapping reference. SVDetect also uses clustering parameters like minimum number of reads supporting a SV, filtering those reads whose orientation is different from the majority in a cluster. The filtering process is user defined and the thresholds for filtering can be changed.

Another important feature added to SVDetect is the ability to predict copy number variations/duplication events. The algorithm does this by finding the ratio of depth of coverage of sample to a control dataset in a sliding window along the genome, though they do not define the length of the window. However, the detection of duplications needs a control dataset that has not been well defined in the paper. Thus SVDetect is able to identify insertions, deletions, inter-chromosomal events,

duplications and translocations using a clustering pattern and filtering process. However, SVDetect will also miss reads which were the alignment tool failed to assign a read unambiguously.

CREST (Wang et al., 2011), unlike other applications, does not use the concept of discordant paired reads. Instead it uses direct mapping of reads to the reference to identify the breakpoints. The Burrow Wheeler alignment tool soft clips reads during alignment, *i.e.* when there is partial alignment or when one end of the pair aligns perfectly and the other end is partially aligned the tool clips the sequence at the point of partial alignment. CREST collects all these soft clipped reads together, creates longer contigs of these reads using an assembly algorithm (CAP3) and realigns these contigs using BLAT (**B**LAST **L**ike **A**lignment **T**ool) alignment algorithm. Thus, the identification of first soft clipped region defines the first breakpoint at location 1. All the soft clipped reads are collected and assembled into contigs using CAP3 algorithm and aligned using BLAT. Wherever these soft clipped contigs map to the genome is the second breakpoint, which will have soft clipped regions mapping to location 1. CREST requires the second contig generated from soft clipped regions at location 1 to be within a certain distance (user-defined) of the second location.

The final call of variants includes only those reads with >97% sequence similarity and a BLAT score >30 and defines the probability of observing at least *c* soft clipped reads with sequence coverage *C* (at that location) to be less than or equal to 0.05

The signatures defined by CREST use the same basic signature of all the SVs but do not use the mapping distance of 2-3 s.d. It uses direct alignment information and therefore is able to identify the SVs at breakpoint coordinate level. In experiments comparing the platform it performed better than BreakDancer in detecting insertions, deletions, and inversions. CREST is more apt for inter-chromosomal event detection and therefore performs better in translocation events seen in cancer genomes.

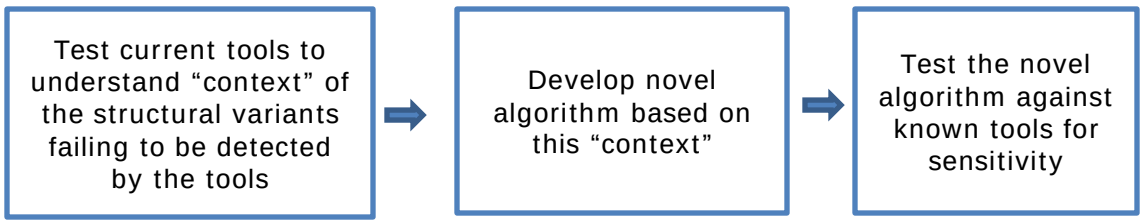
However due to the alignment heavy algorithm, it performs badly in regions of repeats as the tool will perform only as good as the assembly/alignment tool.

All of the above methods use the mapping tool output as the input and derive possible anomalous reads based on either the insert size distribution or flags set by the mapping tool. Thus the ability of the tools to detect variants depends entirely on the robustness of the alignment. In regions where the aligner cannot reliably map sequence, such as repetitive regions, it is assumed that the ability of the tools to detect variants is compromised. This study aimed to test this theory and to design a more effective approach to variant detection by utilizing the known biology driving variant formation, focusing specifically on translocations.

3. TESTING CONTEXT: REPEAT STRUCTURE OF GENOME

3.1 Overview

The overall approach for this study was three pronged: first, test the currently available tools against an experimentally validated dataset; second, develop an algorithm for translocation detection based on known biological information; and finally test our algorithm against the known tools. The first step was essentially to identify and define the reason why tools would fail to detect experimentally validated variants. Our hypothesis was that the tools would fail to recognize variants that were in genomic regions of repeat due to low complexity of these regions and therefore the inability of the aligner to map these regions uniquely. Thus stated differently, variants in repeat regions were less likely to be detected by the tool compared to variants in unique regions.



3.2 Testing Current Tools

The first step for proof of concept was to test the available structural variant detection tool on an experimentally validated dataset. The 1000-Genomes project was designed to use extensive sequencing of many individuals around the world with different ethnicities in order to characterize all types of variants found in these individuals and to relate it to the phenotype (1000 genomes project). This project is a major international collaboration between universities around the world providing samples, sequencing data, and bioinformatics analysis in order to map the entire spectrum of genetic variation in the human population. The data has been made

publicly available on the website (www.1000genomes.org) (1000 Genomes Project Consortium, Abecasis, Altshuler, Auton, Brooks, Durbin, Gibbs, Hurles, & McVean, 2010b). The project had low-coverage whole genome sequencing, exome sequencing, and high-coverage sequencing of trio-subjects (mother-father-child) from different ethnic populations around the world. For purpose of testing we used deep sequencing data of the trio subject from Nigeria (YRI: NA19238, NA19239, NA19240).

3.2.1 FASTQ file format

Most sequencing machine output the data in a widely accepted format known as the Sanger-FASTQ (Cock, Fields, Goto, Heuer, & Rice, 2010). This is a text file with information about the sequence read and quality score of each base of the sequence read. Each read starts with a '@' followed by the identifier and description of the sequence which may be platform specific. The next line is a string of 'ATCGs', which is the actual read from the sequencing machine. The next line is a + sign which may be followed by a repeat of the sequence identifier and description line. The last line represents quality scores for each of the bases in the read sequence (Figure 3.1).

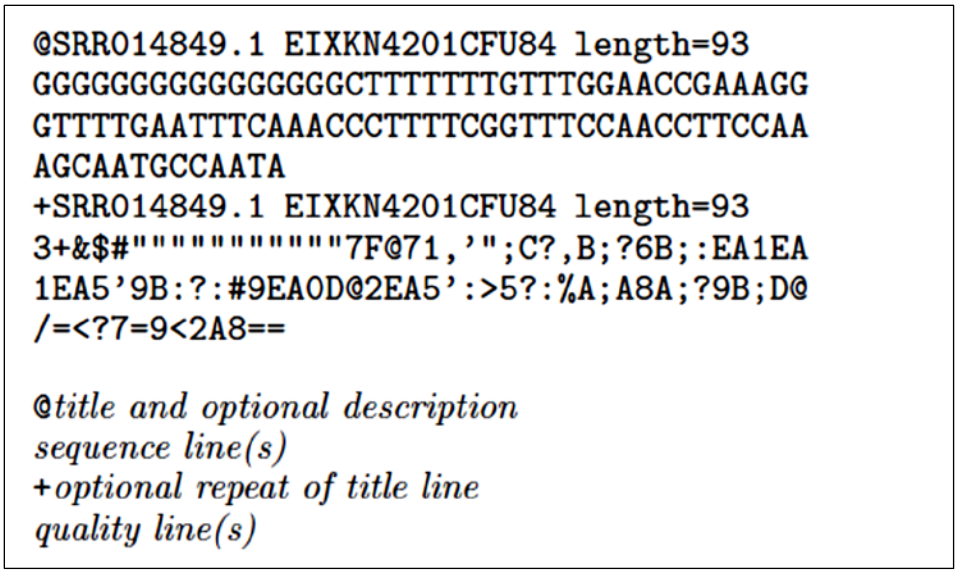


Figure 3.1. Sanger FASTQ format. Reprinted by permission from Oxford University Press, "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants", by P.J.Cock et al., 2010, Nucleic Acids Research, 38, p.1769, copyright 2010.

The last line equates quality values for each base read by the sequencing machine. This is a PHRED-based quality score converted to ASCII characters. PHRED was a computation tool developed by Ewing *et al*. (Ewing & Green, 1998; Ewing, Hillier, Wendl, & Green, 1998) to automatically assign quality values to sequencing trace files such as chromatograms that are generated by the sequencing machines. This is defined in the equation below.

$$q = -10 \times \log_{10}(p) \text{ ----------(Ewing } et\ al.)$$

q= quality score of the base

p=estimated error probability for the base

According to this formula, a base having a 1/1000 probability of being erroneous will have a quality score *q* of 30. The lower the probability score, the higher the quality.

3.2.2 1000-Genomes data analysis

The files obtained from 1000-Genomes website were in FASTQ format. This study used Yoruba subject data (YRI: NA19238, NA19239, NA19240). The FASTQ files were downloaded from ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/pilot_data/data/.

Datasets for each subject had sequencing files of various insert sizes. To maintain consistency we only used deep sequencing high coverage paired end library of insert size 260. Briefly, experiment ID-SRX001106 with 8 billion reads was used for NA19238, experiment ID-SRX000654 with 7.4 billion reads was used for NA19239 and experiment ID-SRX001102 with 6.1 billion reads was used for NA19240. Appendix A lists the files used for this analysis.

Sequencing reads from these files listed in Appendix A Table 1.1 were aligned to the reference genome NCBI build 36.1/ UCSC hg18. The reference human genome is assembled and maintained by the Genome Reference Consortium (GRC) (http://www.ncbi.nlm.nih.gov/projects/genome/assembly/grc/human/) which includes NCBI, Wellcome Trust, Sanger Institute, Genome Institute at Washington University and European Bioinformatics Institute (EBI) at NCBI. The complete genome and chromosome level FASTA files are stored at UCSC website which can be downloaded at (http://hgdownload.cse.ucsc.edu/goldenPath/hg18/chromosomes). The GRC, which is an extension of the Human Genome project (http://www.genome.gov/10001772), produces overlapping segments of high quality, longer length DNA sequence from a group of volunteers. Using in-house tools they release a comprehensive and rigorous representative of a consensus normal human reference genome for use in the public domain. The builds are constantly getting updated as more data becomes available on previously un-sequenced regions of the human genome or improved on currently sequences regions.

Data for these various builds are stored in the FASTA format, which is a text file with a header line starting with '>' containing information about the sequence followed on the next line by the longest continuous available sequence data as shown in Figure 3.2. Standard codes to represent amino acid sequences and nucleotides are used and can also have lower case letters representing the same nucleotide/amino acid.

```
>gi|12345|ref|NM_12345.01|Homo sapiens XYZ
ATTTCGATTAATCGAGAAAAAAAATATTTTAGGGGGCCATTTATATACCCCCCCC
TACACCCACAC
```

Figure 3.2. FASTA format example. First line starts with ">" followed by information about the sequence and next line contains actual sequence

Fastq files for the same insert size and same library were downloaded from the ftp site[7] for each of the trio subjects; NA19238, NA19239, NA19240. These were aligned to the reference genome UCSC hg18 using a popular tool for short read alignment, BWA-0.5.9 (Li & Durbin, 2009).[8] Briefly, BWA uses the Burrow-Wheeler transformation algorithm to compresses the data such that repetitive information is stored in a compressed format in prefix and suffix arrays in order to perform searches on the entire human genome. The output of the mapping tool is in SAM format described in Section 4.2.1. SAM is then converted to BAM, which is the binary representation of a SAM file using 'samtools' suite.[9]

The output from BWA mapping was used to further detect structural variants using an open-source toolkit. SVDetect (Zeitouni et al., 2010) is one such toolkit which uses paired-end mapping data to identify reads which occur at a distance greater than expected insert size and/or are in the incorrect orientation with respect to each

[7] 1000 genomes data download:ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/data/
[8] BWA sourceforge page: http://bio-bwa.sourceforge.net/bwa.shtml
[9] SAMTools sourceforge page: http://samtools.sourceforge.net/

other. It then breaks the genome to create overlapping windows. Each anomalous paired-end read can anchor to these windows such that the two windows where the two paired-end reads map form a link. A filtering process removes links that has less than a certain number of reads supporting a link and other features defined in the tool. Structural variants are called based on mapping signatures for a particular type of variant. This tool has a tendency to output many false positives as it only uses a clustering method to call variants. Thus, any variant with 2 or more reads supporting it is called as a variant without using the underlying variants distribution to calculate confidence scores of calling a variant as done in BreakDancer. We therefore used this tool which reports all possible variants in order to test our hypothesis that tools fail to detect variants because these variants lie in the repeat regions.

Experimentally validated structural variants detected in the same trio subjects were used as the validation standard. The variant dataset was downloaded from the ftp site.[10]

Variants were detected computationally using various algorithms developed in-house by 1000-Genomes team (1000 Genomes Project Consortium, Abecasis, Altshuler, Auton, Brooks, Durbin, Gibbs, Hurles, & McVean, 2010a). The variants were defined as mobile element insertions, tandem duplication, deletions, and novel sequences. We extracted only those variants from the variant files that were experimentally validated as indicated in the *Description* field of the file.

[10] 1000-Genomes structural variation data page: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/pilot_data/paper_data_sets/a_map_of_human_variation/trio/sv/

3.2.3 SVDetect Analysis

SVDetect defines its set of variants as described in the Table 3.1.

Table 3.1: SVDetect defined structural variants

NORMAL_SENSE	Correct ends orientation using <mates_orientation> as reference
REVERSE_SENSE	One of the ends has an incorrect orientation
DELETION	Deletion (NORMAL_SENSE & mean insert size > μ+threshold*σ)
INSERTION	Insertion (NORMAL_SENSE & mean insert size < μ-threshold*σ)
INVERSION	Inversion (REVERSE_SENSE)
INV_FRAGMT	Inversion of a genomic fragment, defined by balanced signatures (BAL)
INS_FRAGMT	Insertion of a genomic fragment, defined by balanced signatures (BAL)
INV_INS_FRAGMT	Inverted INS_FRAGMT (BAL)
LARGE_DUPLI	Large duplication
DUPLICATION	Duplication, medium size
SMALL_DUPLI	Small duplication (mean insert size < μ-threshold*σ & overlap between subgroups)
INV_DUPLI	Inverted duplication (REVERSE_SENSE & mean insert size < μ-threshold*σ & UNBAL)
TRANSLOC	Translocation
INV_TRANSLOC	Inverted translocation
COAMPLICON	Co-amplicons, two different fragments repeated in the same strand sense (BAL)
INV_COAMPLICON	Inverted co-amplicons, two different fragments repeated in the opposite strand sense (BAL)
SINGLETON	Singleton (mean insert size < μ-threshold*σ), for Illumina mate-pairs only
UNDEFINED	Undefined inter/intra-chromosomal SV type

The purpose of this analysis was to see if the variants detected by SVDetect had some overlap in the same genomic region as those in the 1000-Genomes experimentally-validated variant dataset. To further characterize these variants, we looked for overlap of the variants with mapped repeat structure in the human genome (Smit et al., 2014)[11]. We used the RepeatMasker mapped repeat regions on the hg18 build human genome to identify the repeat regions in the human genome. The repeat elements in RepeatMasker database have been classified as LINEs: Long

[11] RepeatMasker webpage: http://www.repeatmasker.org

interspersed nuclear elements; SINEs: Short interspersed nuclear elements; LTR: Long terminal repeats; DNA repeats and other.

3.3 Extracting Overlapping Regions

SVDetect-0.7f was used on the mapped BAM files to detect variants. Each variant file from 1000-Genomes classified as 'Deletion', 'Novel Sequences', 'Mobile Element Insertion' and 'Tandem Duplications' was used separately to extract overlapping variants detected by SVDetect. 1000-Genomes variant files did not have subject identifier in file and thus had structural variants detected in all three trio subjects combined. SVDetect was run on each BAM file separately and then combined. We were only looking for regions of the genome that showed an overlap irrespective of the type of variant as defined in SVDetect. Overlap was defined as at least 10 percent of the insert size. In this case the insert size library chosen for download was 260 and therefore 10 percent overlap was at least 26 basepairs on either side of the read as represented in the Figure 3.3. Ten percent of insert size was used as an arbitrary cutoff.

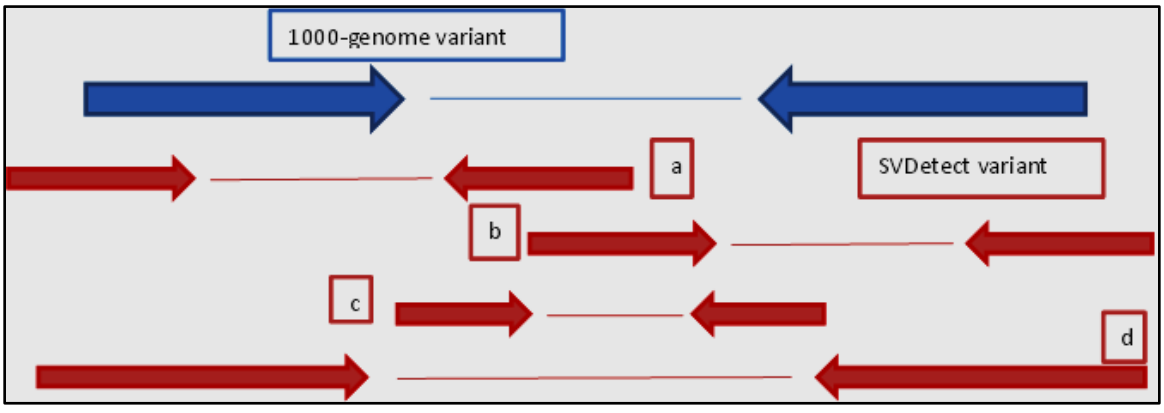


Figure 3.3: Overlap definition for SVDetect; a: left overlap; b: right overlap; c: SVDetect variant completely within; d: 1000-genome variant completely within

A perl script was used to do the overlap extraction. Similarly, since the purpose was to define the context of the genomic region that is causing the tools to fail, RepeatMasker mapped repeat regions was used to do a similar overlap of with 1000-genome variants. However in order to identify the overlapping regions with repeat structure within, the entire repeat region had to be present within the variant as shown in Figure 1.1.

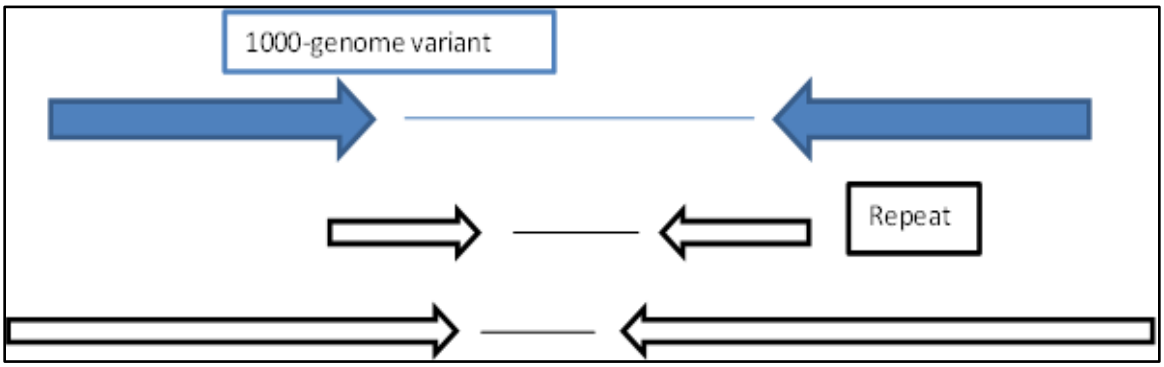


Figure 3.4: Overlap definition for repeat; Blue arrows: 1000 genomes, Black arrows: repeat elements in RepeatMasker.

3.3.1 Results: SVDetect performance

The result from extraction analysis showed that SVDetect did poorly in detecting insertion elements (17%) while it did very well in identifying validated novel sequences high in detection percent (67%) (Table 3.2).

Table 3.2: Overlap results for SVDetect

Type of Event	1000-Genomes events	Events detected by SV-Detect: True Positive	Events not detected by SV-Detect: False Negative
Deletion	9695	4657 (48%)	5038
Mobile Element Insertion	492	85 (17%)	407
Tandem Duplication	65	23 (35%)	42
Novel Sequences	66	42 (67%)	24

SVDetect is known to give many false positives and thus is a highly sensitive tool for detection of structural variants. However, even with such a highly sensitive tool, less than 50% of the variants detected by SVDetect were in the same region as deletions events identified by 1000-Genomes. SVDetect was in the true variants regions anywhere from 17 to 67 percent of the time depending on the type of variant. Even a highly sensitive tool was clearly not able to detect many of the validated variants.

3.3.2 Understanding context: Repeat

We therefore tried to understand if the underlying repeat structure within these variants was driving the ability of the tool to detect a variant. If a variant is in a repeat region, the mapping tool will not be able to definitively place the read representing the variant in a specific region due to low complexity of the repeat region. We hypothesized that validated variants that were not detected by the tool were more likely to have repeat elements within them than variants that were detected by the tool. As shown in Figure 3.4, we extracted those reads that had the

repeat element completely contained within the variants or variant completely contained within the repeat. The results are shown in Table 3.3, 3.4.

Table 3.3: Repeat elements within undetected events

Type of Event	1000-Genomes events	Events not detected by SV-Detect: False-Negatives (a)	Events with repeat structure within (b) (a/b%)
Deletion	9695	5038	2678 (53%)
Mobile Element Insertion	492	407	134 (33%)
Tandem Duplication	65	42	11 (26%)
Novel Sequences	66	24	13 (54%)

Table 3.4: Repeat elements within detected events

Type of Event	1000-Genomes events	Events detected by SV-Detect: True-Positives(a)	Events with any repeat structure within (b) (a/b%)
Deletion	9695	4657	3754 (81%)
Mobile Element Insertion	492	85	36 (42%)
Tandem Duplication	65	23	13 (57%)
Novel Sequences	66	42	26 (62%)

Although 53% of deletion events that were not detected by the tool showed some form of repeat structure, 81% of deletion events which were detected also showed a repeat structure. A similar pattern was seen with all other variant types where the events which were detected by the tool had a greater percent of repeat elements compared with those which were not detected by the tool. A chi-squared analysis (Tables 3.5, 3.6, 3.7, 3.8) showed a significant association between repeat structure and the ability of the tool to detect the variant for deletions and tandem duplication events but not significant for novel sequences and mobile element insertion events at $p<0.05$ level. Thus if there was a repeat structure within the variant, the tool was

more likely to detect it, which was contrary to our hypothesis. We therefore failed to reject the null of no association between repeat structure and ability of the tool to detect the variant.

Table 3.5: Chi-squared test: Relation of deletion events with repeat

Deletion Events	Repeat structure within	No Repeat structure within	Total
Detected by tool (True positives)	3754	903	4657
Not detected by tool (False Negatives)	2678	2360	5038

Note: Table tests if ability of the tool to detect deletion events is influenced by the repeat structure within it. The Chi-square statistic is 816.873 with p-value of 0. This result is significant at $p < 0.05$.

Table 3.6: Chi-squared test: Relation of insertion events with repeat

Insertion Events	Repeat structure within	No Repeat structure within	Total
Detected by tool (True positives)	36	49	85
Not detected by tool (False Negatives)	134	273	407

Note: Table tests if ability of the tool to detect insertion events is influenced by the repeat structure within it The Chi-square statistic is 2.764 with p-value of $p=0.096$. This result is not significant at $p<0.05$.

Table 3.7: Chi-squared test: Relation of duplication events with repeat

Duplication Events	Repeat structure within	No Repeat structure within	Total
Detected by tool (True positives)	13	10	23
Not detected by tool (False Negatives)	11	31	42

Note: Table tests if ability of the tool to detect duplication events is influenced by the repeat structure within it The Chi-square statistic is 5.871 with p-value of 0.015. This result is significant at $p<0.05$.

Table 3.8: Chi-squared test: Relation of novel sequences events with repeat

Novel Sequence Events	Repeat structure within	No Repeat structure within	Total
Detected by tool (True positives)	26	16	42
Not detected by tool (False Negatives)	13	11	24

Note: Table tests if ability of the tool to detect novel sequences events is influenced by the repeat structure within it The Chi-square statistic is 0.378 with p-value of 0.538. This result is not significant at $p<0.05$.

Since defining genomic context was the goal of the analysis, we further broke down the detected and undetected events by type of repeat structure as classified by RepeatMasker. Note, however, that the same variants can have more than one type of repeat structure. This breakdown also showed similar results. While LINEs and SINEs were more abundant of the repeat structure, undetected events (false-negatives) showed a lower percentage of these events than detected events (true-positives) (Tables 3.9, 3.10, 3.11, 3.12).

Table 3.9: Repeats structure by type: Deletion

1000 genome events	DNA_repeats		LINEs		SINEs		LTR		Other	
	Yes	No	Yes	No	Yes	No	yes	No	Yes	No
Un-detected (5038)	245 (5%)	4793	825 (16%)	4213	1096 (22%)	3942	423 (8%)	4615	1113 (22%)	3925
Detected (4657)	1560 (34%)	3097	2480 (53%)	2177	2710 (58%)	1947	1954 (42%)	2703	2549 (55%)	2108

Note: LINEs: Long interspersed nuclear elements; SINEs: Short interspersed nuclear elements; LTR: Long terminal repeats; Undetected: False negative events; Detected: True positive events (1000-Genomes event)

Table 3.10: Repeats structure by type: Mobile element insertion

1000 genome events	DNA_repeats		LINEs		SINEs		LTR		Other	
	Yes	No	Yes	No	Yes	No	yes	No	Yes	No
Undetected (407)	11 (3%)	396	69 (17%)	338	20 (5%)	387	24 (6%)	383	31 (8%)	376
Detected (85)	12 (14%)	73	15 (18%)	70	18 (21%)	67	12 (14%)	73	14 (16%)	71

Note: LINEs: Long interspersed nuclear elements; SINEs: Short interspersed nuclear elements; LTR: Long terminal repeats; Undetected: False negative events; Detected: True positive events (1000-Genomes event)

Table 3.11: Repeats structure by type: Tandem duplication

1000 genome events	DNA_repeats		LINES		SINES		LTR		Other	
	Yes	No	Yes	No	Yes	No	yes	No	Yes	No
Undetected (24)	1 (4%)	23	3 (13%)	21	6 (25%)	18	2 (8%)	22	5 (21%)	19
Detected (42)	3 (7%)	39	7 (17%)	35	9 (21%)	33	4 (10%)	38	13 (31%)	29

Note: LINEs: Long interspersed nuclear elements; SINEs: Short interspersed nuclear elements; LTR: Long terminal repeats; Undetected: False negative events; Detected: True positive events (1000-Genomes event)

Table 3.12: Repeats structure by type: Novel sequences

1000 genome events	DNA_repeats		LINES		SINES		LTR		Other	
	Yes	No	Yes	No	Yes	No	yes	No	Yes	No
Undetected (42)	1 (2%)	41	5 (12%)	37	2 (5%)	40	3 (7%)	39	1 (2%)	41
Detected (23)	6 (26%)	17	10 (43%)	13	8 (34%)	15	9 (39%)	14	9 (39%)	14

Note: LINEs: Long interspersed nuclear elements; SINEs: Short interspersed nuclear elements; LTR: Long terminal repeats; Undetected: False negative events; Detected: True positive events (1000-Genomes event)

Since presence of repeat elements was not driving the ability of the tool to detect variants, we decided that we had to be even more specific in defining the “context” of these variants. Repeat element in general, (including every type of repeat) within or around these variants was not the “context” causing these variants to go undetected.

Our goal was to design a tool to detect translocations, and thus the design step started from defining the context for translocations occurring in the genome *i.e.* understand how translocations occur in the genome (Bunting & Nussenzweig, 2013) and build the context using this information. When we looked at the overlap between the NAHR regions and the genomic location of the variants missed by SVDetect, we observed a pileup in the regions where the tools failed to detect variants and the NAHR regions (Figure 3.5). Thus, we used NAHR as one part of the biological

context for the novel algorithm. However, there are multiple substrates for NAHR in the human genome. The likelihood of any two regions of NAHR taking part in exchange of genomic material (translocation) depends in part on the probability of these regions being in close proximity in 3-dimensional space. Therefore, the second part of our biological context was to incorporate information about the distribution of the genome in 3-dimensional space as derived from Hi-C data.

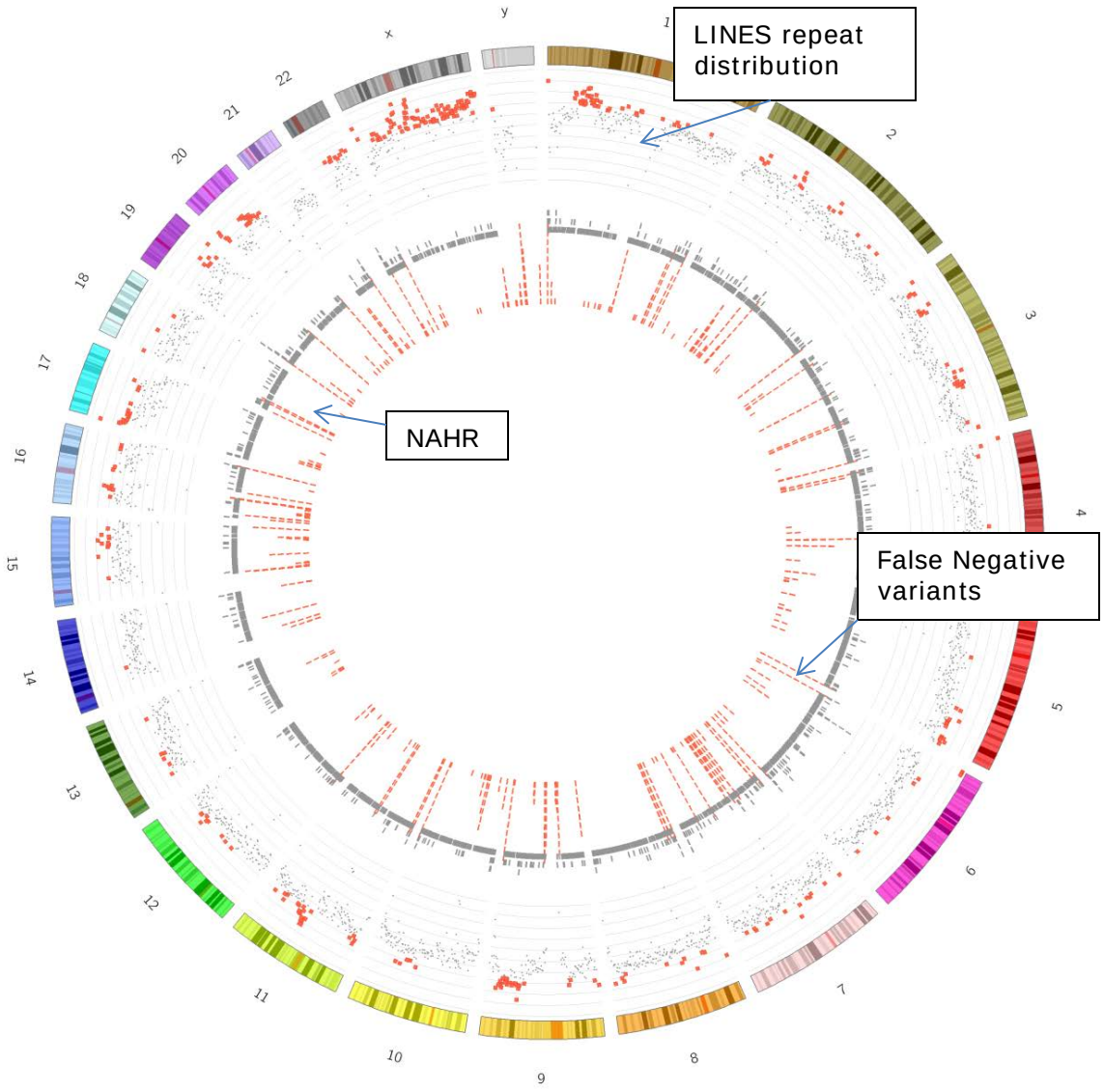


Figure 3.5: NAHR overlap with 1000-Genomes variants. Pile-up of false negative deletion ebents (middle grey bar plot) shows concordant spike with pile-up of NAHR (non-allelic homologous recombination) regions (inner red bar plot). Outermost grey and red dot plot shows count of number of LINE elements (long interspersed nuclear elements) in 1Kbps windows.

4. ALGORITHM DESIGN AND DEVELOPMENT

4.1 A Context Definition for Translocations

The purpose of defining “context” for detecting translocation in the genome was to effectively use information on mechanism of translocation formation in designing the tool. Earlier held belief that cancer occurs due to random genomic events is being proven ineffective due to studies which show non-random patterns to breakage sites, also called ‘hotspots’ (Jeffreys, Kauppi, & Neumann, 2001). Recombination event is a process by the cellular machinery to create diversity through evolution. These events occur through double stranded breaks and the aberrant repair of these breakpoints leads to structural variations. However these breakpoint regions have specific signatures of low copy number variations which share significant homology (Colnaghi, Carpenter, Volker, & O'Driscoll, 2011b). The breakpoint acting as substrates for anomalous repair (non-allelic homologous recombination, NAHR) lead to translocations and cause phenotypic changes (Deininger & Batzer, 1999), (Ou et al., 2011), (Gu, Zhang, & Lupski, 2008). Using this information for the purpose of designing a structural variant has not been done so far by any of the current tools. There is sufficient evidence to define these regions bioinformatically (Ou et al., 2011) and thus be able to use the signature of NAHR programmatically in tool design which was attempted in this study. The novelty of this approach is the use of inherent biologically processes driving these variants formation and is the first such attempt to program this information in a tool. This study tried to capture 3-D packing of the genome within a cell bioinformatically, thus taking the concept of “context” one step further. Translocations occurring from double-stranded breaks also do not occur randomly. Chromosomes which are more close to each other and with similar replication timings are more likely to interact and exchange genomic material through NAHR (Yaffe & Tanay, 2011), (De & Michor, 2011), (Wijchers, 2011). Instead of using empirical distribution of reads and variants in the data to derive

confidence score, this study attempted to use probability distribution of two regions in the genome interacting due to physical proximity. This study has thus introduced the concept of context-based evaluation of structural variation in the human genome as a novel approach.

4.2 Designing the Algorithm

The algorithm for the tool is shown in Figure 4.1. Broadly the tool could be divided into 3 major steps:

1. Extraction of anomalous reads from the bam file
2. Re-alignment using BLAT
3. Identifying potential translocations and defining probabilities using Hi-C data

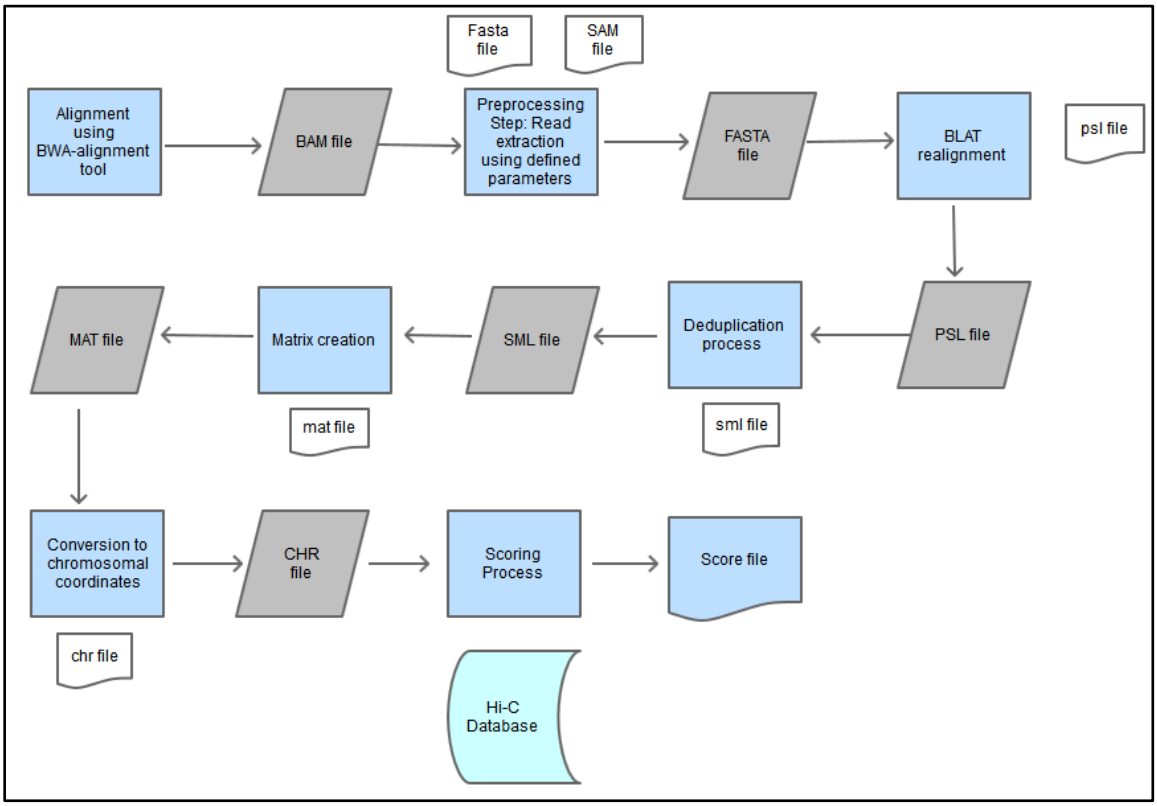


Figure 4.1: Novel algorithm flowchart

4.2.1 Read extraction overview

Sequence Alignment Format: There are many tools available for mapping reads to the reference genome like BWA, Bowtie, MAQ. These tools give the output in a common format known as SAM format (Sequence Alignment/Map Format), which had been widely accepted as the default format for generating output files. Briefly the alignment output has a header section and an alignment section. All headers start with @ (*eg*. *@HD*) and contain general information about the experimental process used to generate the sequencing reads. Alignment sections contain 11 mandatory fields as listed in Table 4.1.

Table 4.1: SAM format fields: alignment section

Field	Type	Regexp/Range	Brief description
QNAME	String	[!-?A-~]{1,255}	Query template NAME
FLAG	Int	$[0,2^{16}-1]$	bitwise FLAG
RNAME	String	*|[!-()+-<>-~][!-~]*	Reference sequence NAME
POS	Int	$[0,2^{29}-1]$	1-based leftmost mapping POSition
MAPQ	Int	$[0,2^{8}-1]$	MAPping Quality
CIGAR	String	*|([0-9]+[MIDNSHPX=])+	CIGAR string
RNEXT	String	*|=|[!-()+-<>-~][!-~]*	Ref. name of the mate/next read
PNEXT	Int	$[0,2^{29}-1]$	Position of the mate/next read
TLEN	Int	$[-2^{29}+1,229-1]$	observed Template LENgth
SEQ	String	*|[A-Za-z=.]+	segment SEQuence
QUAL	String	[!-~]+	ASCII of Phred-scaled base QUALity+33

The purpose of the extraction algorithm is to retrieve reads which were assigned by the aligner as improperly mapped. This information is coded in the bitwise 'FLAG' field of the SAM file as shown in Table 4.2.

Table 4.2: Bitwise FLAG of SAM file

Bit	Description
0x1	template having multiple segments in sequencing
0x2	each segment properly aligned according to the aligner
0x4	segment unmapped
0x8	next segment in the template unmapped
0x10	SEQ being reverse complemented
0x20	SEQ of the next segment in the template being reversed
0x40	the first segment in the template
0x80	the last segment in the template
0x100	secondary alignment
0x200	not passing quality controls
0x400	PCR or optical duplicate
0x800	supplementary alignment

The above table is represented in the hexadecimal system. The corresponding binary and decimal conversion is shown in Table 4.3.

Table 4.3: Bitwise Flag of SAM FLAG- binary to decimal conversion

Bit	Binary	Decimal
0x1	1	1
0x2	10	2
0x4	100	4
0x8	1000	8
0x10	100000	16
0x20	1000000	32
0x40	10000000	64
0x80	100000000	128
0x100	1000000000	256
0x200	10000000000	512
0x400	100000000000	1024
0x800	1000000000000	2048

The value for each bit is set to 0 or 1 based on the description, with 0 being *no* and 1 being *yes*. Thus if a sequencing segment which has been mapped has a value of 64, it is the first segment in the template (Table 4.1, Table 4.2). Each mapped segment

will have a value as represented in the decimal system in the SAM file. Thus a value of 99 equals 64+32+2+1. The corresponding natural language interpretation based on the tables 4.1 and 4.2 will be as below:

FLAG	Bit	Description
64	0x40	the first segment in the template
32	0x20	SEQ of the next segment in the template being reversed
2	0x2	each segment properly aligned according to the aligner
1	0x1	template having multiple segments in sequencing

Thus the read with a SAM flag of 99 meant it mapped correctly. The conversion of SAM flag values to the corresponding interpretation can be done using one of the utilities of PICARD tool suite.[12]

CIGAR String: The sixth field in a SAM file is represented by the 'CIGAR' string. Just as 'FLAG' is used to describe the overall information of sequence read in terms of mapping to the reference genome, 'CIGAR' string explains similar information in terms of mapping at the base-pair level. Thus each base in a sequence read will have any one of the CIGAR values listed in Table 4.4.

Table 4.4: CIGAR String

Operation	BAM	Description
M	0	alignment match (can be a sequence match or mismatch)
I	1	insertion to the reference
D	2	deletion from the reference
N	3	skipped region from the reference
S	4	soft clipping (clipped sequences present in SEQ)
H	5	hard clipping (clipped sequences NOT present in SEQ)
P	6	padding (silent deletion from padded reference)
=	7	sequence match
X	8	sequence mismatch

[12] PICARD tool webpage: http://picard.sourceforge.net/index.shtml, http://picard.sourceforge.net/explain-flags.html

An example of a sequence read mapped to the reference genome can be represented as:

Reference Genome Sequence: A T T T G C A T C C C G T A T T G G C A
Query Read Sequence: A T T T G C T C C C G A T A T T G G C A
CIGAR String: 6M1D5M1I5M 6 Match, 1 Deletion, 5 Match, 1 Insertion, 8 Match

Reference Genome Sequence: A T T T G C A T C C C G T A T T G G C A
Query Read Sequence: A T T T C A T C C C G T A g g g g g
CIGAR String: 4M1D9M5S 4 Match, 1 Deletion, 9 Match, 5 Soft clipped

The CIGAR string informs about how many insertions and deletions of base pairs is needed to be done in the query sequence for the mapping tool to align the read to a given location on the reference genome.

FLAGs for read extraction: All mapping tools will give the output in SAM format with a *FLAG* assigned. The purpose of read extraction is to identify those reads that did not have enough information contained in the sequence for the mapping tool to align it at the correct location. This information is extracted using the *FLAG* field and 'CIGAR' string explained in Section 4.2.1. The *FLAG* fields used for extraction are shown in Table 4.5.

Table 4.5: FLAGs for read extraction

Flag	Fields set to derive Flag	Description of fields for Flag
UNMAPPED READS		
101	64+32+4+1	first in pair, mate reverse strand, **read unmapped**, read paired
89	64+16+8+1	first in pair, read reverse strand, **mate unmapped**, read paired
117	64+32+16+4+1	first in pair, mate reverse strand, read reverse strand, **read unmapped**, read paired
121	64+32+16+8+1	first in pair, mate reverse strand, read reverse strand, **mate unmapped**, read paired
165	128+32+4+1	second in pair, mate reverse strand, **read unmapped**, read paired
153	128+16+8+1	second in pair, read reverse strand, **mate unmapped**, read paired
185	128+32+16+4+1	second in pair, mate reverse strand, read reverse strand, **read unmapped**, read paired
181	128+32+16+8+1	second in pair, mate reverse strand, read reverse strand, **mate unmapped**, read paired
69	64+4+1	first in pair, **read unmapped**, read paired
73	64+8+1	first in pair, **mate unmapped**, read paired
133	128+4+1	second in pair, **read unmapped**, read paired
137	128+8+1	second in pair, **mate unmapped**, read paired
77	64+8+4+1	first in pair, **mate unmapped, read unmapped**, read paired
141	128+8+4+1	second in pair, **mate unmapped, read unmapped**, read paired
READS WITH NO INFORMATION ON MAPPING		
65	64+1	first in pair, read paired
129	128+1	second in pair, read paired
81	64+16+1	first in pair, read reverse strand, read paired
97	64+32+1	first in pair, mate reverse strand, read paired
145	128+16+1	second in pair, read reverse strand, read paired
161	128+32+1	second in pair, mate reverse strand, read paired
113	64+32+16+1	first in pair, mate reverse strand, read reverse strand, read paired
177	128+32+16+1	second in pair, mate reverse strand, read reverse strand, read paired

Flag	Fields set to derive Flag	Description of fields for Flag
READS WITH IMPROPER ORIENTATION		
115	64+32+16+2+1	first in pair, mate reverse strand, read reverse strand, read mapped in proper pair, read paired
179	128+32+16+2+1	second in pair, mate reverse strand, read reverse strand, read mapped in proper pair, read paired
67	64+2+1	first in pair, read mapped in proper pair, read paired
131	128+2+1	second in pair, read mapped in proper pair, read paired

4.2.2 Read extraction algorithm

Perl script algorithm to extract the reads using the 'Flag' field and 'CIGAR' string was used as described in Table 4.6.

Table 4.6: Read_Extract Algorithm

Assumption: The input file has to be in BAM/SAM format.	Creating FASTA file from BAM output of mapping tool

Input Arguments for file name;
if (file is in SAM/BAM format)
{ Create output files, both FASTA and SAM;}
Else
{output error to screen}
Matchlist array=(List of FLAGs)
	Creates input argument array and array of FLAGs to be used, output files with .fa and .sam extension

While { Read each line of SAM/BAM file; Split into fields based on delimiter; Set fields 0: Read Identifier 1: FLAG 5: CIGAR 8: TLEN 9: SEQUENCE as variables;	Variables identified for read extraction
If (readcounter==1) {Calculate 30% of read length Create pattern for longest continuous 'N's which equals 30% of read length Increment readcounter}	Creates 'NNNN' pattern which is 30% of read length to search for in SEQUENCE; pattern created just once
If (currentReadIdentifier==previousReadIdentifier	Matches FLAG

AND (currentReadIdentifier OR previousReadIdentifier == matchlist array)) { if ('N' pattern not found) {print to output FASTA file with <RID1>/ <RID2> as suffix to identify two reads of a pair and also entire line to SAM file} } } END OF WHILE LOOP Close all files	array and extracts only those reads where both reads of a pair are present and did not have more than 30% 'N's. It further adds identifier <RID1>, <RID2> to keep paired end reads information

The output from the extraction process is saved in both FASTA and SAM format. The FASTA file serves as the input for BLAT mapping tool. Both reads of a **pair** now have a **unique identifier** (RID1, RID2) which is an important addition in this step.

4.3 Re-alignment Using BLAT

4.3.1 The need for BLAT

BLAT which stands for BLAST-like alignment tool was developed in 2001 by James Kent at University of California, Santa Cruz. The popular alignment tool BLAST (Altschul, Gish, Miller, Myers, & Lipman, 1990) developed by the NCBI uses the Smith-Waterman algorithm (Smith et al., 1981) to do local alignment of a defined length. The extension is done such that further extension of the segment will not improve the score.

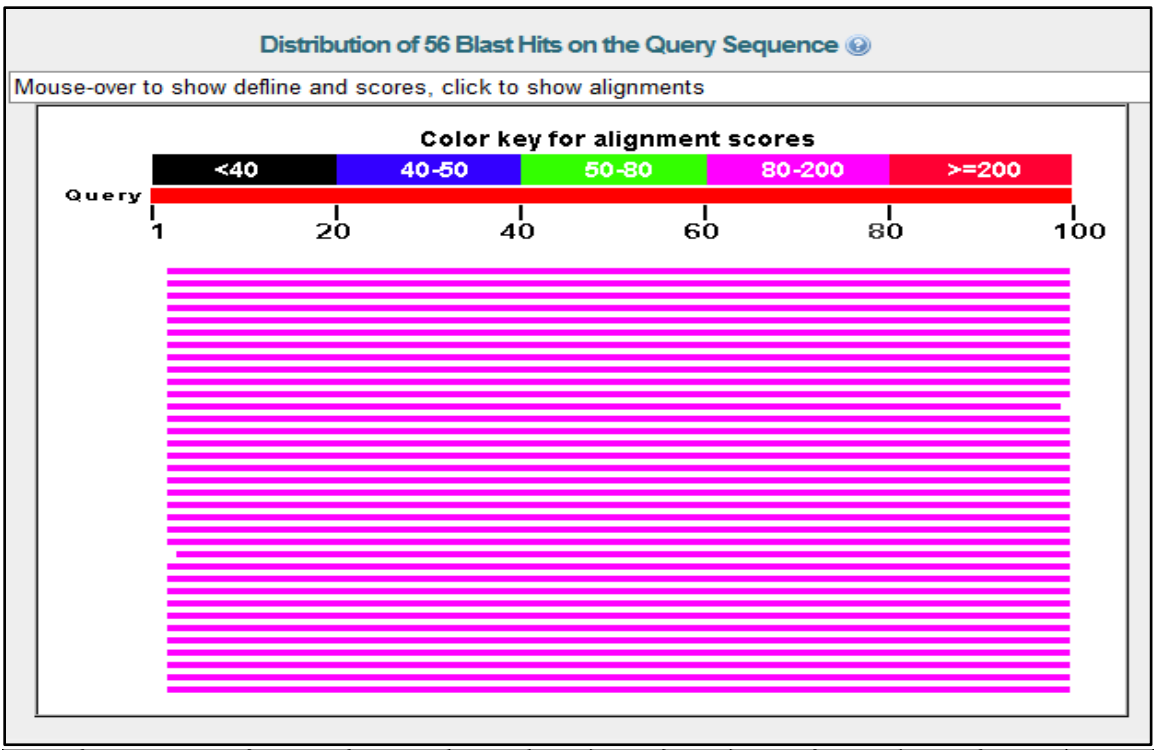
BLAT (Kent, 2002) was used primarily because of its speed and comparable output to BLAST. For example a sample output from the simulated data of translocations had FLAGs set as 117 and 153 such that the first read was unmapped and the second read mapped to a region with 100% match (Figure 4.2). The output from BLAST and BLAT are shown in Figures 1 and 2. BLAT is very similar to BLAST in terms of its search algorithm but differs in the fact that BLAST builds an index of the

query to search through its database, while BLAT used the database to build its index and search through the query which significantly affects search time.

```
chr12_12610644_12611092_6:0:0_3:0:0_23bee    117   chr4   191129790    0    *    =    191129790    0
ACTCTCCCATGTCTACTTCTTTATACACAGTCACGGCAACCATCTGATTTATCAATCTTTTCCCCACCTGACCCCCCTTTCCATTCCACAAAACCGCCAT
2222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
RG:Z:transl_RG_F
chr12_12610644_12611092_6:0:0_3:0:0_23bee    153   chr4   191129790    2    100M   =    191129790    0
CCTAGGAAAACCAGAGACCTTTGTTCACTTGTTTATCTGCTGACCTTCCCTACTCTATTGTCCTGTGACCCGGCCAAATCCCCCTCTGCGAGAAACACCC
2222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
RG:Z:transl_RG_F    XT:A:U  NM:i:2  SM:i:2  AM:i:0  X0:i:1  X1:i:137    XM:i:2  XO:i:0  XG:i:0  MD:Z:51C1A46
```

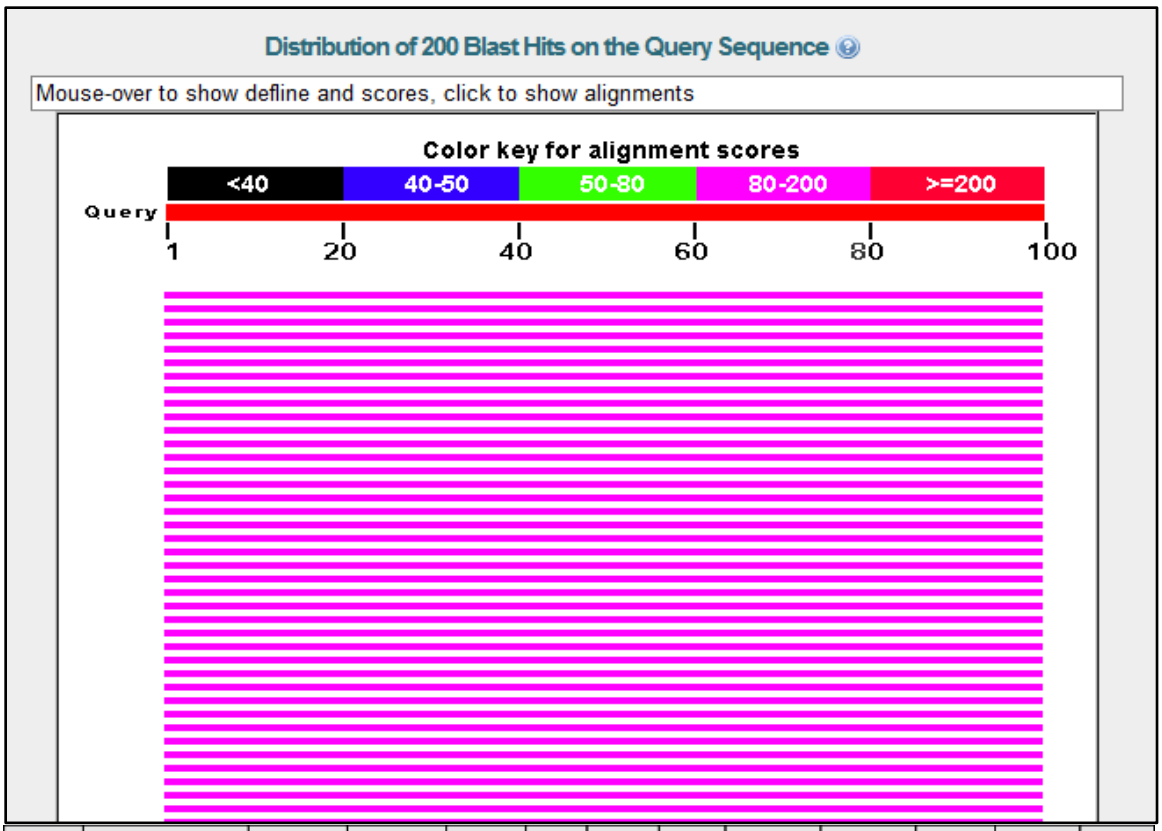
Figure 4.2: Sample paired-end read of simulated dataset: SAM format

BLAST output for the first segment of the pair (with FLAG=117) showed a match with 56 regions in the 'Human plus Transcript' database and MegaBLAST algorithm (Figure 4.3). The corresponding mate segment of the pair (with FLAG=153) showed match with 200 regions in the same database and same algorithm (Figures 4.4).



query id	subject ids	% identity	alignment length	mis matches	gap opens	q. start	q. end	s. start	s. end	evalue	bit score	Chromosome
87745	gi|528476594|ref|NC_018923.2|	95	98	5	0	3	100	12684410	12684313	2.00E-35	154	12
87745	gi|157704453|ref|AC_000144.1|	95	98	5	0	3	100	12484692	12484595	2.00E-35	154	12
87745	gi|568815586|ref|NC_000012.12|	95	98	5	0	3	100	12566540	12566443	2.00E-35	154	12
87745	gi|528476586|ref|NC_018924.2|	94	98	6	0	3	100	23401327	23401424	1.00E-33	148	13
87745	gi|528476558|ref|NC_018928.2|	94	98	6	0	3	100	29125785	29125882	1.00E-33	148	17
87745	gi|528476524|ref|NC_018934.2|	94	98	6	0	3	100	41505041	41504944	1.00E-33	148	X
87745	gi|528476524|ref|NC_018934.2|	92	98	8	0	3	100	46286655	46286752	2.00E-30	137	X
87745	gi|157734237|ref|AC_000155.1|	94	98	6	0	3	100	39203380	39203283	1.00E-33	148	X
87745	gi|157734237|ref|AC_000155.1|	92	98	8	0	3	100	43967639	43967736	2.00E-30	137	X
87745	gi|157713538|ref|AC_000149.1|	94	98	6	0	3	100	25272864	25272961	1.00E-33	148	17
87745	gi|157704454|ref|AC_000145.1|	94	98	6	0	3	100	4241786	4241883	1.00E-33	148	13
87745	gi|568815581|ref|NC_000017.11|	94	98	6	0	3	100	30736111	30736208	1.00E-33	148	17
87745	gi|568815575|ref|NC_000023.11|	94	98	6	0	3	100	41613264	41613167	1.00E-33	148	X

Figure 4.3. MegaBLAST output with defaults for query sequence pair1



query id	subject ids	% identity	alignment length	mis matches	gap opens	q. start	q. end	s. start	s. end	evalue	bit score	Chromosome
214287	gi|528476651|ref|NC_018915.2|	98	100	2	0	1	100	190877127	190877226	2.00E-41	174	4
214287	gi|528476651|ref|NC_018915.2|	97	100	3	0	1	100	37671084	37671183	8.00E-40	169	4
214287	gi|528476651|ref|NC_018915.2|	97	100	3	0	1	100	46378025	46378124	8.00E-40	169	4
214287	gi|528476651|ref|NC_018915.2|	97	100	3	0	1	100	70664880	70664979	8.00E-40	169	4
214287	gi|528476651|ref|NC_018915.2|	97	100	3	0	1	100	79446585	79446684	8.00E-40	169	4
214287	gi|528476651|ref|NC_018915.2|	97	100	3	0	1	100	170945138	170945237	8.00E-40	169	4
214287	gi|528476651|ref|NC_018915.2|	96	100	4	0	1	100	847826	847727	4.00E-38	163	4
214287	gi|528476651|ref|NC_018915.2|	96	100	4	0	1	100	15791910	15792009	4.00E-38	163	4
214287	gi|528476651|ref|NC_018915.2|	96	100	4	0	1	100	23371967	23372066	4.00E-38	163	4
214287	gi|528476651|ref|NC_018915.2|	96	100	4	0	1	100	28516996	28516897	4.00E-38	163	4
214287	gi|528476651|ref|NC_018915.2|	96	100	4	0	1	100	30930756	30930855	4.00E-38	163	4
214287	gi|528476651|ref|NC_018915.2|	96	100	4	0	1	100	37822314	37822413	4.00E-38	163	4
214287	gi|528476651|ref|NC_018915.2|	96	100	4	0	1	100	38809775	38809874	4.00E-38	163	4

Figure 4.4: MegaBLAST output with defaults for query sequence pair2

For the same two query sequences, BLAT output is represented below (Figures 4.5, 4.6).

BLAT Search Results

ACTIONS	QUERY	SCORE	START	END	QSIZE	IDENTITY	CHRO	STRAND	START	END	SPAN
browser details	YourSeq	88	1	100	100	94.0%	12	-	12610644	12610743	100
browser details	YourSeq	86	1	100	100	93.0%	X	-	41357364	41357463	100
browser details	YourSeq	86	1	100	100	93.0%	2	-	143176874	143176973	100
browser details	YourSeq	86	1	100	100	93.0%	17	+	26087253	26087352	100
browser details	YourSeq	84	1	100	100	92.0%	X	-	73304225	73304324	100
browser details	YourSeq	84	1	100	100	92.0%	X	-	47967119	47967218	100
browser details	YourSeq	84	1	100	100	92.0%	7	-	139778194	139778293	100
browser details	YourSeq	84	1	100	100	92.0%	7	-	26268700	26268799	100
browser details	YourSeq	84	1	100	100	92.0%	6	-	74368418	74368517	100
browser details	YourSeq	84	1	100	100	92.0%	4	-	65448670	65448769	100
browser details	YourSeq	84	1	100	100	92.0%	3	-	48747410	48747509	100

Figure 4.5: BLAT output with defaults for query sequence pair1

BLAT Search Results

ACTIONS	QUERY	SCORE	START	END	QSIZE	IDENTITY	CHRO	STRAND	START	END	SPAN
browser details	YourSeq	96	1	100	100	98.0%	4	+	191129790	191129889	100
browser details	YourSeq	94	1	100	100	97.0%	X	-	119493641	119493740	100
browser details	YourSeq	94	1	100	100	97.0%	X	-	7536075	7536174	100
browser details	YourSeq	94	1	100	100	97.0%	9	-	118642009	118642108	100
browser details	YourSeq	94	1	100	100	97.0%	9	-	111899086	111899185	100
browser details	YourSeq	94	1	100	100	97.0%	9	-	94028941	94029040	100
browser details	YourSeq	94	1	100	100	97.0%	9	-	94632200	94632299	100
browser details	YourSeq	94	1	100	100	97.0%	8	-	145357732	145357831	100
browser details	YourSeq	94	1	100	100	97.0%	8	-	129199276	129199375	100

Figure 4.6: BLAT output with defaults for query sequence pair2

The BLAT output was comparable with BLAST and took less time even with a web interface. BLAT had 200+ entries for each query pair and showed significant identity with chromosome 12 for sequence pair 1 (Figure 1 and 2) and chromosome 4 for sequence pair 2. More importantly, BLAST did not show any output if the option for filtering in repeat regions was turned on. BLAT does not have such an option and gives all possible hits.

4.3.2 BLAT alignment method used in tool

We used the FASTA output from the read extraction as the input for BLAT tool. BLAT can be run as server-client interface where the user creates a server (gfServer) that

keeps the genome index in memory while the client (gfClient) is used to input query sequences that are sent to the gfServer. The server-client program has also been combined into a single standalone program such that the database and query are sent as arguments to the program. Instructions on setting up the client and options used by BLAT are provided in Appendix B. A perl script was written that allows the user to set up the server-client interface in a single step. A pre-requisite for this was that the 2-bit file for the genome needs to be made using a *BLAT-faToTwoBit* program. This program converts a FASTA file into ".2bit" format that is then used by gfServer to create the reference genome index file. In order to speed up the process, the reference genome was split by chromosome, and the query was run against each database in parallel.

We used BLAT-version 35.1 to run the queries with minimum tile match of 4 and minimum identity of 95%. These queries were run in parallel, one for each chromosome. Output from BLAT is presented in the psl format (Appendix B, BLAT specification) which has the fields as defined in Table 1.1. The output from each chromosome mapping was finally merged together and sorted by 'Identifier' (field 10), 'Chromosome' (field 14) and 'Start position' (field 16) to get the sorted psl output file.

Table 4.7: BLAT output file format: psl

Field #	Field Names (*Type*)	Description
1	Matches (*int unsigned*)	Number of bases that match that aren't repeats
2	Mismatches (*int unsigned*)	Number of bases that don't match
3	repMatches (*int unsigned*)	Number of bases that match but are part of repeats
4	nCount (*int unsigned*)	Number of 'N' bases
5	qNumInsert (*int unsigned*)	Number of inserts in query
6	qBaseInsert (*int unsigned*)	Number of bases inserted in query
7	tNumInsert (*int unsigned*)	Number of inserts in target
8	tBaseInsert (*int unsigned*)	Number of bases inserted in target
9	Strand (*char(2)*)	+ or - for query strand optionally followed by + or – for target strand
10	qName (*varchar(255)*)	Query sequence name
11	qSize (*int unsigned*)	Query sequence size
12	qStart (*int unsigned*)	Alignment start position in query
13	qEnd (*int unsigned*)	Alignment end position in query
14	tName (*varchar(255)*)	Target sequence name
15	tSize (*int unsigned*)	Target sequence size
16	tStart (*int unsigned*)	Alignment start position in target
17	tEnd (*int unsigned*)	Alignment end position in target
18	blockCount (*int unsigned*)	Number of blocks in alignment. A block contains no gaps.
19	blockSizes (*longblob*)	Size of each block in a comma separated list
20	qStarts (*longblob*)	Start of each block in query in a comma separated list
21	tStarts (*longblob*)	Start of each block in target in a comma separated list

4.4 De-Duplication Algorithm

The de-duplication algorithm essentially checks for rows with the same 'Identifier' and 'Chromosome' and bins it into buckets of 1000 basepairs windows. The output for this algorithm produces an SML file, an in-house defined format (Table 4.8).

Table 4.8: De-Duplication Algorithm

Assumption: The input file is in psl format which has been sorted by 'Read identifier', 'Chromosome' and 'Start Position'	Creating sml file from psl file

Input Arguments for file name;
if (file is in sorted psl format)
{ Create output file with sml extension;}
Else
{output error to screen}
	Creates output file with .sml extension

While { Read each line of psl file; Split into fields based on delimiter; Set fields 9:Identifier 13:Chromosome 15:Start position 16:End position 17:BlockCount as variables; Define variables current-range and previous-range using GetRange() function return value	Sets variables and defines new variables current-range and previous-range

If (Prev Identifier not equal Curr Identifier)
{Insert Read(to output file)}
Else
if (Prev Chr not equal Curr Chr)
{if (BlockSize==1)
{Insert Read(to output file)}
Else
if (current-range not equals previous-range)
{if (BlockSize==1)
{Insert Read(to output file)}
}
}
} END OF WHILE LOOP
 	
Close all files
	Bins data in 1000 basepair regions. If identifiers and chromosomes are same, compress the rows into 1000 basepair bins, finally filtered on BlockSize; de-duplication of rows

Sub GetRange{
rangeSpan=1000;
midpoint=(StartPosition+EndPosition)/2
if (midpoint < rangeSpan)
{
	GetRange function defines the 1000 base-pair blocks into which the consecutive rows

grange = 1;
}
else
{
grange = int(midpoint/rangespan) + 1;
}
return grange ;
}
	with same identifier and chromosome are compressed i.e. de-duplicated. Mid-point of length of mapping region is used to check for range span. The return value of the function is the **offset** for each bin starting from 1 for 1-1000 bp bin

Sub InsertRead{
print 'Identifier', 'Chromosome', 'Start position',
'Start position', 'current-range'
}
	Output format for the sml file defined

Table 4.9: Example sml file from de-duplication algorithm

Identifier	Chromosome	Start Position	End Position	Offset for Range
4_103112562_103113055_5:1:0_2:0:0_dedbba<RID>1	4	950	1010	1
4_14019666_14020194_6:0:0_1:0:0_17f93b6<RID>1	8	999	1010	2
4_4086095_4086615_3:0:0_2:0:0_a6acbd<RID>1	4	4086515	4086615	4087
4_4086106_4086642_3:0:0_2:0:0_5cdfa1<RID>1	4	4086105	4086205	4087
4_4086106_4086642_3:0:0_2:0:0_5cdfa1<RID>2	4	4086542	4086642	4087

4.5 Create_Matrix Algorithm

The SML file created from de-duplication algorithm serves as input file for matrix creation. The SML file essentially contains all the possible mapping locations for the queries in PSL file on the reference genome based on BLAT mapping algorithm in 1000 base pair windows.

The *Create_Matrix* algorithm creates all possible combinations of the read pair (RID1 and RID2), such that each row in output file is the location where readpair1 maps and the corresponding location where readpair2 maps on the human genome. The algorithm is defined in Table 4.10. The *Create_Matrix* algorithm essentially defines how to capture the NAHR regions and can be represented in the Figure 4.7a, 4.7b. Output is a MAT file with an example in Table 4.11.

Table 4.10: Create_Matrix Algorithm

Assumption: The input file is in sml format and is sorted by 'Read Identifier' and 'Chromosome'	Creating mat file from sml file

Input Arguments for file name;
if (file is in sorted sml format)
{ Create output files with mat extension;}
Else
{output error to screen}
	Creates output files with .mat extension

While { Read each line of psl file; Split into fields based on delimiter; Set fields 1:Identifier 2:Chromosome 3:Start position 4:End position 5:Range offset;	Sets variables

If (previous Identifier==current Identifier)
{Insert Read(to output file)}
Else
{ Create_Matrix()
Insert Read()
}
If first line of file
{Insert Read(to output file}
} END OF WHILE LOOP	

Close all files Do UNIX sort on Read1,Read2, Range Offset1, Range Offset2	

Sub Create_Matrix{
Foreach read1
{
Foreach read2
{
If (Chromosome and Range offset is same)
{do not print}
Else
{
If (swapFunction==False)
{print to outfile read1, read2}
Else
{print to outfile read2, read1}
}
}
}
For (read1)
{
For (next read1)
{
If (Chromosome and Range offset is same)
{do not print}
Else
{
If (swapFunction==False)
{print to outfile read1, next read1}
Else
{print to outfile nextread1, read1}
}
}
}
}
For (read2)
{
For (next read2)
{
If (Chromosome and Range offset is same)
{do not print}
Else
{
If (swapFunction==False)
{print to outfile read2, next read2}
Else
{print to outfile nextread2, read2}
}
}
	Matrix is created based on conditions; if both read pairs map to the same region we do not print the output as these do not suggest a translocation. Else if read1 and read2 go on different locations then print to outfile.
	
	Second condition checks for read1 going to different locations i.e. the same read mapping to different locations and creating a matrix of these combinations. Printing of values is such that Chromosome field1 is always less than Chromosome field2 and Chromosome X is always less than Chromosome Y
	
	Third condition checks for read2 going to different locations i.e. the same read mapping to different locations and creating a matrix of these combinations. Printing of values is such that Chromosome field1 is always less than Chromosome field2 and Chromosome X is always less than Chromosome Y.

} }	

Sub InsertRead {
If (Identifier==Read1)
{ write array for Read1;Chromosome and Offset}
 	
If (Identifier==Read2)
{ write array for Read2;Chromosome and Offset}
}
	Creates array for read1 and read 2 and pushes each read into the respective array based on Identifier

Sub SwapFunction{
Set default SwapFunction return-value==No;
Create arg1: Chromosome read1
arg2: Chromosome read2
arg3: Offset read1
arg4: Offset read2
If (Chromosome read1 ==Y and Chromosome read2==X)
{ swap return value==Yes}
Else
If (Chromosomes read1 and 2 are integers and Chromosome read1>Chromosome read2)
{ swap return value==Yes }
Return swap return-value;
}
	This function returns a value of “Yes” if Chromosome on field1 is greater than Chromosome in field2. Y is greater than X as defined.

Table 4.11: Example ‘mat’ file from create_matrix algorithm

Chromosome1	Offset1	Chromosome2	Offset2
1	1001	4	2021
1	1004	4	5036
1	2385	8	6236
1	5321	8	8288
4	4087	8	6974
4	4087	8	6974
4	4087	8	6974

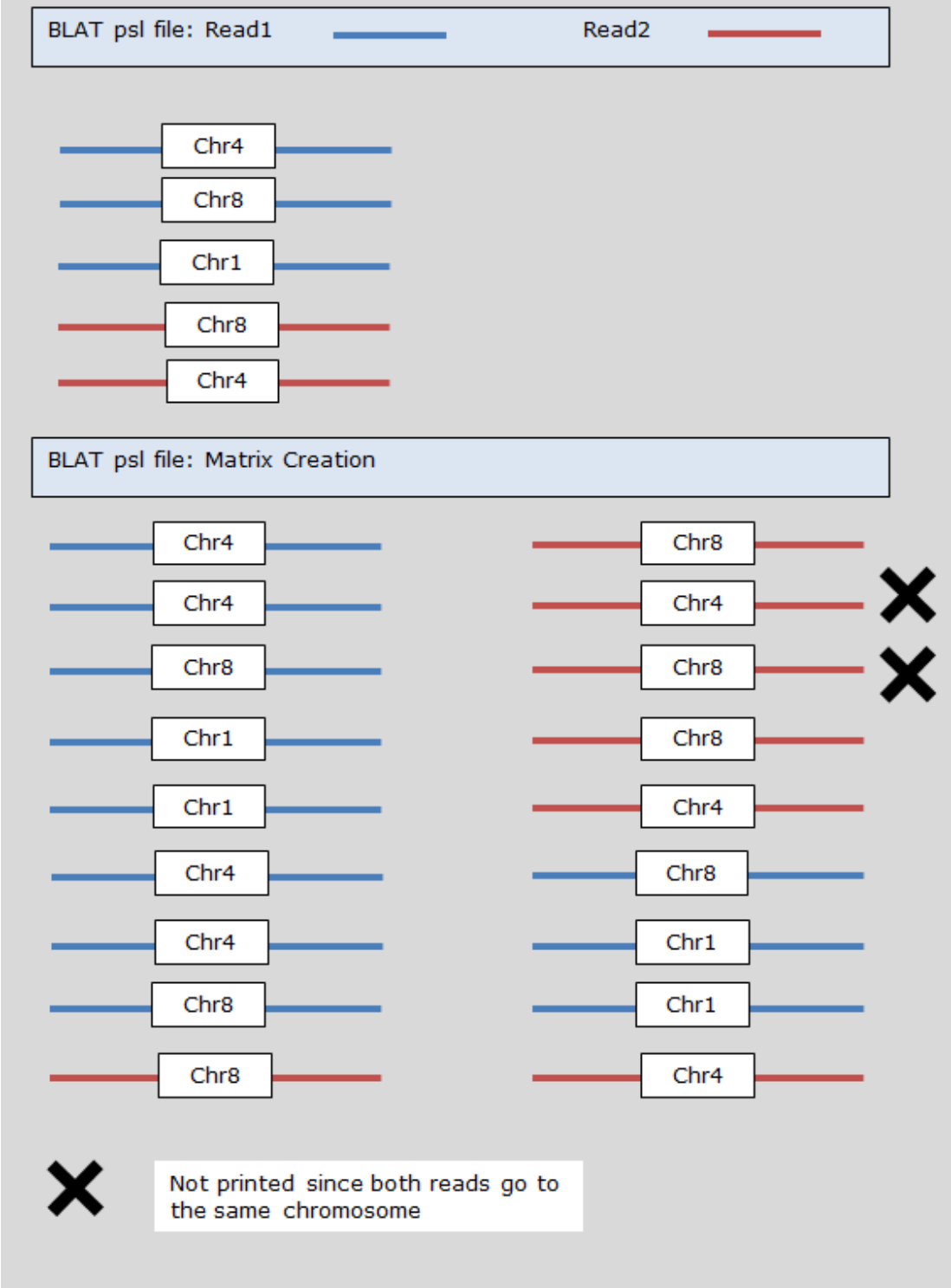


Figure 4.7a: Create_Matrix Algorithm: Matrix creation with all possible combinations of read1 and read2 excluding reads where both go to same chromosome.

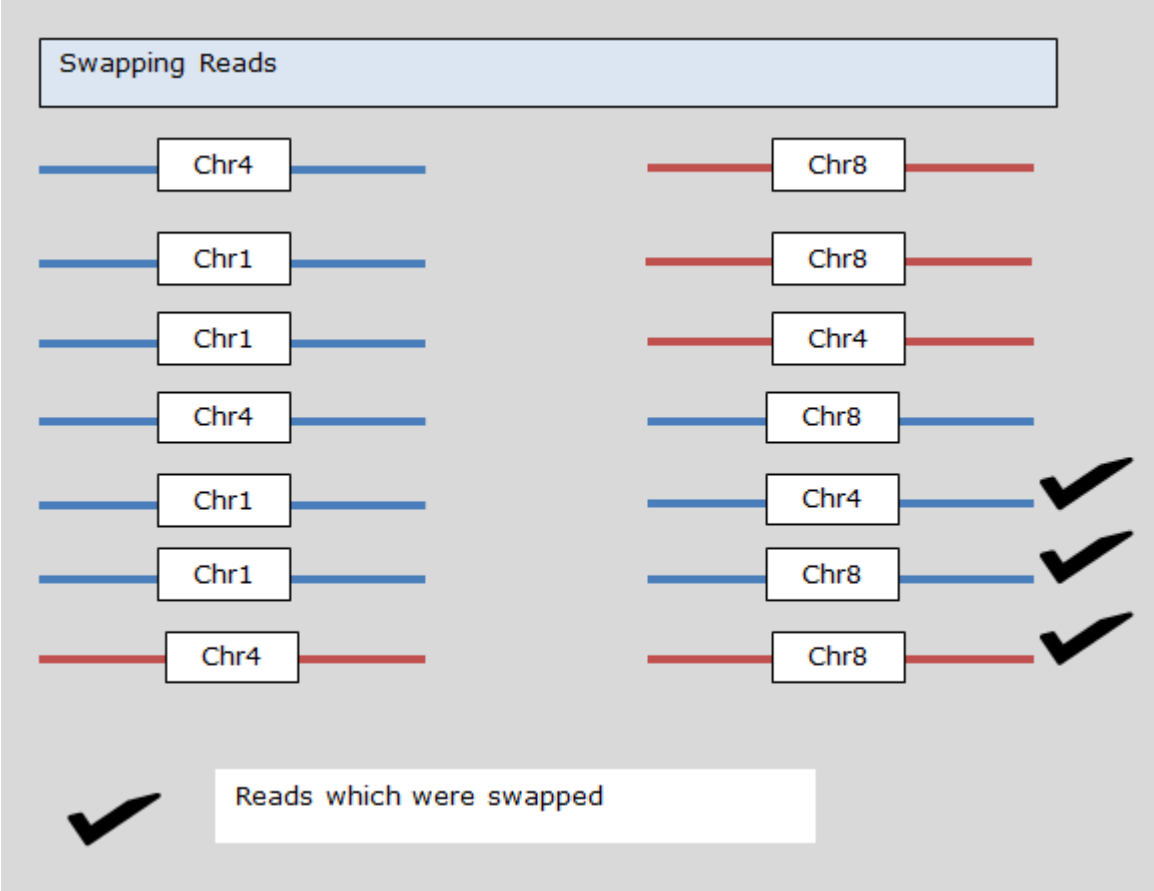


Figure 4.7b: Create_Matrix Algorithm: Swapping reads such that lower number chromosome is always on the left which makes the counting process efficient.

Non-allelic homologous recombination regions are defined by their sequence identity (>95%) with different regions of the genome and greater than 5kb in size such that these form substrates for anomalous pairing during repair. BLAT identifies all possible regions such that read-pair1 maps to one region and read-pair2 maps to another region which will therefore identify potential NAHR regions. The *Create_Matrix* algorithm, in creating the matrix, identifies potential NAHR pairs and therefore potential translocation breakpoints. Further support is added if read1 also maps to both locations on different chromosomes identified as potential breakpoint regions (e.g. Read1 mapping to chromosome 4 and chromosome 8) suggesting that these regions could be homologous and share significant identity. The purpose of

swapping the reads is to sort the reads for easier counting. This finally gives the number of reads supporting a translocation identified by the matrix.

4.6 Write_Count Algorithm

Output from matrix creation is a sorted MAT file and is the input for the write_count algorithm. The output for the write_count algorithm is a CHR file as shown in example Table 4.12.

Table 4.12: Write_count Algorithm

Assumption: The input file is in mat format and is sorted by field1, field3, field2 and field4 in that order, i.e. Chromosome1, Chromosome2, Offset1, Offset2	Creating chr file from mat file

Input Arguments for file name;
if (file is in sorted .mat format)
{ Create output files with .chr extension;}
Else
{output error to screen}
	Creates output files with .chr extension

While (Read each line of mat file)
{
If (previous record==current record)
{Count=1; Increment count}
Else
{WriteCount(to output file)}
} END OF WHILE LOOP
 	
Close all files
	Counts number of records which are same

Sub WriteCount{
Print each unique row with count;
}
	Creates '.chr' output file format

Table 4.13: Example 'chr' file from write_count algorithm

Chromosome1	Offset1	Chromosome2	Offset2	Count
1	1001	4	2021	1
1	1004	4	5036	1
1	2385	8	6236	1
1	5321	8	8288	1
4	4087	8	6974	3

4.7 Get_HiC-Score Algorithm

The last step in the process was to define the probability of calling a translocation as not false. We did not used distribution of insert size within the dataset to define this probability as is done by other variant calling programs (SVDetect, BreakDancer, PEMer). The cancer genome is highly heterogeneous with various clonal populations showing different types of variation signatures. Even with a 30X coverage of sequencing experiment, the likelihood of picking a variant in the heterogeneous cancer sample cannot be defined based on the distribution of reads in the current sample. We therefore wanted to capture as many variants as we could define using our algorithm and assign the probability based on an informative prior. This was determined using Hi-C experiments (Lieberman-Aiden et al., 2009) which takes into account the three dimensional positioning of the genomes within a cell and defines the probability of two regions of the genome interacting based on their physical proximity to each other, as determined experimentally. For the purpose of this analysis we used the database created by Lieberman-Aiden's groups at BROAD/MIT.[13] The datasets contain Pearson's correlation coefficient for each combination of chromosome in one million base-pair (1Mbps) windows. This is the highest level of resolution for these experiments, and we had our variants defined in 1000 basepair windows. We had to accept this as a limitation of the study.

[13] Hi-C database webpage: http://hic.umassmed.edu/welcome/welcome.php

Table 4.14: Get_HiC-Score Algorithm

Assumption: The input file is in chr format and is sorted by field1(chr1), field3(chr2), field2(Offset1) and field4(Offset2) in that order	Creating FASTA file from BAM output of mapping tool

Input Arguments for file name, path to Hi-C file;
if (file is in sorted .chr format)
{ Create output files with .score extension;}
Else
{output error to screen}
	Creates output files with .score extension

While (Read each line of chr file)
{
Split into fields based on delimiter;
Set fields 1: Chromosome1
2: Offset1
3: Chromosome2
4: Offset2
5: Chromosome Count
6: Current Hi-C filename
Create Hi-C filename by dynamically setting values of chromosome1 and chromosome2 on Hi-C filename template	

If (current Hi-C filename not equals previous Hi-C filename)
{
Open Hi-C file ()
}
row-midrange= CalcMidPosition(offset1)
column-midrange=CalcMidPosition(offset2)
rowposition=GetFilePosition(row-midrange);
colposition=GetFilePosition(column-midrange);
score = GetHi-CScore()
CalcRange(Offset1,Offset2)
If (score>0)
{
Print to outfile chr1,start1,end1,chr2,start2,end2
numberOfReads, scorefrom Hi-C
}
} END OF WHILE LOOP
 	
Close all files
	Open each Hi-C file only once and calculate midrange of row and column offsets. Use these offsets to get the exact column and row position in Hi-C file, open Hi-C file and extract the score for x-row and y-cloumn.
	Only Hi-C scores with Pearson's probability correlation greater than zero printed to output

Sub Open Hi-C file{
Concatenates Hi-C filepath variable to dynamically created current Hi-C filename and open corresponding file.
Puts entire file into @filearray;
}
	Opens Hi-C file bases on chromosome1 and chromosome2 values and their offsets. Puts the entire file into file array which is scanned in the GetHi-CScore function for getting score.

	Each Hi-C file is opened only once since 'chr' file is sorted by chromosomes and therefore all corresponding Hi-C scores for each combination of row and column offset values can be obtained in one open file function call.

Sub CalcMidPosition {
chrspan=1000;
end = (column_offset * $chrspan)-1;
start = end - (chrspan-1);
cpos = (end+start)/2;
return cpos;
}
	Calcuate mid position of Offset. Offsets are in the range of 1000 basepairs. Start and end values for offsets are calculated and a midpoint for the offset is obtained and returned as function value.

Sub GetFilePosition {
readpos=Offset midpoint of chr1/chr2
colspan =1000000; #hi-c file span by million
pos = 0;
if (readpos < colspan)
{
pos = 1;
}
else
{
pos = (readpos/colspan) + 1;
}
return pos ;
}
	Hi-C files are in 1million basepair range. Get corresponding Hi-C file positions based on Offset midpoints (CalcMidPosition function) and return the corresponding row/column position for Hi-C file

Sub GetHi-CScore {
@cline= Split @filearray on rowpos ;
Score=cline[colpo];
Return score;
}
	Scans the @filearray (the entire Hi-C file for a chromosome row-column combination) to get the exact row and column position for Hi-C score and return score

Sub CalcRange
{
chrspan = 1000;
r1end = (r1col * chrspan)-1;
r1start = (r1end-(chrspan-1);
r2end = (r2col * chrspan)-1;
	Calculates start and end positions of offset1 and offset2

```
        r2start = r2end-(chrspan-1);

}
```

The output from this program is as represented in Table 4.15 that is presented as the final output to the user. Further, the algorithm filtered out any Hi-C scores below zero and only reported translocations with positive Hi-C probability scores.

Table 4.15: Example 'score' file from get_HiC-score algorithm

Chromosome1	Start	End	Chromosome2	Start	End	No. of reads	Hi-C Score
chrX	0	999	chrX	1000	1999	2	1
chrX	4000	4999	chrX	9000	9999	3	1
chr1	5170000	5170999	chrX	4620000	4620999	2	0.050044
chr1	5193000	5193999	chrX	17311000	17311999	2	0.018999
chr1	5236000	5236999	chrX	2850000	2850999	8	0.061457
chr1	16000	16999	chr20	45680000	45680999	3	0.089301
chr1	23000	23999	chr20	63000	63999	2	0.359193

4.8 Proof of Concept

BLAT provides various input options to the user that can change the output significantly based on the user requirements. In our case, since we were looking for regions which share 95% sequence identity and can therefore give more than one significant hit, we wanted to keep the *minIdentity* option as 95%. As a proof of concept we created a single translocation in chromosomes 4 and 8 (t4;8)(p16.3;p23.1) with breakpoint of chromosome 4 at chr4: 4086365 and on chromosome 8 at chr8: 6973436 (Figure 4.8).

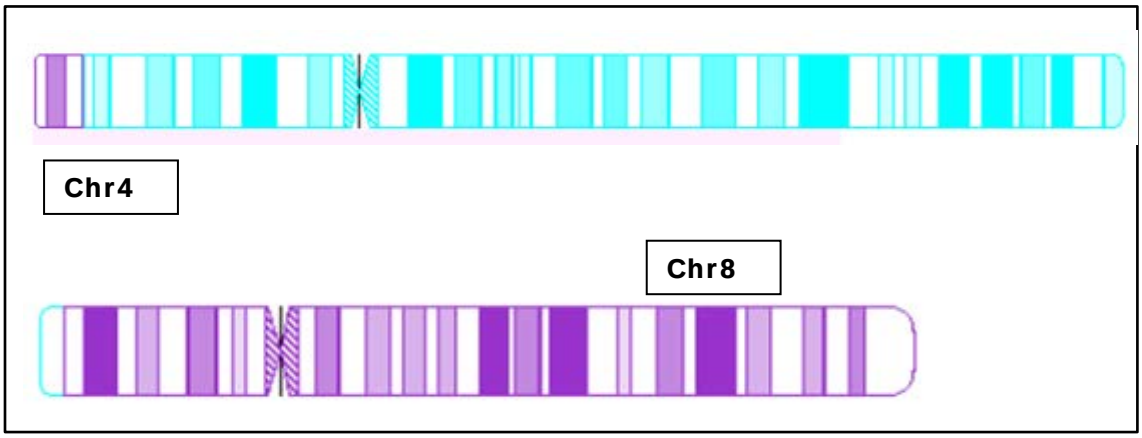


Figure 4.8: Translocation (4;8); derivative chromosomes created using (Hiller, Bradtke, Balz, & Rieder, 2005).

The simulation data had derivative chromosomes 4 and 8 as well as normal chromosome 4 and 8. This was created using the perl script provided by Dr Hayes for inserting manufactured translocations in the normal human genome. This mini FASTA file containing normal chromosome 4 and 8 and the derivative chromosome 4 and 8 was mapped to reference genome hg19 (UCSC) using BWA-0.5.9. The BAM file from this alignment was used as input for extraction of anomalous reads (Section 4.1). Since this simulated data only had chromosomes 4 and 8, the database file for

BLAT was also only created for chromosome 4 and 8. BLAT was run on FASTA output file obtained from the extraction algorithm applied to the BWA-alignment BAM file.

There were 2,441,572 anomalous reads identified from the extraction step. We used two BLAT parameters, *minMatch* and *minIdentity* to test for BLAT output. These are defined in the BLAT manual (Appendix B) as below:

-tileSize=N sets the size of match that triggers an alignment, usually between 8 and 12. Default is 11 for DNA and 5 for protein.

-minMatch=N Sets the number of tile matches. Usually set from 2 to 4. Default is 2 for nucleotide, 1 for protein.

-minIdentity=N Sets minimum sequence identity (in percent).Default is 90 for nucleotide searches, 25 for protein or translated protein searches.

We kept the default tile size of 11 for this analysis and adjusted the other two parameters. The output from BLAT is summarized in Table 4.16.

Table 4.16: BLAT parameter adjustment results

Parameter	Number of alignments	Number of translocation reported	Number of reads supporting
Defaults: minIdentity=90 minMatch=2	15111989	339887	3
minIdentity=95 minMatch=2	7482820	141527	2
minIdentity=99 minMatch=2	1073276	8733	0
minIdentity=95 minMatch=4	6620511	118282	0
minIdentity=99 minMatch=4	934711	8166	0

Testing the algorithm with a simulated single translocation (t(4;8)) dataset showed that adjusting BLAT parameters to 95% identity and minimum tile match of two, captured the positive translocations while reducing false positive by almost a third (Table 4.16). Therefore these BLAT options were used in the final simulated dataset analysis. When tightening the alignment option to report 99% identity many false positives are reduced though at the cost of true positives. Since the aim was to detect these events which are most likely to be missed by BWA alignment tool and since 95% identity is the criteria for a non-allelic homologous recombination substrate, we used this option at the cost of reporting many false positives.

5. SIMULATED DATA ANALYSIS

5.1 Creating Simulated Dataset

The idea for creating a simulated dataset was to include known documented translocations that were biologically plausible and be able to test the algorithm on this dataset. An attempt to define drivers of translocation based on genomic architecture was done by Ou *et al.*, (Ou et al., 2011) whereby they demonstrated recurrent translocation driven by non-allelic homologous recombination in unrelated families. They also mapped the regions which could be involved in potential translocation computationally using low-copy repeat regions in the genome which shares >94% sequence identity and more than 5kb in length. They were able to experimentally verify the computationally predicted translocations for the three regions identified in Table 5.1.

Burrow *et al.* analyzed recurrent translocation in cancers from various databases and tried to define the characteristics of these translocations. They found that over 50% of the recurrent translocations mapped to fragile sites, defined as regions on the genome that show multiple gaps (Burrow, Williams, Pierce, & Wang, 2009). We tried to derive our list of translocations from this comprehensive list (Appendix C) that could be near NAHR regions and found seven listed in Table 5.1. This list of translocation breakpoints was used to create derivative chromosomes using the translocation perl script mentioned in section 4.8.

Table 5.1: Translocation list used in creating simulated dataset

Translocation from Burrow et al paper	Breakpoint First chr.	Breakpoint Second chr.	Gene First chr	Gene Second chr.
t(1;22)(1q21;22q11)	142749690	19819306	BCL9	IGL@
t(12;13)(12q14;13q13)	50787223	40914700	HMGA2	LHFP
t(19;22)(19q13;22q11)	63789868	49571663	IGL@	BCL3
t(12;16)(12p13;16p13)	8270437	5069858	LAG3	MYH11
t(16;21)(16q24.3;21q22.12)	88815835	46924874	RUNX1	CBFA2T3
t(9;14)(9p21;14q11)	27286005	42868270	TRA@	CDKN2A
t(7;11)(7q34;11p15)	128044540	124564	TRB@	LMO1
Translocation from Ou et al. paper (Ou et al., 2011)	Breakpoint First chr.	Breakpoint Second chr.	Gene First chr	Gene Second chr.
t(4;8)(4p16.3;8p23.1)	4088911	6992273	NA	NA
t(4:11)(4p16.2;11p15.4)	3852863	3569449	NA	NA
t(8;12)(8p23.1;12p13.31)	6992273	8367384	NA	NA

This simulated dataset included normal chromosomes and the derivative translocated chromosomes. The FASTA file was used to create simulated data using wgsim program that is a part of SAMTOOLS suite. A description of the options used for this program is presented in Table 1.1.

Table 5.2: Simulation read creating program-'wgsim' options

Options	Type	Description
e	FLOAT	base error rate [0.020]
d	INT	outer distance between the two ends [500]
s	INT	standard deviation [50]
N	INT	number of read pairs [1000000]
1	INT	length of the first read [70]
2	INT	length of the second read [70]
r	FLOAT	rate of mutations [0.0010]
R	FLOAT	fraction of indels [0.10]
X	FLOAT	probability an indel is extended [0.30]
c	NA	generate reads in color space (SOLiD read
C	NA	show mismatch info in comment rather than
h	NA	haplotype mode

At the very least, it requires input FASTA file (translocation FASTA file) and length of the reads. In order to get 30X coverage of the genome we needed to define the number of read pairs needed (-N, Table 1.1). Coverage is calculated using equation 1 below:

$$(1)\ Coverage = \frac{Length\ of\ read \times Number\ of\ reads}{Haploid\ Genome\ Length}$$

Thus for 30X coverage and read length of 100 basepairs, we calculated the number of read required for the haploid genome of length $3x10^9$ to be 900 million reads. Output from this program gives two FASTQ files, one each for a paired end library. Further, a random subsample of this dataset was produced to simulate 15X coverage. BWA mapping tool was used to produce the alignment BAM files. The BAM file was then run against current tools, namely SVDetect-0.7f and BreakDancer-1.1. The process flow for our algorithm was as described in Figure 5.1.

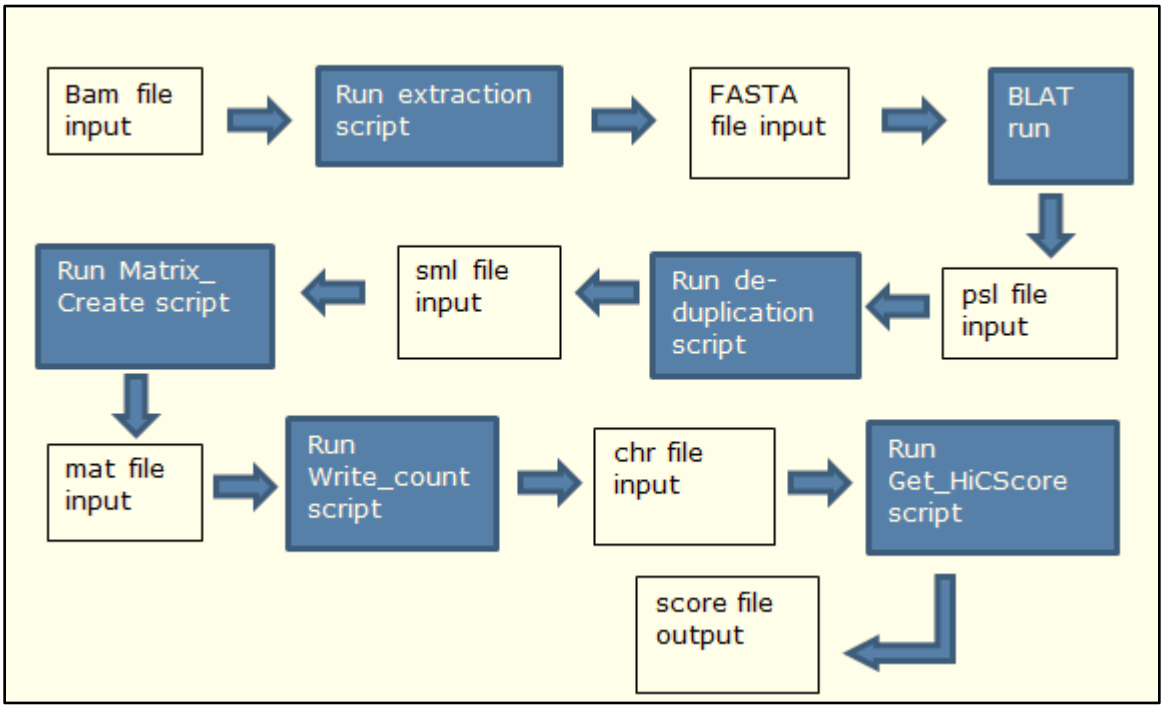


Figure 5.1. Process flow of novel algorithm

5.2 Simulated Data Analysis Results

SVDetect could correctly identify four out of the ten translocations created (4/10) and BreakDancer detected six out of ten (6/10) briefly summarized in Table 5.3.

Table 5.3: Comparison of current tools with novel algorithm

Translocation from Burrow et al paper	SVDetect	BreakDancer	Novel Algorithm
t(1;22)(1q21;22q11)	**Detected**	No	**Detected**
t(12;13)(12q14;13q13)	**Detected**	**Detected**	**Detected**
t(19;22)(19q13;22q11)	No	No	**Detected**
t(12;16)(12p13;16p13)	**Detected**	**Detected**	**Detected**
t(16;21)(16q24.3;21q22.12)	No	No	**Detected**
t(9;14)(9p21;14q11)	**Detected**	**Detected**	**Detected**
t(7;11)(7q34;11p15)	No	No	**Detected**
Translocation from Ou et al. paper (Ou et al., 2011)	SVDetect	BreakDancer	Novel Algorithm
t(4;8)(4p16.3;8p23.1)	No	**Detected**	**Detected**
t(4:11)(4p16.2;11p15.4)	No	**Detected**	**Detected**
t(8;12)(8p23.1;12p13.31)	No	**Detected**	**Detected**

The novel algorithm was able to detect all the translocations at a 1000 base pair resolution. The number of reads supporting a translocation varied from 3 to 50. The three translocations that were not detected by either SVDetect or BreakDancer, t(19;22), t(16;21) and t(7;11), all had less than 10 reads supporting the translocation in the novel algorithm output. The score file output is represented in Table 5.4.

Table 5.4: Final score file output of novel algorithm 30X coverage

Chr_1	Start	End	Chr_2	Start	End	No_ of reads	Pearsons Corr-elation coeff icient
chr1	142749000	142749999	chr22	19819000	19819999	3	0.01779
chr12	50786000	50786999	chr13	40914000	40914999	3	0.00739
chr12	50787000	50787999	chr13	40914000	40914999	17	0.00739
chr19	63789000	63789999	chr22	49571000	49571999	4	0.30154
chr12	8270000	8270999	chr16	5069000	5069999	29	0.28795
chr16	88815000	88815999	chr21	46924000	46924999	9	0.42250
chr9	42867000	42867999	chr14	27285000	27285999	15	0.25198
chr9	42868000	42868999	chr14	27285000	27285999	23	0.25198
chr9	42868000	42868999	chr14	27286000	27286999	23	0.25198
chr7	128044000	128044999	chr11	124000	124999	6	0.25539
chr4	4088000	4088999	chr8	6991000	6991999	2	0.21970
chr4	4088000	4088999	chr8	6992000	6992999	15	0.21970
chr4	3852000	3852999	chr11	3569000	3569999	23	0.1985
chr8	6991000	6991999	chr12	8367000	8367999	7	0.24114
chr8	6992000	6992999	chr12	8367000	8367999	51	0.24114

SVDetect, although correctly identified chromosome 1 breakpoint for translocation t(1;22) did not identify chromosome 2 breakpoint as precisely and had very few reads supporting this translocation (Table 5.5). BreakDancer did a very good job at breakpoint resolution but could only detect six out of ten (Table 5.6) while SVDetect picked four (Table 5.5).

Table 5.5: SVDetect output (trimmed): simulated data 30X coverage

SV_type	BAL_type	chromosome1	chromosome2	no_pairs	final_score	breakpoint1	breakpoint2
INV_TRANSLOC	UNBAL	chr1	chr22	3	1	14274984-142750363	19960253-19960750
INV_TRANSLOC	UNBAL	chr1	chr22	1	1	14274900-142749572	19958766-19959329
TRANSLOC	UNBAL	chr1	chr22	1	1	14274981-142750378	22001986-22002549
INV_TRANSLOC	UNBAL	chr12	chr13	1	1	50786416-50786979	40903686-40904249
TRANSLOC	UNBAL	chr12	chr16	24	0.83	8270917-8271106	5069215-5069396
INV_TRANSLOC	UNBAL	chr9	chr14	36	0.88	42867615-42867796	27285344-27285586

Table 5.6: BreakDancer output: simulated data 30X coverage

Chromosome1	Pos1	Chromosome2	Pos2	Type	Size	Score	Number of Reads
chr4	3852787	chr11	3569702	CTX	-499	99	4
chr4	3853180	chr11	3569260	CTX	-499	99	6
chr4	4088565	chr8	6992324	CTX	-499	99	2
chr4	4088649	chr8	6992415	CTX	-499	99	2
chr8	6992535	chr12	8367239	CTX	-499	99	2
chr8	6992629	chr12	8367239	CTX	-499	99	3
chr8	6991970	chr12	8367520	CTX	-499	99	3
chr8	6992195	chr12	8367520	CTX	-499	99	3
chr8	6992195	chr12	8367640	CTX	-499	99	5
chr9	42868057	chr14	27285682	CTX	-499	99	19
chr12	8270277	chr16	5069881	CTX	-499	99	3
chr12	8270277	chr16	5069996	CTX	-499	99	5
chr12	8270277	chr16	5070133	CTX	-499	95	2
chr12	8270357	chr16	5070133	CTX	-499	99	3
chr12	8270471	chr16	5069481	CTX	-499	99	3
chr12	8270742	chr16	5069481	CTX	-499	98	2
chr12	8270742	chr16	5069585	CTX	-499	99	11
chr12	50787289	chr13	40914358	CTX	-499	99	3
chr12	50787452	chr13	40914358	CTX	-499	99	8

As described, cancer genome complexity is overburdened further by factors like sample collection, tumor heterogeneity (Ulahannan, Kovac, Mulholland, Cazier, & Tomlinson, 2013), and platform specific issues like AT-rich and GC-rich bias in the Illumina platform (Metzker, 2010). Even with a 30X coverage of the genome and a 100% representation of the alternative allele, the tools did not detect 40% to 60% of the simulated translocations. For the next step the dataset was randomly sampled so that 50% of the reads were picked to do analysis on a 15X coverage dataset. This is more realistic for current whole-genome sequencing strategies of cancer. 30X or greater coverage is usually reserved for exome-sequencing mainly due to cost constraints. Even if a cancer genome is sequenced at 30X or greater, the inherent intercellular heterogeneity coupled with the aneuploidy that typifies most tumors means that a given translocation may be represented by even fewer supporting reads for the regions of interest.

Even with 15X coverage, our novel algorithm tool was able to detect all ten simulated translocations (Table 5.7) albeit with fewer reads supporting the translocations (compare with Table 5.4).

Table 5.7: Final score file output of novel algorithm 15X coverage

Chr_1	Start	End	Chr_2	Start	End	No_ of reads	Pearsons Corr-elation coeff icient
chr1	142749000	142749999	chr22	19819000	19819999	2	0.01779
chr12	50786000	50786999	chr13	40914000	40914999	2	0.00739
chr12	50787000	50787999	chr13	40914000	40914999	7	0.00739
chr19	63789000	63789999	chr22	49571000	49571999	3	0.30154
chr12	8270000	8270999	chr16	5069000	5069999	14	0.28795
chr16	88815000	88815999	chr21	46924000	46924999	5	0.42250
chr9	42867000	42867999	chr14	27285000	27285999	10	0.25198
chr9	42868000	42868999	chr14	27285000	27285999	8	0.25198
chr9	42868000	42868999	chr14	27286000	27286999	14	0.25198
chr7	128044000	128044999	chr11	124000	124999	3	0.25539
chr4	4088000	4088999	chr8	6992000	6992999	11	0.21970
chr4	3852000	3852999	chr11	3569000	3569999	9	0.1985
chr8	6991000	6991999	chr12	8367000	8367999	4	0.24114
chr8	6992000	6992999	chr12	8367000	8367999	28	0.24114

BreakDancer was only able to detect four (4/10) translocations with 15X coverage (Table 5.8).

Table 5.8: BreakDancer output: simulated data 15X coverage

Chrom osome1	Pos1	Chrom osome2	Pos2	Type	Size	Score	Number of Reads
chr4	3852745	chr11	3569702	CTX	-499	99	3
chr4	3853180	chr11	3569294	CTX	-499	99	4
chr8	6992049	chr12	8367520	CTX	-499	99	2
chr9	42867966	chr14	27285748	CTX	-499	99	5
chr9	42868047	chr14	27285748	CTX	-499	99	2
chr12	8270471	chr16	5069481	CTX	-499	99	2
chr12	8270566	chr16	5069585	CTX	-499	99	2

SVDetect was only able to detect two (2/10) variants (Table 5.9).

Table 5.9: SVDetect output (trimmed): simulated data 15X coverage

SV_type	BAL_type	Chr_1	Chr_2	nb_pairs	final_score	breakpoint1	breakpoint2
INV_TRANSLOC	UNBAL	chr1	chr22	1	1	142749572-142749671	19959329-19959428
TRANSLOC	UNBAL	chr12	chr13	4	0.8	50786565-50786795	40915026-40915369
TRANSLOC	UNBAL	chr12	chr13	4	0.8	50787552-50788028	40913925-40914365

One of the major issues with the novel algorithm is that it gives large number of false positives despite filtering out variants with negative Hi-C scores. Therefore distribution-based filtering is another approach to reduce false positives. Frequency distribution of Hi-C data for chromosome 1 and chromosome 22 showed most of the regions (1 Mbps windows) with Pearson's correlation coefficient between 0.03 and 0.09 (Figure 5.2).

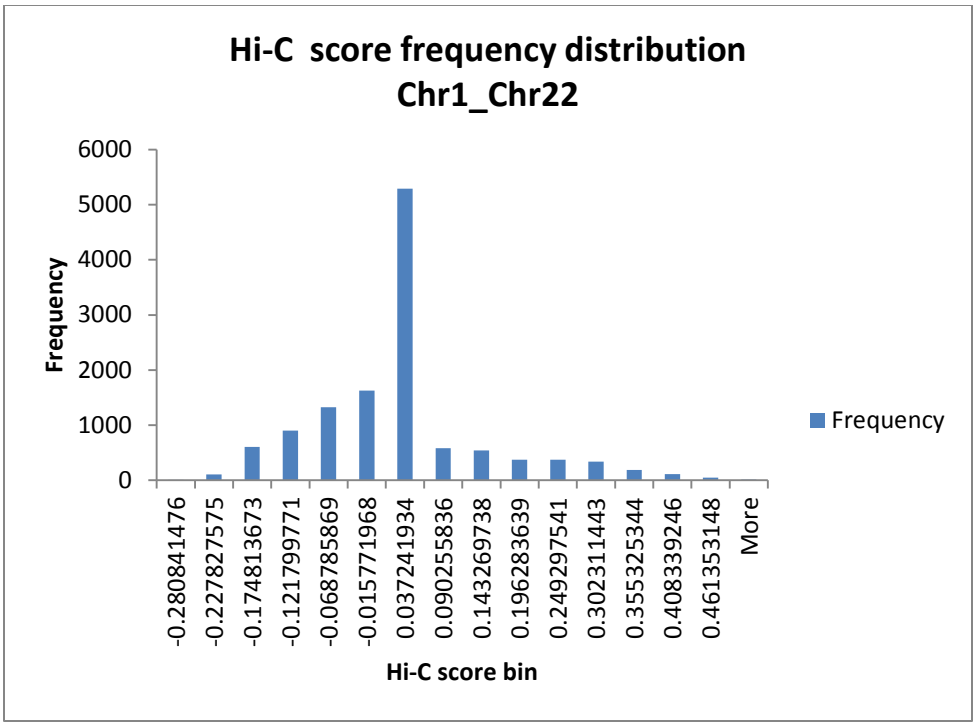


Figure 5.2: Hi-C score frequency distribution for chromosome 1 and 22

Setting the filtering cutoff at 0.09 would make the tool miss this translocation while setting it at the tail end at 0.249 will capture it (Table 5.4).

Frequency distribution of chromosome 12 and chromosome 13 (Figure 5.3) with cutoff above zero will include the translocation (Table 5.4).

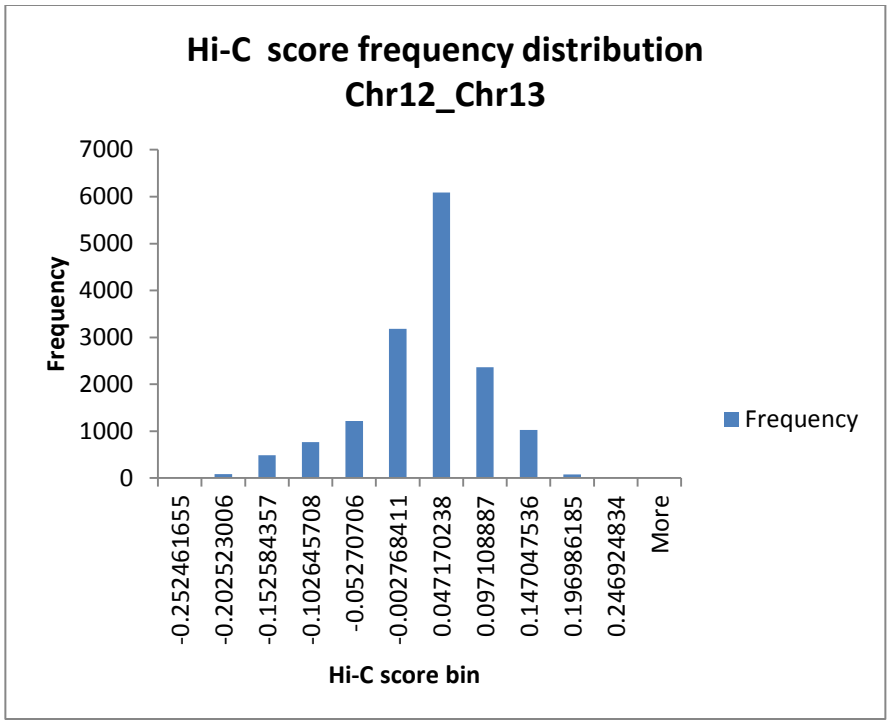


Figure 5.3: Hi-C score frequency distribution for chromosome 12 and 13

Similarly to capture t(12;16), t(19;22), t(16;21), t(9;14), t(7;11), t(4;8), t(4;11), and t(8;12) we need to include the far right end of the distribution (Figure 5.4, Figure 5.5, Figure 5.6, Figure 5.7, Figure 5.8, Figure 5.9, Figure 5.10, Figure 5.11). Thus, although these translocations had a strong Pearson's correlation value, filtering based on distribution will lose these translocations. A filtering method to filter out variants which are one standard deviation away from the mean in the positive direction will be a better approach to capture these events.

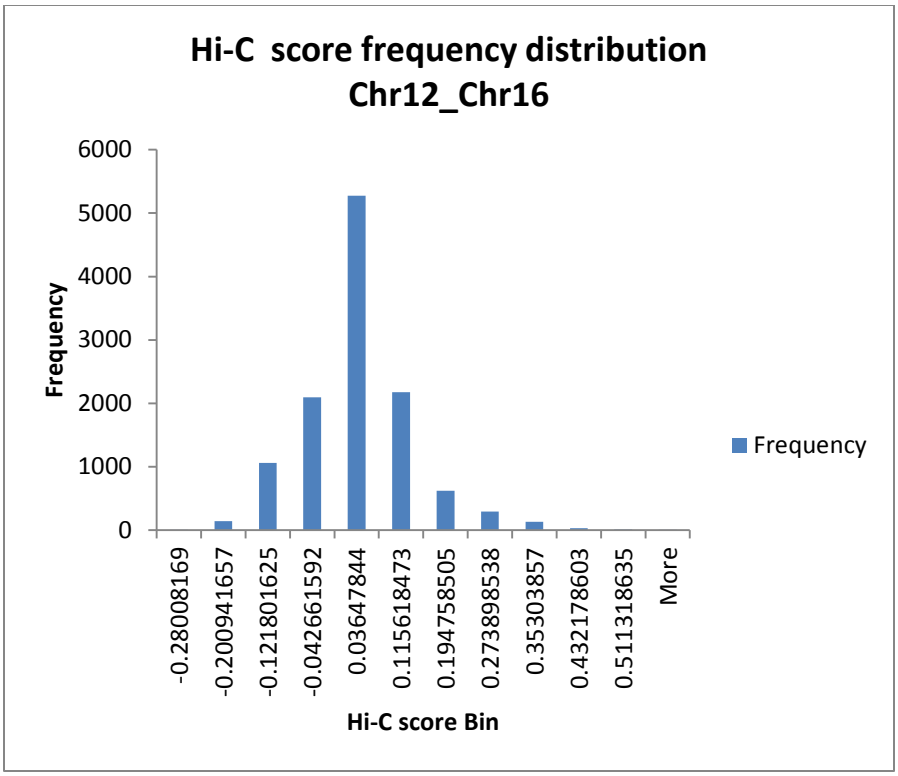


Figure 5.4: Hi-C score frequency distribution for chromosome 12 and 16

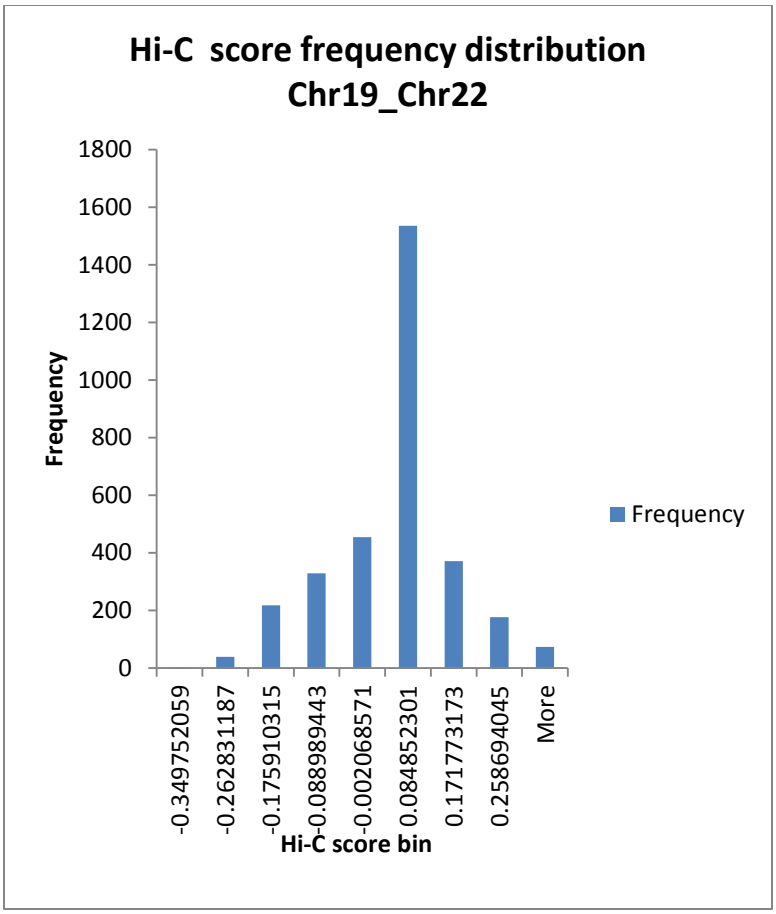


Figure 5.5: Hi-C score frequency distribution for chromosome 19 and 22

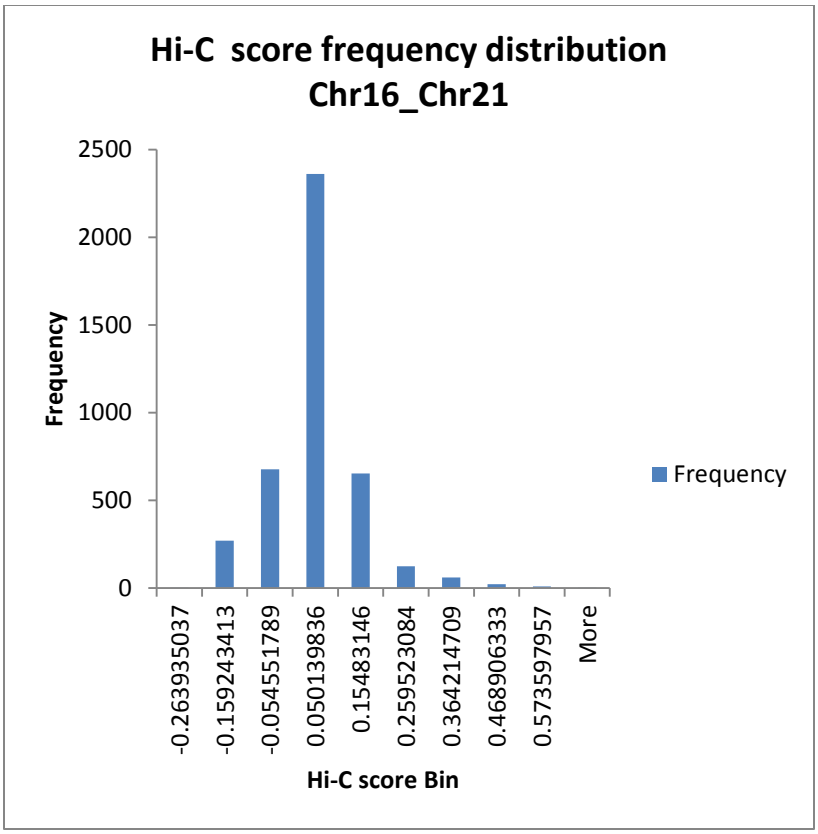


Figure 5.6: Hi-C score frequency distribution for chromosome 16 and 21

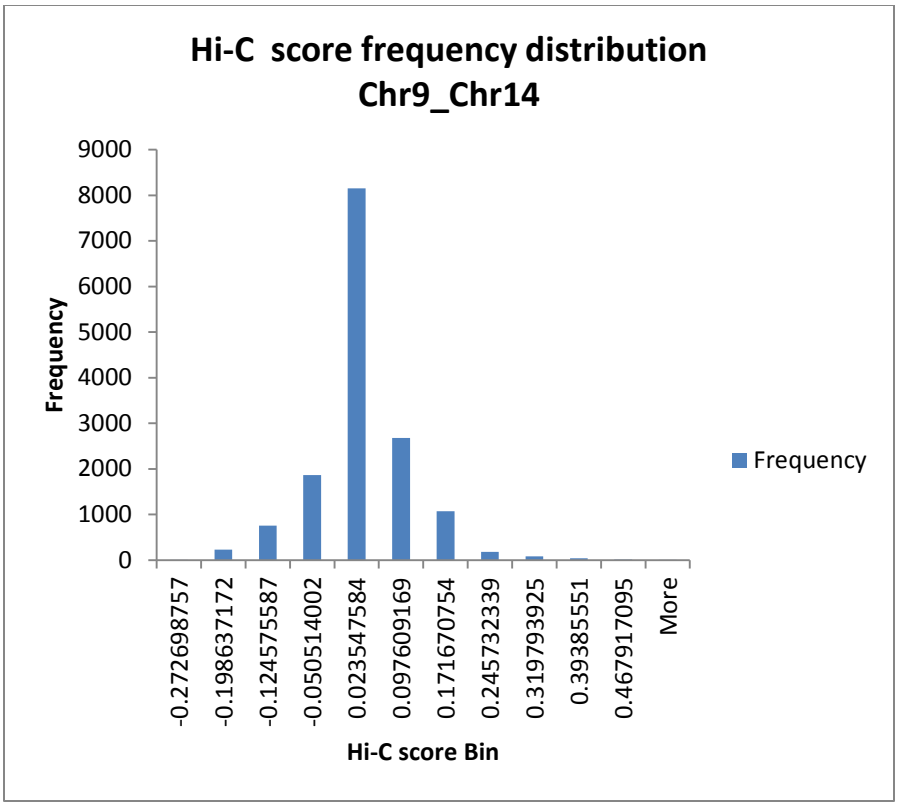


Figure 5.7: Hi-C score frequency distribution for chromosome 9 and 14

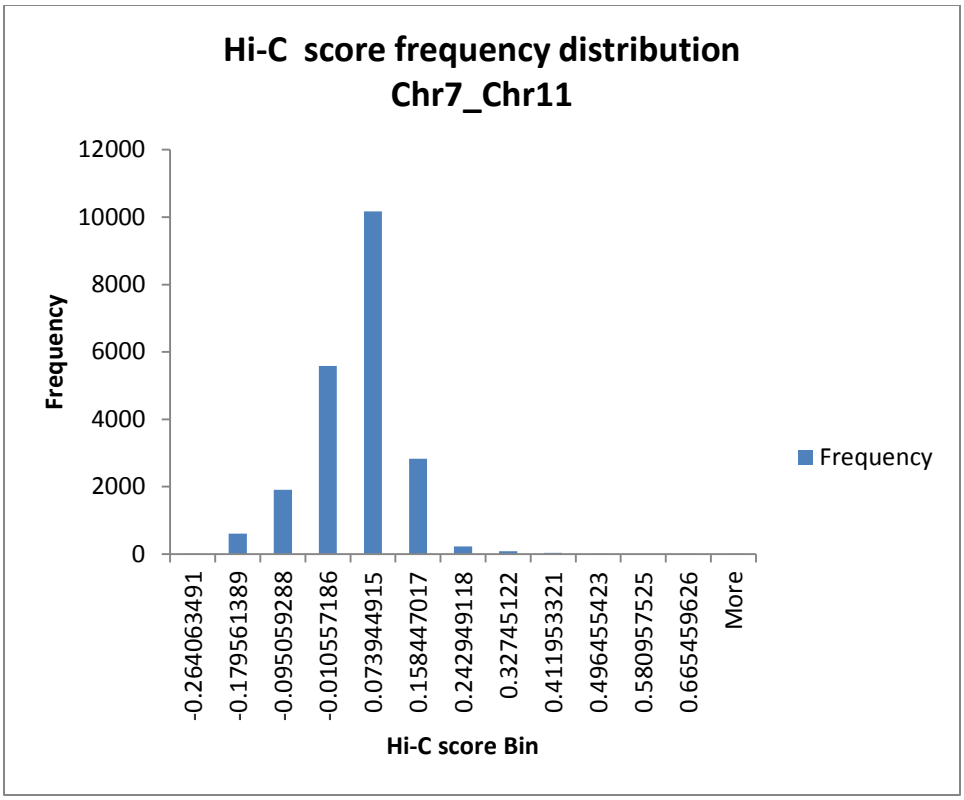


Figure 5.8: Hi-C score frequency distribution for chromosome 7 and 11

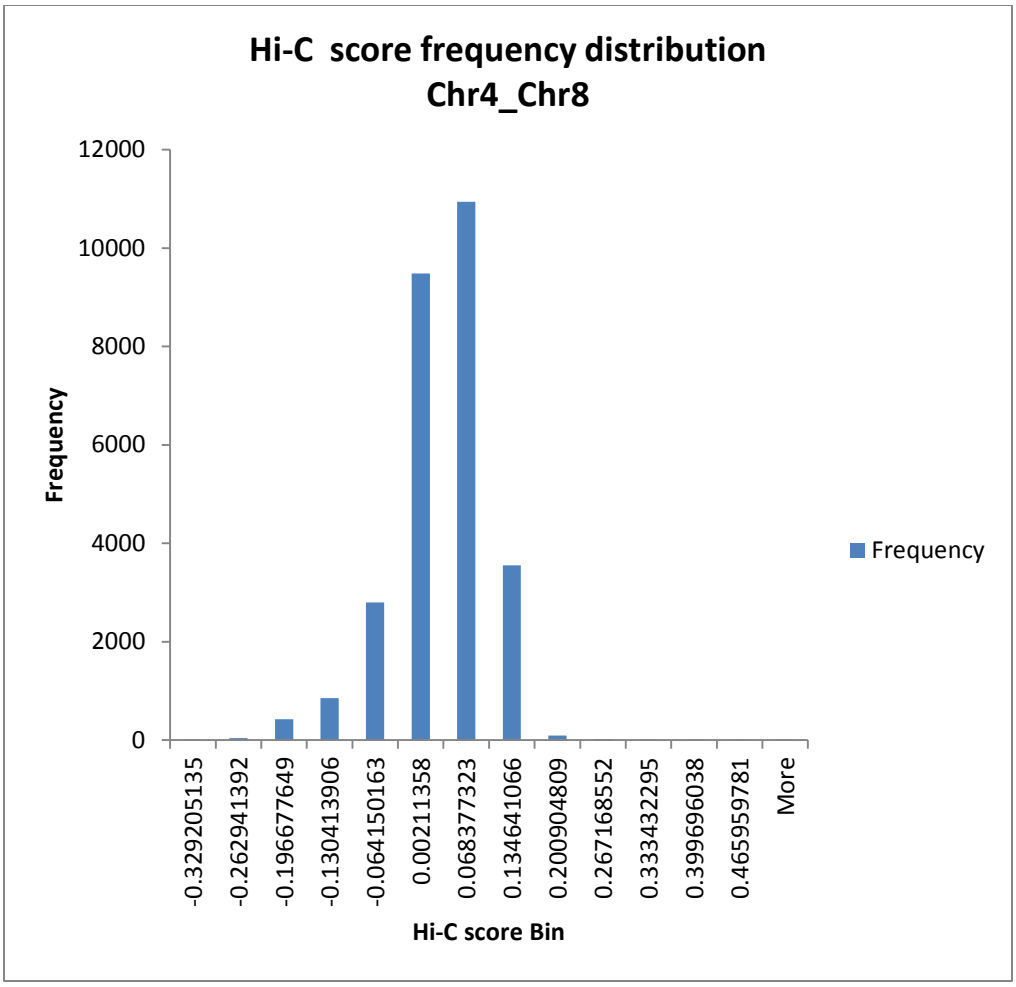


Figure 5.9: Hi-C score frequency distribution for chromosome 4 and 8

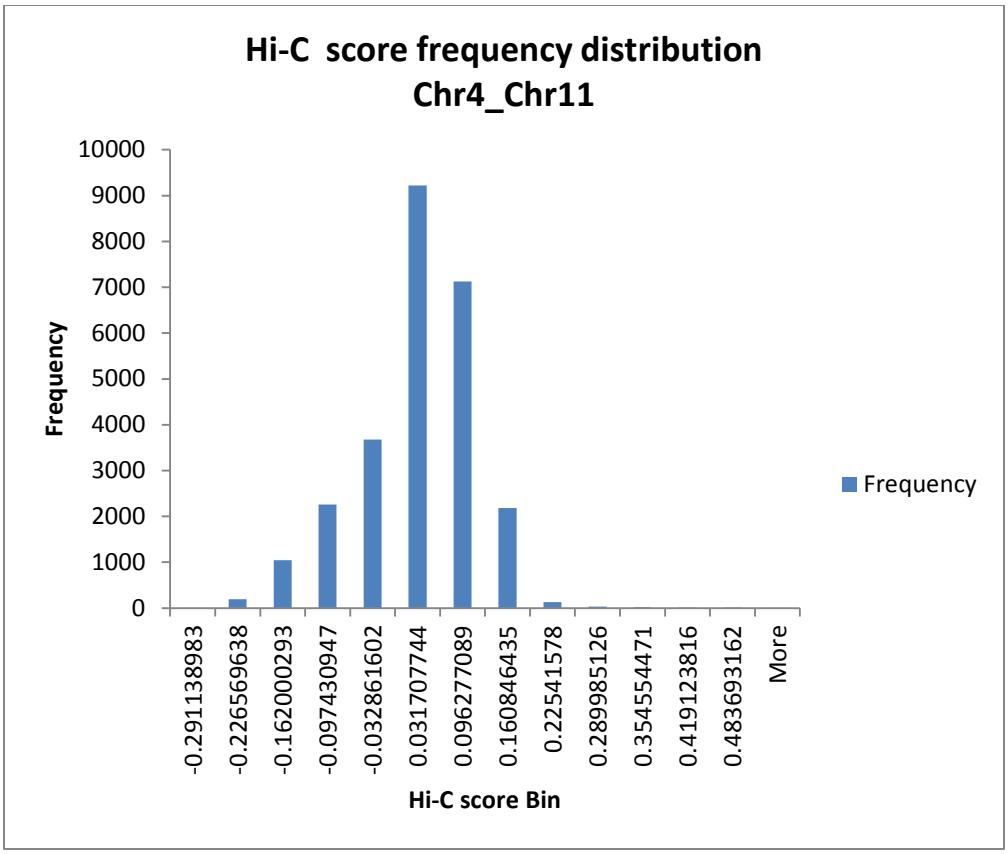


Figure 5.10: Hi-C score frequency distribution for chromosome 4 and 11

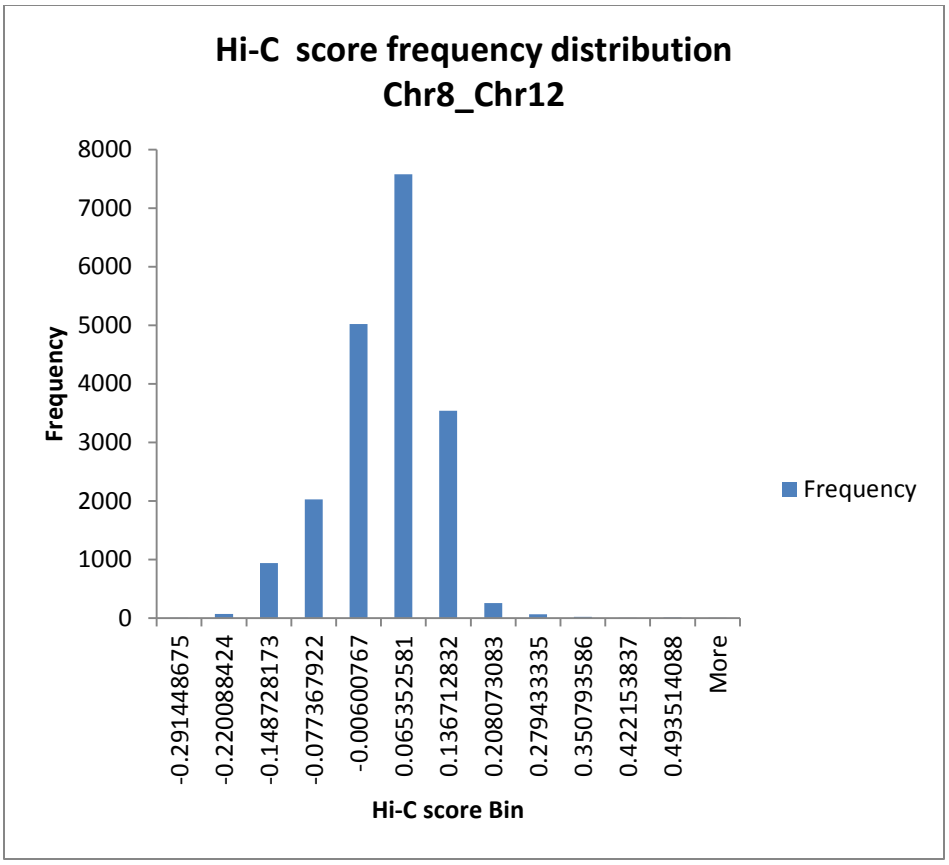


Figure 5.11: Hi-C score frequency distribution for chromosome 8 and 12

Thus in order to reduce noise, filtering variants with negative correlation coefficients is an essential first step. Further, arriving at a cutoff by looking at the distribution of Hi-C scores to filter variants with strong positive correlation can further reduce noise and fine-tune the tool.

6. DISCUSSION

The aim of this study was to understand if characteristics of the structural variant made it more or less conducive to detection by current computational methods in use. We wanted to understand the structural variant in relation to its position in the genome and determine if this genomic context made some variants more difficult to be detected by current tools. The human genome is made up of approximately 20,000-25,0000 genes and targeted sequencing of only these known coding regions covers about 1% of the 3 billion bases of the human genome, *i.e.* approximately 30 million (Brunham & Hayden, 2013). Applying genome linkage analysis to single-gene Mendelian disease has met with considerable success as in cystic fibrosis (Kerem et al., 1989; Rommens et al., 1988) and Huntington's disease (Fox, Bloch, Fahy, & Hayden, 1989). Application of next generation sequencing efforts to Mendelian disorders has led to discovery of more than 3000 genes associated with a phenotypically visible trait and more than 5000 phenotypic traits with known molecular basis as recorded to date in the Online Mendelian Inheritance in Man (OMIM database) (McKusick, 2014). Discovery of these specific genes has tremendous value in the predictive capability of genetic diagnosis through biomarkers as in Alzheimer as well as targeted gene therapy as in cystic fibrosis with the mean survival rate of cystic fibrosis going from 6 months in 1930's to 40 years in 1990's (Ikpa, Bijvelds, & de Jonge, 2014) due to novel therapy. The impact of genotype-to-phenotype translation, while more obvious in Mendelian diseases, are not so easily translated in complex disease with possible multi-gene etiology. Translational bioinformatics tries to fill up this gap using high throughput data analysis techniques. Cancers are even more complicated due to presence of somatic variants that are not inherited and therefore more difficult to define. Further, tumors are highly heterogeneous, and therefore the presence of a variant will be highly

dependent on sample extraction and preparation. Thus, the aim is to identify the best approach to detect novel variants in a heterogeneous sample with highly mutated genome. The currently available tools were designed around the normal genome architecture assuming 100% allele frequency. Cancer genome does not conform to this assumption.

6.1 Repeat Analysis

The study analysis started with the hypothesis that the tools were failing to detect variants due to presence of these variants in repeat regions. If the mapping tools are unable to map reads uniquely, the tools will not be able to pick these variants. The first analysis for this study tried to define the characteristics of the validated variants in 1000 genomes trio dataset using current variant detection algorithms. The purpose was to understand the reason these tools were missing to detect the validated variants. SVDetect was chosen for this analysis due to its high false positive rate and thus be highly sensitive. Since the idea was to define context, true positive was defined as those variants detected by SVDetect with at least 10% of insert size overlap with variants in the 1000-genome validated file. Although less than 50% of the true variants had some overlap with the variants detected by the tools, their representation in the repeat regions was not as hypothesized. The hypothesis that structural variants missed by the tools were more likely to be in repeat regions was rejected by the analysis. A greater percent of the variants detected by the tools had repeats compared to variants not detected by the tools. This was consistent across all four variant types, deletion, mobile element insertion, tandem duplication, and novel sequences classified in the 1000-Genomes dataset. Repeat structure was not driving the tool's inability to pick less than 50% of true variants. Since 50% of the human genome is made up of repeats, this very broad classification of context did not prove very useful. Since the study was trying to

design a tool specific for translocation, the context was now focused towards mechanisms driving translocation which can be captured informatically.

6.2 Algorithm Development and Simulated Data Analysis

Translocations lead to genetic imbalances and are a precursor to cancers. Detection of the same recurrent translocation in four unrelated families by Ou *et al*. (Ou et al., 2011) led to implication of non-allelic homologous recombination (NAHR) as the driver for these variants in all these subjects. NAHR occurs due to aberrant DNA repair mechanism between regions that share considerable homology, also known as low copy repeat regions (LCRs). Unlike repeat elements, these LCRs are several thousand basepairs long and share greater than 95% identity. Using these characteristics, Ou *et al*. computationally mapped the NAHR regions on the human genome and predicted validated translocations in their database. This mapping was based on the segmental duplication map created using comparative genomic hybridization which identified novel structural variants in these regions of LCRs by Sharp *et al*. (Sharp et al., 2006). Segmental duplications as possible hotspots for structural variation events were first hypothesized and mapped by Bailey *et al*. (Bailey et al., 2002) identifying 169 such regions in the human genome. However, the physical co-location of these LCR regions in cell is also an important contributing factor to the actual interaction between these regions. Regions of chromosomes that are physically close to each other in 3-dimensional space are more likely to interact with each other as proved by chromosomal conformation capture experiments known as (3C or Hi-C) which reveal three-dimensional architecture of genome packing in the cell (Wijchers, 2011). The mobility of different regions in the genome is limited by the location of these regions in the genome (Chubb, Boyle, Perry, & Bickmore, 2002). While accounting for these two types of important contextual based information about translocation etiology, the study was able to design and test the

algorithm based on this context and proved to do better than current methodologies for translocation detection.

The algorithm was designed to capture all reads that would be ambiguously aligned by the mapping tool. The most popular mapping tool currently in use, namely BWA (Li & Durbin, 2009), was used for generating initial mapping. BWA's popularity is due to its speed in mapping billions of shorts reads in hours, achieved by its effective use of cache memory in indexing the reads and also wide acceptance in the bioinformatics community as a preferred mapping tool. The output from BWA is also in the SAM/BAM format (Li et al., 2009) accepted as the standard output format for alignment. BWA randomly assigns reads which map to more than one region on the genome. In their simulated read alignment, BWA mapped 11/1,569,108 incorrectly (Li & Durbin, 2009), which still is a very low error rate but in a sequencing experiment with for example 50 million reads, approximately 500 reads would be wrongly assigned. While this may be sufficient for a normal genome, in a cancer genome sequencing project aimed at finding novel variants and showing heterogeneity, this number might make a difference. Analysis in this study showed that even with 100% allele frequency, which is not the case in tumors; the tools were missing variants in the simulated dataset. The tools pick up anomalous reads based on these reads mapping at greater/shorter distance than the normal distribution of insert size or incorrect orientation. Our novel algorithm does not use the probability distribution of insert size to pick reads. This is because reads with NAHR characteristics, *i.e.* reads which share greater than 95% identity, could have been placed at a location selected randomly by the mapping tool since these regions are so similar, and thus their imputed insert size is suspect. The algorithm accounted for this information by extracting reads which could have multiple mappings on the genome while extracting the partner paired read irrespective of its mapping score.

Re-mapping of these ambiguous reads with a local alignment tool like BLAT (W. J. Kent, 2002) further helps define **context**, since all possible genomic regions of identity are now reported. Further, instead of using read distribution of the data and number of reads supporting a type of variant to assign probability of calling a true variant, the algorithm again used **context** to define the probability of two regions being involved in translocation based on their known proximity to each other in 3-D cellular space and therefore the probability of interacting regions (Lieberman-Aiden et al., 2009). Even with 15X coverage, the novel tool was able to detect all ten simulated translocations.

6.3 Conclusion

Designing a **novel context based approach** to detect translocations, the study showed a very effective way to detect these variants using a biologically derived context-based approach which has not been used so far to effectively mine structural variants. The study also rejected the hypothesis that repeat structure within the variant was driving the inability of current tools to detect true positive events.

The output from this novel algorithm could help discover many de-novo variants in cancers and provide a starting point for mining variant information from sequencing data. The purpose of this tool was not to define a few variants, but to give as many possible variants that could then be teased out by the user through experimental validation. Bioinformatics data analysis of such big volume data does suffer from copious output of false positives, but at the same time is the first step in moving towards more comprehensive follow-up using laboratory tools. Providing the user with biological context-based algorithm instills more confidence in the output, which was the purpose of this analysis.

6.4 Limitations

The major limitation of this study is the number of false positive reported in output. Ideally the user would like to see only the most relevant information that is currently embedded in a lot of noise. Noise can be reduced by narrowing BLAT's extraction parameters for stricter re-alignment. How this would play out in a real dataset was not explored in this analysis.

Since re-alignment with BLAT is a computationally intensive process, the access to high performance computing environment is a pre-requisite. Without high-throughput computing the analysis can get prohibitively time consuming. Running BLAT alignments in parallel greatly reduces the time, and we assume that users would have access to parallel, high performance computing resources.

The tool was also specifically designed for detecting translocations, and users would prefer getting the entire spectrum of structural variation in a single tool, which is another limitation for this study.

6.5 Future Direction and Research

The tool was specifically designed to obtain as much information from mapping as possible to be able to derive de-novo variants that it achieved at the cost of reporting a lot of noise. I would like to explore new methods to reduce noise in the data without compromising on the mining ability of the tool. Noise reduction parameters could also include evolutionary information of conserved versus non-conserved regions to remove implausible variants. I would also like to expand the tool capability to detect all type of other structural variants including deletion, insertion and inversion.

This is the **first context-based tool** designed to date and can prove useful for helping lay the framework for further algorithm development along these lines which take other biological context into account while designing bioinformatics tools.

REFERENCES

1000 Genomes Project Consortium, Abecasis, G. R., Altshuler, D., Auton, A., Brooks, L. D., Durbin, R. M., et al. (2010a). A map of human genome variation from population-scale sequencing. *Nature, 467*(7319), 1061-1073.

1000 Genomes Project Consortium, Abecasis, G. R., Altshuler, D., Auton, A., Brooks, L. D., Durbin, R. M., et al. (2010b). A map of human genome variation from population-scale sequencing. *Nature, 467*(7319), 1061-1073.

Alkodsi, A., Louhimo, R., & Hautaniemi, S. (2014). Comparative analysis of methods for identifying somatic copy number alterations from deep sequencing data. *Briefings in Bioinformatics,*

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology, 215*(3), 403-410.

Bailey, J. A., Gu, Z., Clark, R. A., Reinert, K., Samonte, R. V., Schwartz, S., et al. (2002). Recent segmental duplications in the human genome. *Science (New York, N.Y.), 297*(5583), 1003-1007.

Bridge, J. A., & Cushman-Vokoun, A. M. (2011). Molecular diagnostics of soft tissue tumors. *Archives of Pathology & Laboratory Medicine, 135*(5), 588-601.

Brunham, L. R., & Hayden, M. R. (2013). Hunting human disease genes: Lessons from the past, challenges for the future. *Human Genetics, 132*(6), 603-617.

Buchdunger, E., Zimmermann, J., Mett, H., Meyer, T., Muller, M., Druker, B. J., et al. (1996). Inhibition of the abl protein-tyrosine kinase in vitro and in vivo by a 2-phenylaminopyrimidine derivative. *Cancer Research, 56*(1), 100-104.

Bunting, S. F., & Nussenzweig, A. (2013). End-joining, translocations and cancer. *Nature Reviews.Cancer, 13*(7), 443-454.

Burrow, A. A., Williams, L. E., Pierce, L. C., & Wang, Y. H. (2009). Over half of breakpoints in gene pairs involved in cancer-specific recurrent translocations are mapped to human chromosomal fragile sites. *BMC Genomics, 10*, 59-2164-10-59.

Chen, K., Wallis, J. W., McLellan, M. D., Larson, D. E., Kalicki, J. M., Pohl, C. S., et al. (2009). BreakDancer: An algorithm for high-resolution mapping of genomic structural variation. *Nature Methods, 6*(9), 677-681.

Chen, S., Li, S., Xie, W., Li, X., Zhang, C., Jiang, H., et al. (2014). Performance comparison between rapid sequencing platforms for ultra-low coverage sequencing strategy. *PloS One, 9*(3), e92192.

Chubb, J. R., Boyle, S., Perry, P., & Bickmore, W. A. (2002). Chromatin motion is constrained by association with nuclear compartments in human cells. *Current Biology : CB, 12*(6), 439-445.

Cock, P. J., Fields, C. J., Goto, N., Heuer, M. L., & Rice, P. M. (2010). The sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research, 38*(6), 1767-1771.

Colnaghi, R., Carpenter, G., Volker, M., & O'Driscoll, M. (2011a). The consequences of structural genomic alterations in humans: Genomic disorders, genomic instability and cancer. *Seminars in Cell & Developmental Biology, 22*(8), 875-885.

Colnaghi, R., Carpenter, G., Volker, M., & O'Driscoll, M. (2011b). The consequences of structural genomic alterations in humans: Genomic disorders, genomic instability and cancer. *Seminars in Cell & Developmental Biology, 22*(8), 875-885.

De, S., & Michor, F. (2011). DNA replication timing and long-range DNA interactions predict mutational landscapes of cancer genomes. *Nature Biotechnology, 29*(12), 1103-1108.

Deininger, P. L., & Batzer, M. A. (1999). Alu repeats and human disease. *Molecular Genetics and Metabolism, 67*(3), 183-193.

Druker, B. J., Tamura, S., Buchdunger, E., Ohno, S., Segal, G. M., Fanning, S., et al. (1996). Effects of a selective inhibitor of the abl tyrosine kinase on the growth of bcr-abl positive cells. *Nature Medicine, 2*(5), 561-566.

Eddy, S. R. (2004). What is dynamic programming? *Nature Biotechnology, 22*(7), 909-910.

Estevezj. (2012). *Sanger-sequencing.* Retrieved 03/27, 2014, from http://commons.wikimedia.org/wiki/File:Sanger-sequencing.svg

Ewing, B., & Green, P. (1998). Base-calling of automated sequencer traces using phred. II. error probabilities. *Genome Research, 8*(3), 186-194.

Ewing, B., Hillier, L., Wendl, M. C., & Green, P. (1998). Base-calling of automated sequencer traces using phred. I. accuracy assessment. *Genome Research, 8*(3), 175-185.

Fox, S., Bloch, M., Fahy, M., & Hayden, M. R. (1989). Predictive testing for huntington disease: I. description of a pilot project in british columbia. *American Journal of Medical Genetics, 32*(2), 211-216.

Fragouli, E., Alfarawati, S., Daphnis, D. D., Goodall, N. N., Mania, A., Griffiths, T., et al. (2011). Cytogenetic analysis of human blastocysts with the use of FISH, CGH and aCGH: Scientific data and technical evaluation. *Human Reproduction (Oxford, England), 26*(2), 480-490.

Futreal, P. A., Coin, L., Marshall, M., Down, T., Hubbard, T., Wooster, R., et al. (2004). A census of human cancer genes. *Nature Reviews.Cancer, 4*(3), 177-183.

Gu, W., Zhang, F., & Lupski, J. R. (2008). Mechanisms for human genomic rearrangements. *PathoGenetics, 1*(1), 4-8417-1-4.

Hastings, R. J., Nisbet, D. L., Waters, K., Spencer, T., & Chitty, L. S. (1999). Prenatal detection of extra structurally abnormal chromosomes (ESACs): New cases and a review of the literature. *Prenatal Diagnosis, 19*(5), 436-445.

Hiller, B., Bradtke, J., Balz, H. & Rieder, H. (2005). *CyDAS online analysis site.* Retrieved March,2014, 2014, from http://www.cydas.org/About/index.html

Ikpa, P. T., Bijvelds, M. J., & de Jonge, H. R. (2014). Cystic fibrosis: Toward personalized therapies. *The International Journal of Biochemistry & Cell Biology(2014),* doi: 10.1016/j.biocel.2014.02.008.

Inokuchi, K. (2006). Chronic myelogenous leukemia: From molecular biology to clinical aspects and novel targeted therapies. *Journal of Nippon Medical School = Nippon Ika Daigaku Zasshi, 73*(4), 178-192.

Istrail, S., Sutton, G. G., Florea, L., Halpern, A. L., Mobarry, C. M., Lippert, R., et al. (2004). Whole-genome shotgun assembly and comparison of human genome assemblies. *Proceedings of the National Academy of Sciences of the United States of America, 101*(7), 1916-1921.

Jeffreys, A. J., Kauppi, L., & Neumann, R. (2001). Intensely punctate meiotic recombination in the class II region of the major histocompatibility complex. *Nature Genetics, 29*(2), 217-222.

Kallioniemi, A., Kallioniemi, O. P., Sudar, D., Rutovitz, D., Gray, J. W., Waldman, F., et al. (1992). Comparative genomic hybridization for molecular cytogenetic analysis of solid tumors. *Science (New York, N.Y.), 258*(5083), 818-821.

Kent, P., O'Donoghue, J. M., O'Hanlon, D. M., Kerin, M. J., Maher, D. J., & Given, H. F. (1995). Linkage analysis and the susceptibility gene (BRCA-1) in familial breast cancer. *European Journal of Surgical Oncology : The Journal of the European Society of Surgical Oncology and the British Association of Surgical Oncology, 21*(3), 240-241.

Kent, W. J. (2002). BLAT--the BLAST-like alignment tool. *Genome Research, 12*(4), 656-664.

Kerem, B., Rommens, J. M., Buchanan, J. A., Markiewicz, D., Cox, T. K., Chakravarti, A., et al. (1989). Identification of the cystic fibrosis gene: Genetic analysis. *Science (New York, N.Y.), 245*(4922), 1073-1080.

Lee, H., Kim, M., Lim, J., Kim, Y., Han, K., Cho, B. S., et al. (2013). Acute myeloid leukemia associated with FGFR1 abnormalities. *International Journal of Hematology, 97*(6), 808-812.

Lee, S., Cheran, E., & Brudno, M. (2008). A robust framework for detecting structural variations in a genome. *Bioinformatics (Oxford, England), 24*(13), i59-67.

Li, H., & Durbin, R. (2009). Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics (Oxford, England), 25*(14), 1754-1760.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., et al. (2009). The sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England), 25*(16), 2078-2079.

Lieberman-Aiden, E., van Berkum, N. L., Williams, L., Imakaev, M., Ragoczy, T., Telling, A., et al. (2009). Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science (New York, N.Y.), 326*(5950), 289-293.

Lynch, T. J., Bell, D. W., Sordella, R., Gurubhagavatula, S., Okimoto, R. A., Brannigan, B. W., et al. (2004). Activating mutations in the epidermal growth factor receptor underlying responsiveness of Non–Small-cell lung cancer to gefitinib. *N Engl J Med, 350*(21), 2129-2139.

Mahadevaiah, S. K., Costa, Y., & Turner, J. M. (2009). Using RNA FISH to study gene expression during mammalian meiosis. *Methods in Molecular Biology (Clifton, N.J.), 558*, 433-444.

Mathews, V., George, B., Chendamarai, E., Lakshmi, K. M., Desire, S., Balasubramanian, P., et al. (2010). Single-agent arsenic trioxide in the treatment of newly diagnosed acute promyelocytic leukemia: Long-term follow-up data. *Journal of Clinical Oncology, 28*(24), 3866-3871.

Maxam, A. M., & Gilbert, W. (1977). A new method for sequencing DNA. *Proceedings of the National Academy of Sciences of the United States of America, 74*(2), 560-564.

McKusick, V. A. (2014). *Online mendelian inheritance in man, OMIM®. McKusick-nathans institute of genetic medicine, johns hopkins university (baltimore, MD).* Retrieved 04/01, 2014, from http://omim.org/

McNeil, N., Montagna, C., Difilippantonio, M. & Ried, T. (2012). *Comparative cancer cytogenetics. atlas genet cytogenet oncol haematol.* Retrieved 03/2013, 2014, from http://atlasgeneticsoncology.org//Deep/ComparCancerCytogID20011.html

McVean, G. (2010). What drives recombination hotspots to repeat DNA in humans? *Philosophical Transactions of the Royal Society of London.Series B, Biological Sciences, 365*(1544), 1213-1218.

McVean, G. A., Myers, S. R., Hunt, S., Deloukas, P., Bentley, D. R., & Donnelly, P. (2004). The fine-scale structure of recombination rate variation in the human genome. *Science (New York, N.Y.), 304*(5670), 581-584.

Mende, D. R., Waller, A. S., Sunagawa, S., Jarvelin, A. I., Chan, M. M., Arumugam, M., et al. (2012). Assessment of metagenomic assembly using simulated next generation sequencing data. *PloS One, 7*(2), e31386.

Metzker, M. L. (2010). Sequencing technologies - the next generation. *Nature Reviews.Genetics, 11*(1), 31-46.

Mitelman, F., Johansson, B. & Mertens, F. (.). (2014). *Mitelman database of chromosome aberrations and gene fusions in cancer.* Retrieved March/15, 2014, from http://cgap.nci.nih.gov/Chromosomes/Mitelman

Monaco, A. P., & Larin, Z. (1994). YACs, BACs, PACs and MACs: Artificial chromosomes as research tools. *Trends in Biotechnology, 12*(7), 280-286.

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology, 48*(3), 443-453.

Niu, C., Yan, H., Yu, T., Sun, H. P., Liu, J. X., Li, X. S., et al. (1999). Studies on treatment of acute promyelocytic leukemia with arsenic trioxide: Remission induction, follow-up, and molecular monitoring in 11 newly diagnosed and 47 relapsed acute promyelocytic leukemia patients. *Blood, 94*(10), 3315-3324.

Nowell, P. C., & Hungerford, D. A. (1960). Chromosome studies on normal and leukemic human leukocytes. *Journal of the National Cancer Institute, 25*, 85-109.

Nunnally, B. (2005). Introduction to DNA sequencing: Sanger and beyond. In B. Nunnally (Ed.), *Analytical techniques in DNA sequencing* (1st ed., pp. 1-11). Boca Raton,FL: Taylor&Francis Group.

Ou, Z., Stankiewicz, P., Xia, Z., Breman, A. M., Dawson, B., Wiszniewska, J., et al. (2011). Observation and prediction of recurrent human translocations mediated by NAHR between nonhomologous chromosomes. *Genome Research, 21*(1), 33-46.

Paez, J. G., Janne, P. A., Lee, J. C., Tracy, S., Greulich, H., Gabriel, S., et al. (2004). EGFR mutations in lung cancer: Correlation with clinical response to gefitinib therapy. *Science (New York, N.Y.), 304*(5676), 1497-1500.

Pevsner, J. (2009). Pairwise sequence alignment. *Bioinformatics and functional genomics* (2nd ed., pp. 47-94). Hoboken, New Jersey: John Wiley & Sons, Inc.

Przybytkowski, E., Ferrario, C., & Basik, M. (2011). The use of ultra-dense array CGH analysis for the discovery of micro-copy number alterations and gene fusions in the cancer genome. *BMC Medical Genomics, 4*, 16-8794-4-16.

Rommens, J. M., Zengerling, S., Burns, J., Melmer, G., Kerem, B. S., Plavsic, N., et al. (1988). Identification and regional localization of DNA markers on chromosome 7 for the cloning of the cystic fibrosis gene. *American Journal of Human Genetics, 43*(5), 645-663.

Rowley, J. D. (1973). Letter: A new consistent chromosomal abnormality in chronic myelogenous leukaemia identified by quinacrine fluorescence and giemsa staining. *Nature, 243*(5405), 290-293.

Sanders, S. J., Murtha, M. T., Gupta, A. R., Murdoch, J. D., Raubeson, M. J., Willsey, A. J., et al. (2012). De novo mutations revealed by whole-exome sequencing are strongly associated with autism. *Nature, 485*(7397), 237-241.

Sanger, F., Nicklen, S., & Coulson, A. R. (1977). DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences of the United States of America, 74*(12), 5463-5467.

Santarius, T., Shipley, J., Brewer, D., Stratton, M. R., & Cooper, C. S. (2010). A census of amplified and overexpressed human cancer genes. *Nature Reviews.Cancer, 10*(1), 59-64.

Sharp, A. J., Hansen, S., Selzer, R. R., Cheng, Z., Regan, R., Hurst, J. A., et al. (2006). Discovery of previously unidentified genomic disorders from the duplication architecture of the human genome. *Nature Genetics, 38*(9), 1038-1042.

Shizuya, H., & Kouros-Mehr, H. (2001). The development and applications of the bacterial artificial chromosome cloning system. *The Keio Journal of Medicine, 50*(1), 26-30.

Smit, A., Hubley, R. & Green, P. (2014). *RepeatMasker open-3.0.*. Retrieved 03,25, 2014, from http://www.repeatmasker.org

Smith, T. F., Waterman, M. S., & Fitch, W. M. (1981). Comparative biosequence metrics. *Journal of Molecular Evolution, 18*(1), 38-46.

Stankiewicz, P., & Lupski, J. R. (2002). Genome architecture, rearrangements and genomic disorders. *Trends in Genetics : TIG, 18*(2), 74-82.

Tomlins, S. A., Rhodes, D. R., Perner, S., Dhanasekaran, S. M., Mehra, R., Sun, X. W., et al. (2005). Recurrent fusion of TMPRSS2 and ETS transcription factor genes in prostate cancer. *Science (New York, N.Y.), 310*(5748), 644-648.

Tuzun, E., Sharp, A. J., Bailey, J. A., Kaul, R., Morrison, V. A., Pertz, L. M., et al. (2005). Fine-scale structural variation of the human genome. *Nature Genetics, 37*(7), 727-732.

Ulahannan, D., Kovac, M. B., Mulholland, P. J., Cazier, J. B., & Tomlinson, I. (2013). Technical and implementation issues in using next-generation sequencing of cancers in clinical practice. *British Journal of Cancer, 109*(4), 827-835.

Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., et al. (2001). The sequence of the human genome. *Science (New York, N.Y.), 291*(5507), 1304-1351.

Vissers, L. E., de Ligt, J., Gilissen, C., Janssen, I., Steehouwer, M., de Vries, P., et al. (2010). A de novo paradigm for mental retardation. *Nature Genetics, 42*(12), 1109-1112.

Volpi, E. V., & Bridger, J. M. (2008). FISH glossary: An overview of the fluorescence in situ hybridization technique. *BioTechniques, 45*(4), 385-6, 388, 390 passim.

Wang, J., Mullighan, C. G., Easton, J., Roberts, S., Heatley, S. L., Ma, J., et al. (2011). CREST maps somatic structural variation in cancer genomes with base-pair resolution. *Nature Methods, 8*(8), 652-654.

Wijchers, P. (2011). Genome organization influences partner selection for chromosomal rearrangements. *Trends in Genetics, 27*(2), 63; 63-71; 71.

Yaffe, E., & Tanay, A. (2011). Probabilistic modeling of hi-C contact maps eliminates systematic biases to characterize global chromosomal architecture. *Nature Genetics, 43*(11), 1059-1065.

Zeitouni, B., Boeva, V., Janoueix-Lerosey, I., Loeillet, S., Legoix-ne, P., Nicolas, A., et al. (2010). SVDetect: A tool to identify genomic structural variations from paired-end and mate-pair sequencing data. *Bioinformatics (Oxford, England), 26*(15), 1895-1896.
108

APPENDIX A
[LIST OF 1000-GENOMES FILES USED IN ANALYSIS]

NA19238	NA19238
FASTQ_FILE	MD5
data/NA19238/sequence_read/SRR005192_1.filt.fastq	3110d6d61bdcc620c9a179b93c0f
data/NA19238/sequence_read/SRR005192_2.filt.fastq	e7bda40a41048d14827fb0cc4f38
data/NA19238/sequence_read/SRR005193_1.filt.fastq	c8798ad272b04ec4048c78ce85b1
data/NA19238/sequence_read/SRR005193_2.filt.fastq	7bf0c21b54862ed57a1657a43c6d
data/NA19238/sequence_read/SRR005194_1.filt.fastq	ec8fe3d8289f22c8baacbd43e16c5
data/NA19238/sequence_read/SRR005194_2.filt.fastq	077f9e9e8921f966c3b659fc02c8e
data/NA19238/sequence_read/SRR005195_1.filt.fastq	8966bbc2201e0632e609193bf66
data/NA19238/sequence_read/SRR005195_2.filt.fastq	5c37e56b89e025904ddc5d7cebcb
data/NA19238/sequence_read/SRR005196_1.filt.fastq	e8b748413e193c88b89307200ef
data/NA19238/sequence_read/SRR005196_2.filt.fastq	c8222dcc6a8a598be80d5847aa84
data/NA19238/sequence_read/SRR005197_1.filt.fastq	a973556332823824b53fd22e438
data/NA19238/sequence_read/SRR005197_2.filt.fastq	8f1c57c94b08cb6ce7f2c2bdd553b
data/NA19238/sequence_read/SRR005198_1.filt.fastq	8a9d339a0eaa86a693fc6112e4a9
data/NA19238/sequence_read/SRR005198_2.filt.fastq	571be24092fd1461fc603c24fca88
data/NA19238/sequence_read/SRR005207_1.filt.fastq	b58c47719b1fd3978bb66c13cf21
data/NA19238/sequence_read/SRR005207_2.filt.fastq	ac1b7529997dff1a66caa91f7c7a0
data/NA19238/sequence_read/SRR005208_1.filt.fastq	c9be0f679ddb17d80c939b4310b7
data/NA19238/sequence_read/SRR005208_2.filt.fastq	e4ec92a85bbc5bb67b224b9af827
data/NA19238/sequence_read/SRR005209_1.filt.fastq	863bae9c44ce63c6e6bc05157a6e
data/NA19238/sequence_read/SRR005209_2.filt.fastq	78a0b36d85660647acb9de113ca
data/NA19238/sequence_read/SRR005210_1.filt.fastq	bc0be085804c2516b49c7a36a899
data/NA19238/sequence_read/SRR005210_2.filt.fastq	8476cad371e344af09e05b5cea38
data/NA19238/sequence_read/SRR005211_1.filt.fastq	54adf7a8bae6c62a5461c400f99e
data/NA19238/sequence_read/SRR005211_2.filt.fastq	2522236016de36b57a53bd109ab
data/NA19238/sequence_read/SRR005212_1.filt.fastq	6ba1954e295bda218a2118fc8aa2
data/NA19238/sequence_read/SRR005212_2.filt.fastq	b445a0cfc600db40e0b691418333
data/NA19238/sequence_read/SRR005213_1.filt.fastq	fd4ae153f97419fcde27757202adf
data/NA19238/sequence_read/SRR005213_2.filt.fastq	9445853a49d0bf79210427ed0f3f
data/NA19238/sequence_read/SRR005214_1.filt.fastq	e30e3fe5bef08ff7426972e513d22
data/NA19238/sequence_read/SRR005214_2.filt.fastq	0539bd41f8e05fc48a5725130e5b
data/NA19238/sequence_read/SRR005234_1.filt.fastq	ac5ab344453144ac814e4a641d1
data/NA19238/sequence_read/SRR005234_2.filt.fastq	537dcfd662bb446057a33aec0d15
NA19239	**NA19239**
FASTQ_FILE	MD5
data/NA19239/sequence_read/SRR002955_1.filt.fastq	d2ba15c23b0ea4d141091e29498
data/NA19239/sequence_read/SRR002955_2.filt.fastq	89d1488783426bc5bee1f6486d2
data/NA19239/sequence_read/SRR002956_1.filt.fastq	f6a17acc125140972b1d0e1658d7
data/NA19239/sequence_read/SRR002956_2.filt.fastq	84c5ee8d297a472d2bd98bedd9c
data/NA19239/sequence_read/SRR002957_1.filt.fastq	494845e8f8c23228eff7a51371fe7
data/NA19239/sequence_read/SRR002957_2.filt.fastq	942fc8cbdc1ca14850d436381d6c
data/NA19239/sequence_read/SRR002958_1.filt.fastq	f2ba138f5b4e3d2c22e24ef007e5
data/NA19239/sequence_read/SRR002958_2.filt.fastq	8da0f35e82f3d892ab0af3278192
data/NA19239/sequence_read/SRR002959_1.filt.fastq	ae73beeba370de5bf7e0a3979795

data/NA19239/sequence_read/SRR002959_2.filt.fastq	e074b4e1ac1385f64f8d567f6c648
data/NA19239/sequence_read/SRR002960_1.filt.fastq	8b25fee1184f5637cf16ccbb0d729
data/NA19239/sequence_read/SRR002960_2.filt.fastq	b279a5a991b49db48a5639e5fb9f
data/NA19239/sequence_read/SRR002961_1.filt.fastq	09578c8ecb4d08fe105d9c01e4ce
data/NA19239/sequence_read/SRR002961_2.filt.fastq	af805552162f41cdd043f72c06add
data/NA19239/sequence_read/SRR002962_1.filt.fastq	236fa73d55de2cd976342993bf21
data/NA19239/sequence_read/SRR002962_2.filt.fastq	98ea38969923270d3336af8a2d0
data/NA19239/sequence_read/SRR002963_1.filt.fastq	02ff2466baf9bd74f619305194058
data/NA19239/sequence_read/SRR002963_2.filt.fastq	aa7ee92f6393097d52016b8b151c
data/NA19239/sequence_read/SRR002964_1.filt.fastq	b25bccafd54edd86abf4e389eda1
data/NA19239/sequence_read/SRR002964_2.filt.fastq	872db6265531d791f22027046c6
data/NA19239/sequence_read/SRR002965_1.filt.fastq	88a0503c066051cc88b962882c0c
data/NA19239/sequence_read/SRR002965_2.filt.fastq	bce9ac30781e5ec8fd19cbc70e50
data/NA19239/sequence_read/SRR002966_1.filt.fastq	ae76268b7ad710dfd46aa2df8579
data/NA19239/sequence_read/SRR002966_2.filt.fastq	46f7b4b1b7296fe89f39700d5132
data/NA19239/sequence_read/SRR002967_1.filt.fastq	d6c6ad9f92cb457db2087e6ce363
data/NA19239/sequence_read/SRR002967_2.filt.fastq	503e515dde2f13a78f0e629eba99
data/NA19239/sequence_read/SRR003029_1.filt.fastq	d4c1a8c8c5acb90146499d58d603
data/NA19239/sequence_read/SRR003029_2.filt.fastq	45111b5330c104a73017defcf87c
data/NA19239/sequence_read/SRR007422_1.filt.fastq	250b5c091fe1c11cf1710f445e2db
data/NA19239/sequence_read/SRR007422_2.filt.fastq	8ec2e99eddab5810d66d1ae3c99

NA19240	**NA19240**
FASTQ_FILE	MD5
data/NA19240/sequence_read/SRR004483_1.filt.fastq	b7ce1dcb4a62c3382f86143ced4b
data/NA19240/sequence_read/SRR004483_2.filt.fastq	2d0d9b8c156939e516282f3ea9c0
data/NA19240/sequence_read/SRR004484_1.filt.fastq	0b4cf789a716c1235c31fc574b14
data/NA19240/sequence_read/SRR004484_2.filt.fastq	071c8e38659b21bcfc2e726abfe9
data/NA19240/sequence_read/SRR004485_1.filt.fastq	1b615fb8d04788c413ce5424f7b6
data/NA19240/sequence_read/SRR004485_2.filt.fastq	c666b912d12e3e738a550ac12acb
data/NA19240/sequence_read/SRR004783_1.filt.fastq	19f0db03a8f822b7a3df27f21e1b8
data/NA19240/sequence_read/SRR004783_2.filt.fastq	350412ab18793c4240ef2296d61a
data/NA19240/sequence_read/SRR004784_1.filt.fastq	be46222998b027213cfafa5d9a8d
data/NA19240/sequence_read/SRR004784_2.filt.fastq	77dabc7efefea2c062af0c8fda476
data/NA19240/sequence_read/SRR004785_1.filt.fastq	7aef2b68952f07ceb5a4e3834103
data/NA19240/sequence_read/SRR004785_2.filt.fastq	6df41a7a5290e164adc95642358d
data/NA19240/sequence_read/SRR004786_1.filt.fastq	828c0a376a745e64009b0e2ecc79
data/NA19240/sequence_read/SRR004786_2.filt.fastq	0c3f85ad38d3ad4e8df5d21924e9
data/NA19240/sequence_read/SRR004788_1.filt.fastq	c59b50bd44ad5d3f7f7782e1316f
data/NA19240/sequence_read/SRR004788_2.filt.fastq	a37b8ff65efe7fce535fd6ad7fa73a

NA19238	NA19238	NA19238	NA19238	NA19238
RUN_ID	STUDY_ID	STUDY_NAME	CENTER_NAME	SUBMISSION_ID
SRR005192	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002880
SRR005192	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002880
SRR005193	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002881
SRR005193	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002881
SRR005194	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002882
SRR005194	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002882
SRR005195	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002883
SRR005195	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002883
SRR005196	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002884
SRR005196	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002884
SRR005197	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002885
SRR005197	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002885
SRR005198	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002886
SRR005198	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002886
SRR005207	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002895
SRR005207	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002895
SRR005208	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002896
SRR005208	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002896
SRR005209	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002897
SRR005209	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002897
SRR005210	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002898
SRR005210	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002898
SRR005211	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002899
SRR005211	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002899
SRR005212	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002900
SRR005212	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002900
SRR005213	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002901
SRR005213	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002901
SRR005214	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002902
SRR005214	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002902
SRR005234	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002922
SRR005234	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002922
NA19239	**NA19239**	**NA19239**	**NA19239**	**NA19239**
RUN_ID	STUDY_ID	STUDY_NAME	CENTER_NAME	SUBMISSION_ID
SRR002955	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001427
SRR002955	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001427
SRR002956	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001428
SRR002956	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001428
SRR002957	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001429
SRR002957	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001429
SRR002958	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001430
SRR002958	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001430

SRR002959	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001431
SRR002959	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001431
SRR002960	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001432
SRR002960	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001432
SRR002961	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001433
SRR002961	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001433
SRR002962	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001434
SRR002962	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001434
SRR002963	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001435
SRR002963	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001435
SRR002964	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001436
SRR002964	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001436
SRR002965	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001437
SRR002965	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001437
SRR002966	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001438
SRR002966	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001438
SRR002967	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001439
SRR002967	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001439
SRR003029	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001500
SRR003029	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA001500
SRR007422	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA004284
SRR007422	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA004284
NA19240	**NA19240**	**NA19240**	**NA19240**	**NA19240**
RUN_ID	STUDY_ID	STUDY_NAME	CENTER_NAME	SUBMISSION_ID
SRR004483	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002351
SRR004483	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002351
SRR004484	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002352
SRR004484	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002352
SRR004485	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002353
SRR004485	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002353
SRR004783	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002629
SRR004783	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002629
SRR004784	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002630
SRR004784	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002630
SRR004785	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002631
SRR004785	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002631
SRR004786	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002632
SRR004786	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002632
SRR004788	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002636
SRR004788	SRP000032	1000Genomes Project Pilot 2	WUGSC	SRA002636

NA19238	NA19238	NA19238	NA19238	NA19238
SUBMISSION_DATE	SAMPLE_ID	SAMPLE_NAME	POPULATION	EXPERIMENT_ID
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/24/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/26/2008 0:00	SRS000212	NA19238	YRI	SRX001106
9/26/2008 0:00	SRS000212	NA19238	YRI	SRX001106
NA19239	**NA19239**	**NA19239**	**NA19239**	**NA19239**
SUBMISSION_DATE	SAMPLE_ID	SAMPLE_NAME	POPULATION	EXPERIMENT_ID
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654

8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/15/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/17/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/17/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/21/2008 0:00	SRS000213	NA19239	YRI	SRX000654
8/21/2008 0:00	SRS000213	NA19239	YRI	SRX000654
NA19240	**NA19240**	**NA19240**	**NA19240**	**NA19240**
SUBMISSION_DATE	SAMPLE_ID	SAMPLE_NAME	POPULATION	EXPERIMENT_ID
9/23/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/23/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/23/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/23/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/23/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/23/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/24/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/24/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/24/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/24/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/24/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/24/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/24/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/24/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/24/2008 0:00	SRS000214	NA19240	YRI	SRX001102
9/24/2008 0:00	SRS000214	NA19240	YRI	SRX001102

NA19238	NA19238	NA19238	NA19238	NA19238
INSTRUMENT_PLATFORM	INSTRUMENT_MODEL	LIBRARY_NAME	RUN_NAME	INSERT_SIZE
ILLUMINA	Illumina Genome Analyzer II	2675169269	7592	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7592	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7593	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7593	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7594	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7594	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7595	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7595	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7596	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7596	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7597	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7597	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7598	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7598	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7607	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7607	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7608	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7608	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7609	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7609	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7610	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7610	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7611	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7611	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7612	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7612	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7613	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7613	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7614	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7614	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7646	260
ILLUMINA	Illumina Genome Analyzer II	2675169269	7646	260
NA19239	**NA19239**	**NA19239**	**NA19239**	**NA19239**
INSTRUMENT_P	INSTRUMENT_MODEL	LIBRARY_NAME	RUN_NAME	INSERT_SIZE
ILLUMINA	Illumina Genome Analyzer	2485443314	5685	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5685	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5686	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5686	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5687	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5687	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5688	260

ILLUMINA	Illumina Genome Analyzer	2485443314	5688	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5689	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5689	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5690	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5690	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5691	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5691	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5692	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5692	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5693	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5693	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5694	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5694	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5695	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5695	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5696	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5696	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5697	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5697	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5895	260
ILLUMINA	Illumina Genome Analyzer	2485443314	5895	260
ILLUMINA	Illumina Genome Analyzer	2485443314	6430	260
ILLUMINA	Illumina Genome Analyzer	2485443314	6430	260

NA19240	**NA19240**	**NA19240**	**NA19240**	**NA19240**
INSTRUMENT_L	INSTRUMENT_MODEL	LIBRARY_NAME	RUN_NAME	INSERT_SIZE
ILLUMINA	Illumina Genome Analyzer II	2675080346	7223	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7223	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7224	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7224	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7225	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7225	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7522	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7522	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7523	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7523	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7524	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7524	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7525	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7525	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7526	260
ILLUMINA	Illumina Genome Analyzer II	2675080346	7526	260

NA19238	NA19238	NA19238
LIBRARY	PAIRED_FASTQ	READ
_LAYOUT		_COUNT
PAIRED	data/NA19238/sequence_read/SRR005192_2.filt.fastq.gz	11270293
PAIRED	data/NA19238/sequence_read/SRR005192_1.filt.fastq.gz	11270293
PAIRED	data/NA19238/sequence_read/SRR005193_2.filt.fastq.gz	11173375
PAIRED	data/NA19238/sequence_read/SRR005193_1.filt.fastq.gz	11173375
PAIRED	data/NA19238/sequence_read/SRR005194_2.filt.fastq.gz	10300764
PAIRED	data/NA19238/sequence_read/SRR005194_1.filt.fastq.gz	10300764
PAIRED	data/NA19238/sequence_read/SRR005195_2.filt.fastq.gz	8931016
PAIRED	data/NA19238/sequence_read/SRR005195_1.filt.fastq.gz	8931016
PAIRED	data/NA19238/sequence_read/SRR005196_2.filt.fastq.gz	8075975
PAIRED	data/NA19238/sequence_read/SRR005196_1.filt.fastq.gz	8075975
PAIRED	data/NA19238/sequence_read/SRR005197_2.filt.fastq.gz	7380429
PAIRED	data/NA19238/sequence_read/SRR005197_1.filt.fastq.gz	7380429
PAIRED	data/NA19238/sequence_read/SRR005198_2.filt.fastq.gz	8413304
PAIRED	data/NA19238/sequence_read/SRR005198_1.filt.fastq.gz	8413304
PAIRED	data/NA19238/sequence_read/SRR005207_2.filt.fastq.gz	5792641
PAIRED	data/NA19238/sequence_read/SRR005207_1.filt.fastq.gz	5792641
PAIRED	data/NA19238/sequence_read/SRR005208_2.filt.fastq.gz	5388572
PAIRED	data/NA19238/sequence_read/SRR005208_1.filt.fastq.gz	5388572
PAIRED	data/NA19238/sequence_read/SRR005209_2.filt.fastq.gz	4034189
PAIRED	data/NA19238/sequence_read/SRR005209_1.filt.fastq.gz	4034189
PAIRED	data/NA19238/sequence_read/SRR005210_2.filt.fastq.gz	4510010
PAIRED	data/NA19238/sequence_read/SRR005210_1.filt.fastq.gz	4510010
PAIRED	data/NA19238/sequence_read/SRR005211_2.filt.fastq.gz	4060395
PAIRED	data/NA19238/sequence_read/SRR005211_1.filt.fastq.gz	4060395
PAIRED	data/NA19238/sequence_read/SRR005212_2.filt.fastq.gz	4397559
PAIRED	data/NA19238/sequence_read/SRR005212_1.filt.fastq.gz	4397559
PAIRED	data/NA19238/sequence_read/SRR005213_2.filt.fastq.gz	4996197
PAIRED	data/NA19238/sequence_read/SRR005213_1.filt.fastq.gz	4996197
PAIRED	data/NA19238/sequence_read/SRR005214_2.filt.fastq.gz	6422603
PAIRED	data/NA19238/sequence_read/SRR005214_1.filt.fastq.gz	6422603
PAIRED	data/NA19238/sequence_read/SRR005234_2.filt.fastq.gz	8418918
PAIRED	data/NA19238/sequence_read/SRR005234_1.filt.fastq.gz	8418918
NA19239	**NA19239**	**NA19239**
LIBRARY_	PAIRED_FASTQ	READ_COU
PAIRED	data/NA19239/sequence_read/SRR002955_2.filt.fastq.gz	7258052
PAIRED	data/NA19239/sequence_read/SRR002955_1.filt.fastq.gz	7258052
PAIRED	data/NA19239/sequence_read/SRR002956_2.filt.fastq.gz	7170493
PAIRED	data/NA19239/sequence_read/SRR002956_1.filt.fastq.gz	7170493
PAIRED	data/NA19239/sequence_read/SRR002957_2.filt.fastq.gz	7245012
PAIRED	data/NA19239/sequence_read/SRR002957_1.filt.fastq.gz	7245012
PAIRED	data/NA19239/sequence_read/SRR002958_2.filt.fastq.gz	7414026
PAIRED	data/NA19239/sequence_read/SRR002958_1.filt.fastq.gz	7414026

PAIRED	data/NA19239/sequence_read/SRR002959_2.filt.fastq.gz	8114993
PAIRED	data/NA19239/sequence_read/SRR002959_1.filt.fastq.gz	8114993
PAIRED	data/NA19239/sequence_read/SRR002960_2.filt.fastq.gz	8079331
PAIRED	data/NA19239/sequence_read/SRR002960_1.filt.fastq.gz	8079331
PAIRED	data/NA19239/sequence_read/SRR002961_2.filt.fastq.gz	7959676
PAIRED	data/NA19239/sequence_read/SRR002961_1.filt.fastq.gz	7959676
PAIRED	data/NA19239/sequence_read/SRR002962_2.filt.fastq.gz	5187607
PAIRED	data/NA19239/sequence_read/SRR002962_1.filt.fastq.gz	5187607
PAIRED	data/NA19239/sequence_read/SRR002963_2.filt.fastq.gz	6173753
PAIRED	data/NA19239/sequence_read/SRR002963_1.filt.fastq.gz	6173753
PAIRED	data/NA19239/sequence_read/SRR002964_2.filt.fastq.gz	6055820
PAIRED	data/NA19239/sequence_read/SRR002964_1.filt.fastq.gz	6055820
PAIRED	data/NA19239/sequence_read/SRR002965_2.filt.fastq.gz	6745735
PAIRED	data/NA19239/sequence_read/SRR002965_1.filt.fastq.gz	6745735
PAIRED	data/NA19239/sequence_read/SRR002966_2.filt.fastq.gz	6621194
PAIRED	data/NA19239/sequence_read/SRR002966_1.filt.fastq.gz	6621194
PAIRED	data/NA19239/sequence_read/SRR002967_2.filt.fastq.gz	6706504
PAIRED	data/NA19239/sequence_read/SRR002967_1.filt.fastq.gz	6706504
PAIRED	data/NA19239/sequence_read/SRR003029_2.filt.fastq.gz	5826974
PAIRED	data/NA19239/sequence_read/SRR003029_1.filt.fastq.gz	5826974
PAIRED	data/NA19239/sequence_read/SRR007422_2.filt.fastq.gz	6497852
PAIRED	data/NA19239/sequence_read/SRR007422_1.filt.fastq.gz	6497852
NA19240	**NA19240**	**NA19240**
LIBRARY_	PAIRED_FASTQ	READ_COU
PAIRED	data/NA19240/sequence_read/SRR004483_2.filt.fastq.gz	10553597
PAIRED	data/NA19240/sequence_read/SRR004483_1.filt.fastq.gz	10553597
PAIRED	data/NA19240/sequence_read/SRR004484_2.filt.fastq.gz	11564301
PAIRED	data/NA19240/sequence_read/SRR004484_1.filt.fastq.gz	11564301
PAIRED	data/NA19240/sequence_read/SRR004485_2.filt.fastq.gz	10252162
PAIRED	data/NA19240/sequence_read/SRR004485_1.filt.fastq.gz	10252162
PAIRED	data/NA19240/sequence_read/SRR004783_2.filt.fastq.gz	9921824
PAIRED	data/NA19240/sequence_read/SRR004783_1.filt.fastq.gz	9921824
PAIRED	data/NA19240/sequence_read/SRR004784_2.filt.fastq.gz	11020784
PAIRED	data/NA19240/sequence_read/SRR004784_1.filt.fastq.gz	11020784
PAIRED	data/NA19240/sequence_read/SRR004785_2.filt.fastq.gz	11378879
PAIRED	data/NA19240/sequence_read/SRR004785_1.filt.fastq.gz	11378879
PAIRED	data/NA19240/sequence_read/SRR004786_2.filt.fastq.gz	11581025
PAIRED	data/NA19240/sequence_read/SRR004786_1.filt.fastq.gz	11581025
PAIRED	data/NA19240/sequence_read/SRR004788_2.filt.fastq.gz	11172997
PAIRED	data/NA19240/sequence_read/SRR004788_1.filt.fastq.gz	11172997

NA19238	NA19238
BASE_COUNT	ANALYSIS_GROUP
405730548	high coverage
405730548	high coverage
402241500	high coverage
402241500	high coverage
370827504	high coverage
370827504	high coverage
321516576	high coverage
321516576	high coverage
290735100	high coverage
290735100	high coverage
265695444	high coverage
265695444	high coverage
302878944	high coverage
302878944	high coverage
208535076	high coverage
208535076	high coverage
193988592	high coverage
193988592	high coverage
145230804	high coverage
145230804	high coverage
162360360	high coverage
162360360	high coverage
146174220	high coverage
146174220	high coverage
158312124	high coverage
158312124	high coverage
179863092	high coverage
179863092	high coverage
231213708	high coverage
231213708	high coverage
303081048	high coverage
303081048	high coverage

NA19239	NA19239
BASE_COUNT	ANALYSIS_GROUP
261289872	high coverage
261289872	high coverage
258137748	high coverage
258137748	high coverage
260820432	high coverage
260820432	high coverage
266904936	high coverage

266904936	high coverage
292139748	high coverage
292139748	high coverage
290855916	high coverage
290855916	high coverage
286548336	high coverage
286548336	high coverage
186753852	high coverage
186753852	high coverage
222255108	high coverage
222255108	high coverage
218009520	high coverage
218009520	high coverage
242846460	high coverage
242846460	high coverage
238362984	high coverage
238362984	high coverage
241434144	high coverage
241434144	high coverage
209771064	high coverage
209771064	high coverage
233922672	high coverage
233922672	high coverage

NA19240	**NA19240**
BASE_COUNT	ANALYSIS_GROUP
369375895	high coverage
369375895	high coverage
404750535	high coverage
404750535	high coverage
358825670	high coverage
358825670	high coverage
347263840	high coverage
347263840	high coverage
385727440	high coverage
385727440	high coverage
398260765	high coverage
398260765	high coverage
405335875	high coverage
405335875	high coverage
391054895	high coverage
391054895	high coverage

APPENDIX B

[BLAT SUITE PROGRAM SPECIFICATIONS AND USER'S GUIDE]

BLAT Suite Program Specifications and User Guide

General:

Blat produces two major classes of alignments: at the DNA level between two sequences that are of 95% or greater identity, but which may include large inserts, and at the protein or translated DNA level between sequences that are of 80% or greater identity and may also include large inserts. The output of BLAT is flexible. By default it is a simple tab-delimited file which describes the alignment, but which does not include the sequence of the alignment itself. Optionally it can produce BLAST and WU-BLAST compatable output as well as a number of other formats.

The main programs in the blat suite are:

- gfServer – a server that maintains an index of the genome in memory and uses the index to quickly find regions with high levels of sequence similarity to a query sequence.
- gfClient – a program that queries gfServer over the network, and then does a detailed alignment of the query sequence with regions found by gfServer.
- blat –combines client and server into a single program, first building the index, then using the index, and then exiting.
- webBlat – a web based version of gfClient that presents the alignments in an interactive fashion.

Building an index of the genome typically takes 10 or 15 minutes. Typically for interactive applications one uses gfServer to build a whole genome index. At that point gfClient or webBlat can align a single query within few seconds. If one is aligning a lot of sequences in a batch
mode then blat can be more efficient, particularly if run on a cluster of computers. Each blat run is typically done against a single chromosome, but with a large number of query sequences.

Other programs in the blat suite are:

- pslSort – combines and sorts the output of multiple blat runs. (The blat default output format is .psl).
- pslReps – selects the best alignments for a particular query sequence, using a ‘near best in genome’ approach.

- pslPretty – converts alignments from the psl format, which is tab-delimited format and does not include the bases themselves, to a more readable alignment format.
- faToTwoBit – convert Fasta format sequence files to a dense randomly-accessable .2bit format that gfClient can use.
  - twoBitToFa – convert from the .2bit format back to fasta
- faToNib – convert from Fasta to a somewhat less dense randomly accessible format that predates .2bit. Note each .nib file can only contain a single sequence.

- nibFrag – convert portions of a nib file back to fasta.

In addition you may be interested in the following programs which are not part of the BLAT
suite:

- In Silico PCR – given two primers quickly find the sequence between them. Available from Kent Informatics. This includes webPCR, an interface similar to webBlat.
- The Genome Browser – display annotations as a series of tracks on top of the genome.
  Available from the University of California
  Santa Cruz. See
  http://genome.ucsc.edu/license/.

Running the Programs:

The command line options of each of the programs is described below. Similar summaries of usage are printed when a command is run with no arguments. See the next section for info on installing webBlat.

blat

```
blat - Standalone BLAT sequence search

command line tool usage:
```

```
   blat database query [-ooc=11.ooc]
output.psl where:
   database and query are each either a .fa , .nib or .2bit file,
   or a list these files one file name per line.
   -ooc=11.ooc tells the program to load over-occurring
               11-mers from and external file.      This
               will increase the speed
               by a factor of 40 in many cases, but is
   not required output.psl is where to put the output.
   Subranges of nib and .2bit files may specified using the syntax:
      /path/file.nib:seqid:start-end
   or
      /path/file.2bit:seqid:start-end
   or
      /path/file.nib:start-end
   With the second form, a sequence id of file:start-end will
be used. options:
   -t=type     Database type.  Type is one of:
                 dna - DNA sequence
                 prot - protein sequence
                 dnax - DNA sequence translated in six frames to protein
               The default is dna
   -q=type     Query type.  Type is one of:
                 dna - DNA
                 sequence
                 rna - RNA
                 sequence
```

```
                prot - protein sequence

                dnax - DNA sequence translated in six frames to

                protein rnax - DNA sequence translated in three

                frames to protein

             The default is dna

   -prot     Synonymous with -t=prot -q=prot

   -ooc=N.ooc  Use overused tile file N.ooc.  N should

             correspond to the tileSize

   -tileSize=N sets the size of match that triggers an alignment.

             Usually between 8 and 12

             Default is 11 for DNA and 5 for protein.

   -stepSize=N spacing between tiles. Default is tileSize.

   -oneOff=N   If set to 1 this allows one mismatch in tile and still

          triggers an alignments.  Default
          is 0.

-minMatch=N sets the number of tile matches.  Usually set from 2 to 4

          Default  is  2  for  nucleotide,  1  for
          protein.

-minScore=N sets minimum score.  This is the matches minus the

          mismatches minus some sort of gap penalty.  Default

          is 30

-minIdentity=N Sets minimum sequence identity (in percent).  Default is

          90 for nucleotide searches, 25 for protein or

          translated protein searches.

-maxGap=N   sets the size of maximum gap between tiles in a clump.

                          Usually set from 0 to 3.

                          Default is 2. Only relevent for

          minMatch > 1.
```

-noHead suppress .psl header (so it's just a tab-separated file)

-makeOoc=N.ooc Make overused tile file. Target needs to be complete genome.

-repMatch=N sets the number of repetitions of a tile allowed before

it is marked as overused. Typically this is 256 for tileSize

12, 1024 for tile size 11, 4096 for tile size 10.

Default is 1024. Typically only comes into play with

makeOoc. Also affected by stepSize. When stepSize is

halved repMatch is doubled to compensate.

-mask=type Mask out repeats. Alignments won't be started in masked region

but may extend through it in nucleotide searches. Masked

areas are ignored entirely in protein or translated

searches. Types are

lower - mask out lower cased sequence

upper - mask out upper cased sequence

out - mask according to database.out RepeatMasker

.out file file.out - mask database according to

RepeatMasker file.out

-qMask=type Mask out repeats in query sequence. Similar to -mask

above but for query rather than target sequence.

-repeats=type Type is same as mask types above. Repeat bases will

not be masked in any way, but matches in repeat areas

will be reported separately from matches in other areas

in the psl output.

```
-minRepDivergence=NN - minimum percent divergence of repeats to
            allow them to be unmasked. Default is 15.      Only
            relevant for masking using RepeatMasker .out files.
-dots=N     Output dot every N sequences to show program's progress
-trimT      Trim leading poly-T
-noTrimA    Don't trim trailing poly-A
-trimHardA  Remove poly-A tail from qSize as well as
            alignments in psl output
-fastMap    Run for fast DNA/DNA remapping - not allowing
            introns, requiring high %ID
-out=type   Controls output file format.  Type is one of:
                psl - Default.  Tab separated format, no
                sequence pslx - Tab separated format with
                sequence
                axt - blastz-associated axt
                format maf - multiz-
                associated maf format sim4 -
                similar to sim4 format
                wublast - similar to wublast
                format
                blast - similar to NCBI blast
                format blast8- NCBI blast
                tabular format
                blast9 - NCBI blast tabular format with comments
-fine       For high quality mRNAs look harder for small
            initial and terminal exons.Not recommended for
            ESTs
```

```
-maxIntron=N  Sets maximum intron size. Default is 750000

-extendThroughN - Allows extension of alignment through large blocks of
N's
```

Here are some blat settings for common usage scenarios:

1) Mapping ESTs to the genome within the same species

```
    -ooc=11.ooc
```

2) Mapping full length mRNAs to the genome in the same species

```
    -ooc=11.ooc -fine -q=rna
```

3) Mapping ESTs to the genome across species

```
    -q=dnax -t=dnax
```

4) Mapping mRNA to the genome across species

```
    -q=rnax -t=dnax
```

5) Mapping proteins to the genome

```
    -q=prot -t=dnax
```

6) Mapping DNA to DNA in the same species

```
    -ooc=11.ooc -fastMap
```

7) Mapping DNA from one species to another species

```
    -q=dnax -t=dnax

    When mapping DNA from one species to another the

    query side of the alignment should be cut up

    into chunks of 25kb or less for best

    performance.
```

gfServer

```
gfServer - Make a server to quickly find where DNA occurs

in genome. To set up a server:

   gfServer start host port file(s)

   Where the files are in .nib or .2bit format

To remove a server:

   gfServer stop host port

To query a server with DNA sequence:

   gfServer query host port probe.fa

To query a server with protein sequence:

   gfServer protQuery host port probe.fa

To query a server with translated dna sequence:

   gfServer transQuery host port probe.fa

To process one probe fa file against a .nib format genome (not starting
server):

   gfServer direct probe.fa file(s).nib

To figure out usage

   level gfServer

   status host port

To get input file

   list gfServer

   files host port

Options:

   -tileSize=N size of n-mers to index.  Default is 11 for

               nucleotides, 4 for proteins (or translated

               nucleotides).

   -stepSize=N spacing between tiles. Default is tileSize.

   -minMatch=N Number of n-mer matches that trigger detailed alignment
```

```
            Default is 2 for nucleotides, 3 for
                        protiens.

-maxGap=N   Number of insertions or deletions allowed between n-mers.

            Default is 2 for nucleotides, 0 for
                        protiens.

-trans  Translate database to protein in 6 frames.  Note:

        it is best to run this on RepeatMasked data in this

        case.

-log=logFile keep a log file that records server requests.

-seqLog    Include sequences in log file (not logged with -syslog)

-syslog    Log to syslog

-logFacility=facility log to the specified syslog facility - default
local0.

-mask      Use masking from nib file.

-repMatch=N Number of occurrences of a tile (nmer)

            that trigger repeat masking the tile.

            Default is 1024.

-maxDnaHits=N Maximum number of hits for a dna query that are sent from
the

            server. Default is 100.

-maxTransHits=N Maximum number of hits for a translated query

            that are sent from the server. Default is 200.

-maxNtSize=N Maximum size of untranslated DNA query sequence

            Default is 40000

-maxAsSize=N Maximum size of protein or translated DNA queries

            Default is 8000

-canStop If set then a quit message will actually take

         down the server
```

gfClient

```
gfClient - A client for the genomic

finding program usage:

   gfClient host port nibDir in.fa out.psl

where

   host is the name of the machine running the

   gfServer port is the same as you started

   the gfServer with

   nibDir is the path of the nib files relative to the current dir

       (note these are needed by the client as well as the server)

   in.fa a fasta format file.  May contain

   multiple records out.psl where to put the

   output

options:

   -t=type     Database type.  Type is one of:

                 dna - DNA sequence

                 prot - protein sequence

                 dnax - DNA sequence translated in six frames to protein

                  The
               default
                is dna

   -q=type     Query type.  Type is one of:

                 dna - DNA

                 sequence

                 rna - RNA

                 sequence

                 prot - protein sequence
```

dnax - DNA sequence translated in six frames to protein rnax - DNA sequence translated in three frames to protein

-dots=N Output a dot every N query sequences

-nohead Suppresses psl five line header

-minScore=N sets minimum score. This is twice the matches minus the mismatches minus some sort of gap penalty. Default is 30

-minIdentity=N Sets minimum sequence identity (in percent). Default is 90 for nucleotide searches, 25 for protein or translated protein searches.

-out=type Controls output file format. Type is one of:

psl - Default. Tab separated format without actual sequence pslx - Tab separated format with sequence

axt - blastz-associated axt format maf - multiz-associated maf format

wublast - similar to wublast format blast - similar to NCBI blast format

-maxIntron=N Sets maximum intron size. Default is 750000

webBlat

webBlat generally is not run from the command line. See ‘Setting Up webBlat instructions below` for information on this program.

faToTwoBit

```
faToTwoBit - Convert DNA from fasta to

2bit format usage:

   faToTwoBit in.fa [in2.fa in3.fa ...]

out.2bit options:

   -noMask - Ignore lower-case masking in fa file.
```

twoBitToFa

```
twoBitToFa - Convert all or part of .2bit file

to fasta usage:

   twoBitToFa input.2bit

output.fa options:

   -seq=name - restrict this to just one sequence

   -start=X  - start at given position in sequence (zero-based)

   -end=X - end at given position in sequence (non-inclusive)
```

faToNib

```
faToNib - Convert from .fa

to .nib format usage:

   faToNib in.fa out.nib
```

nibFrag

```
nibFrag - Extract part of a

nib file as .fa usage:

   nibFrag file.nib start end strand out.fa
```

pslPretty

```
pslPretty - Convert PSL to human

readable output usage:

   pslPretty in.psl target.lst

query.lst pretty.out options:

   -axt - save in Scott Schwartz's axt format

   -dot=N Put out a dot every N records

   -long - Don't abbreviate long inserts
```

It's a really good idea if the psl file is sorted by target if it contains multiple targets. Otherwise this will be very very slow. The target and query lists can either be fasta files, nib files, or a list of fasta and/or nib files one per line. Currently this only handles nucleotide based psl files.

pslSort

```
pslSort - merge and sort psCluster

.psl output files usage:

   pslSort dirs[1|2] outFile
        tempDir inDir(s)

This will sort all of the .psl files in the

directories inDirs in two stages - first into

temporary files in tempDir and second into outFile.

                          The device on tempDir

needs to have

enough space (typically 15-20 gigabytes if processing whole genome)

  pslSort g2g[1|2] outFile tempDir inDir(s)
```

```
This will sort a genome to genome alignment,
reflecting the alignments across the diagonal.

Adding 1 or 2 after the dirs or g2g will limit the
program to only the first or second pass repectively
of the sort
Options:
   -verbose=N Set verbosity level, higher for more output. Default 1

Note for huge files pslSort will run out of memory.  The unix
  sort command sort –k 10 *.psl > sorted.psl
may be preferable in these situations, though the psl header
lines should be removed or avoided with the –noHead option to
blat.
```

pslReps

```
pslReps - analyse repeats and generate genome
wide best alignments from a sorted set of
local alignments
usage:
    pslReps in.psl out.psl out.psr
where in.psl is an alignment file generated by
psLayout and sorted by pslSort, out.psl is the
best alignment output
and out.psr contains repeat info
```

```
options:
    -nohead don't add PSL header
    -ignoreSize Will not weigh in favor of larger alignments so much
    -noIntrons Will not penalize for not having introns when
              calculating size factor
    -singleHit  Takes single best hit, not splitting into parts
    -minCover=0.N minimum coverage to output.  Default is 0.
    -ignoreNs Ignore 'N's when calculating minCover.
    -minAli=0.N minimum
               alignment ratio
               default is 0.93
    -nearTop=0.N how much can deviate from top and be taken
               default is 0.01
    -minNearTopSize=N  Minimum size of alignment that is
               near top for aligmnent to be kept.
                       Default 30.
    -coverQSizes=file Tab-separate file with effective query sizes.
                     When used with -minCover, this
                     allows polyAs to be excluded from
                     the coverage calculation
```

Setting Up webBlat

INSTALLING WEBBLAT
Installing A Web-Based Blat Server involves four major steps:

1) Creating sequence databases.

2) Running the gfServer program to create in-memory indexes of the databases.

3) Editing the webBlat.cfg file to tell it what machine and port the gfServer(s)

are running on, and optionally customizing the webBlat appearance to users.

4) Copying the webBlat executable and webBlat.cfg to a directory where the web server can execute webBlat as a CGI.

CREATING SEQUENCE DATABASES

You create databases with the program faToTwoBit. Typically you'll create a separate database for each genome you are indexing. Each database can contain up to four billion bases of sequence in an unlimited number of records. The databases for webPcr and webBlat are identical.

The input to faToTwoBit is one or more fasta format files each of which can contain multiple records. If the sequence contains repeat sequences, as is the case with vertebrates and many plants, the repeat sequences can be represented in lower case and the other sequence in upper case. The gfServer program can be configured to ignore the repeat sequences. The output of faToTwoBit is a file which is designed for fast random access and efficient storage. The output files store four bases per byte. They use a small amount of additional space to store the case of the DNA and to keep track of runs of

N's in the input. Non-N ambiguity codes such as Y and U in the input sequence will be converted to N.

Here's how a typical installation might create a mouse and a human genome database:

cd/data/genomes mkdir twoBit
faToTwoBit human/hg16/*.fa twoBit/hg16.2bit faToTwoBit mouse/mm4/*.fa twoBit/mm4.2bit
There's no need to put all of the databases in the same directory, but it can simplify bookkeeping.

The databases can also be in the .nib format which was used with blat and gfClient/gfServer until recently. The .nib format only packed 2 bases per byte, and could only handle one record per nib file. Recent versions of blat and related programs can use .2bit files as well.

CREATING IN-MEMORY INDICES WITH GFSERVER

The gfServer program creates an in-memory index of a nucleotide sequence database. The index can either be for translated or untranslated searches. Translated indexes enable protein-based blat queries and use approximately two bytes per unmasked base in the database. Untranslated indexes are used nucleotide-based blat queries as well
as for In-silico PCR. An index for normal blat uses approximately 1/4 byte per base. For blat on smaller (primer-sized) queries or for In-silico PCR a more thorough index that requires 1/2 byte per base is recommended. The gfServer is memory intensive but typically doesn not require a lot of CPU power. Memory permitting multiple gfServers can be run on the same machine.

A typical installation might go:

ssh bigRamMachine

cd /data/genomes/twoBit

gfServer start bigRamMachine 17779 hg16.2bit &

gfServer -trans -mask start bigRamMachine 17778 hg16.2bit &

the -trans flag makes a translated index. It will take approximately

15 minutes to build an untranslated index, and 45 minutes to build a translate index. To build an untranslated index to be shared with
In-silico PCR do

gfServer -stepSize=5 bigRamMachine 17779 hg16.2bit &

This index will be slightly more sensitive, noticeably so for small query sequences, with blat.

EDITING THE WEBBLAT.CFG FILE

The webBlat.cfg file tells the webBlat program where to look for gfServers and for sequence. The basic format of the .cfg file is line oriented with the
first word of the line being a command. Blank lines and lines starting with #

are ignored. The webBlat.cfg and webPcr.cfg files are similar. The webBlat.cfg commands are:
gfServer - defines host and port a (untranslated) gfServer is running on, the associated sequence directory, and the name of the database to display in the webPcr web page.

gfServerTrans - defines location of a translated server.

background - defines the background image if any to display on web page
company - defines company name to display on web page
tempDir - where to put temporary files. This path is relative to where the web server executes CGI scripts. It is good to remove files that haven't been accessed for 24 hours from this directory periodically, via a cron job or similar mechanism.
The background and company commands are optional. The webBlat.cfg file must have at least one valid gfServer or gfServerTrans line, and a tempDir line.
. Here is a webBlat.cfg file that you might find at a typical installation:

company Awesome Research Amalgamated background /images/dnaPaper.jpg
gfServer bigRamMachine 17778 /data/genomes/2bit/hg16.2bit Human Genome
gfServer bigRamMachine 17779 /data/genomes/2bit/hg16.2bit Human Genome
gfServer mouseServer 17780 /data/genomes/2bit/mm4.2bit Mouse Genome
gfServer mouseServer 17781 /data/genomes/2bit/mm4.2bit Mouse Genome
tempDir ../trash

PUTTING WEBBLAT WHERE THE WEB SERVER CAN EXECUTE IT

The details of this step vary highly from web server to web server. On a typical Apache installation it might be:
ssh webServer cd kent/webBlat
cp webBlat webBlat.cfg /usr/local/apache/cgi-bin mkdir /usr/local/apache/trash
chmod 777 /usr/local/apache/trash

assuming that you've put the executable and config file in kent/webBlat. The program will create some files in the trash directory. It is good to periodically clean out old files from this directory. On Mac OS-X instead you might do:
cp webBlat webBlat.cfg /Library/WebServer/CGI-Executables mkdir /Library/WebServer/trash
chmod 777 /Library/WebServer/trash

Unless you are administering your own computer you will likely need to ask your local system administrators for help with this part of the webBlat installation.

File Formats

.psl files

A .psl file describes a series of alignments in a dense easily parsed text format. It begins with a five line header which describes each field. Following this is one line for each alignment with a tab between each field. The fields are describe below in a format suitable for many relational databases.

matches int unsigned , # Number of bases that match that aren't repeats

misMatches int unsigned , # Number of bases that don't match

repMatches int unsigned , # Number of bases that match but are part of
repeats nCount int unsigned , # Number of 'N' bases
qNumInsert int unsigned , # Number of inserts in query
qBaseInsert int unsigned , # Number of bases inserted in query
tNumInsert int unsigned , # Number of inserts in target
tBaseInsert int unsigned , # Number of bases inserted in target
strand char(2) , # + or - for query strand, optionally followed by + or – for target strand
qName varchar(255) , # Query sequence name
qSize int unsigned , # Query sequence size

qStart int unsigned , # Alignment start position in query
qEnd int unsigned , # Alignment end position in query
tName varchar(255) , # Target sequence name
tSize int unsigned , # Target sequence size

tStart int unsigned , # Alignment start position in target
tEnd int unsigned , # Alignment end position in target
blockCount int unsigned , # Number of blocks in alignment. A block contains no gaps.
blockSizes longblob , # Size of each block in a comma separated list
qStarts longblob , # Start of each block in query in a comma separated list
tStarts longblob , # Start of each block in target in a comma separated list

In general the coordinates in psl files are “zero based half open.” The first base in a sequence is numbered zero rather than one. When representing a range the end coordinate is not included in the range. Thus the first 100 bases of a sequence are represented as 0-100, and the second 100 bases are represented as 100-200. There is a another little unusual feature in the .psl format. It has to do with how coordinates are handled on the negative strand. In the qStart/qEnd fields the coordinates are where it matches from the point of view of the forward strand (even when the match is on the reverse strand). However on the qStarts[] list, the coordinates are reversed.
Here's an example of a 30-mer that has 2 blocks that align on the minus strand and 2 blocks on the plus strand (this sort of stuff happens in real life in response to assembly errors sometimes).

```
0       1       2       3 tens position in query
012345678901234567890123456789 ones position in query
++++ +++++ plus strand alignment on query
--------        ----------      minus strand alignment on query
```

Plus strand:
qStart 12 qEnd 31 blockSizes 4,5 qStarts 12,26
Minus strand:
qStart 4 qEnd 26 blockSizes 10,8 qStarts 5,19

Essentially the minus strand blockSizes and qStarts are what you would get if you reverse complemented the query.However the qStart and qEnd are non-reversed. To get from one to the other:
qStart = qSize - revQEnd qEnd = qSize - revQStart

.2bit files

A .2bit file can store multiple DNA sequence (up to 4 gig total) in a compact randomly accessible format. The two bit files contain masking information as well as the DNA itself. The file begins with a 16 byte header containing the following fields:

1) signature – the number 0x1A412743 in the architecture of the machine that created the file.

2) version – zero for now. Readers should abort if they see a version number higher than 0.

3) sequenceCount – the number of sequences in the file

4) reserved – always zero for now.

All fields are 32 bits unless noted. If the signature value is not as given, the reader program should byte swap the signature and see if the swapped version matches. If so all multiple-byte entities in the file will need to be byte-swapped. This enables these binary files to be used unchanged on different architectures.

The header is followed by a file index. There is one entry in the index for each sequence. Each index entry contains three fields:

1) nameSize – a byte containing the length of the name field

2) name – this contains the sequence name itself, and is variable length depending on nameSize.

3) offset – 32 bit offset of the sequence data relative to the start of the file

The index is followed by the sequence records. These contain 9 fields:

1) dnaSize – number of bases of DNA in the sequence.

2) nBlockCount – the number of blocks of N's in the file (representing unknown
sequence).

3) nBlockStarts – a starting position for each block of N's

4) nBlockSizes – the size of each block of N's

5) maskBlockCount – the number of masked (lower case) blocks

6) maskBlockStarts – starting position for each masked block

7) maskBlockSizes – the size of each masked block

8) packedDna – the dna packed to two bits per base as so: 00 – T, 01 – C, 10 – A, 11 – G.
The first base is in the most significant 2 bits byte, and the last base in the least significant 2 bits, so that the sequence TCAG would be represented as 00011011. The packedDna field will be padded with 0 bits as necessary so that it takes an even multiple of 32 bit in the file, as this improves i/o performance on some machines.

.nib files

A .nib file describes a DNA sequence packing two bases into each byte. Each nib file contains only a single sequence. A nib file begins with a 32 bit signature which is 0x6BE93D3A in the architecture of the machine that created the file, and possibly a byte-swapped version of the same number on another machine. This is followed by a 32 bit number in the same format which describes the number of bases in the file. This is followed by the bases themselves packed two bases to the byte. The first base is packed in the high order 4 bits (nibble), the second base in the low order four bits. In C code:

byte = (base1<<4) + base2

The numerical values for the bases are:

0 – T, 1 – C, 2 – A, 3 – G, 4 – N (unknown)

The most significant bit in a nibble is set if the base is masked.

Limits

The gfServer program requires approximately 1 byte for every 3 bases in the genome it is indexing in DNA mode, and 1.5 bytes for each unmasked base in translated mode. The blat program requires approximately two bytes for each base in the genome in DNA mode, and three bytes for each base in translated mode. The other programs use relatively little memory.

APPENDIX C

[SIMULATED TRANSLOCATION REFERENCE LIST]

Additional file 1 - Comprehensive list of gene pairs involved in cancer-specific recurrent translocations which result in fusion transcripts						
Translocation	**Gene**	**Location**	**Fragile Site**	**Gene**	**Location**	**Fragile**
t(7;12)(p22;q13)	ACTB	7p22.1	FRA7B (common, apc)	GLI1	12q13.3	
inv(7)(q21q34)	AKAP9	7q21.2	FRA7E (common, apc)	BRAF	7q34	
t(X;17)(p11;q25)	ASPSCR1	17q25.3		TFE3	Xp11.23	
inv(2)(p23q35)	ATIC	2q35		ALK	2p23.2-p23.1	
t(17;20)(q23;q13)	BCAS4	20q13.13		BCAS3	17q23.2	
t(2;3)(p16;q26)	BCL11A	2p16.1		MDS1	3q26.2	
t(5;14)(q35;q32)	BCL11B	14q32.2		NKX2E	5q35.2	FRA5G (rare, folic acid)
t(5;14)(q35;q32)	BCL11B	14q32.2		TLX3	5q35.1	FRA5G (rare, folic acid)
inv(14)(q11q32)	BCL11B	14q32.2		TRD@	14q11.2	
t(14;18)(q32;q21)	BCL2	18q21.33	FRA18B (common, apc)	IGH@	14q32.33	
t(2;18)(p11;q21)	BCL2	18q21.33	FRA18B (common, apc)	IGK@	2p11.2	FRA2L (rare, folic acid)
t(18;22)(q21;q11)	BCL2	18q21.33	FRA18B (common, apc)	IGL@	22q11.22	
t(8;19)(q24;q13)	BCL3	19q13.31	FRA19A (common, 5-aza)	MYC	8q24.21	
t(3;16)(q27;p13)	BCL6	3q27.3	FRA3C (common, apc)	CIITA	16p13.13	
t(3;8)(q27;q24)	BCL6	3q27.3	FRA3C (common, apc)	MYC	8q24.21	
t(1;14)(q21;q32)	BCL9	1q21.1	FRA1F (common, apc)	IGH@	14q32.33	
t(1;22)(q21;q11)	BCL9	1q21.1	FRA1F (common, apc)	IGL@	22q11.22	
t(9;22)(q34;q11)	BCR	22q11.23		ABL1	9q34.12	
t(8;22)(p12;q11)	BCR	22q11.23		FGFR1	8p12	
t(9;22)(p24;q11)	BCR	22q11.23		JAK2	9p24.1	
t(4;22)(q12;q11)	BCR	22q11.23		PDGFRA	4q12	FRA4B (common, BrdU)
t(11;18)(q22;q21)	BIRC3	11q22.2		MALT1	18q21.32	FRA18B (common, apc)
t(X;11)(q21;q23)	BRWD3	Xq21.1		ARHGAP20	11q22.3-q23.1	
t(8;12)(q21;q22)	BTG1	12q21.33	FRA12B (common, apc)	MYC	8q24.21	
t(7;15)(p21;q21)	C15ORF21	15q21.1		ETV1	7p21.2	
t(3;3)(q21;q26)	C3ORF27	3q21.3		EVI1	3q26.2	
t(2;11)(p23;p15)	CARS	11p15.4		ALK	2p23.2-p23.1	
t(16;16)(p13;q22), inv(16)(p13q22)	CBFB	16q22.1	FRA16B (rare, dist A), FRA16C (common, apc)	MYH11	16p13.11	FRA16A (rare, folic acid)
t(5;10)(q33;q21)	CCDC6	10q21.2	FRA10C (common, BrdU)	PDGFRB	5q33.1	
inv(10)(q11q21)	CCDC6	10q21.2	FRA10C (common, BrdU)	RET	10q11.21	FRA10G (common, apc)
t(5;14)(q33;q32)	CCDC88C	14q32.12		PDGFRB	5q33.1	
t(11;19)(q13;p13)	CCND1	11q13 .2	FRA11H (common, apc)	FSTL3	19p13.3	FRA19B (rare, folic acid)
t(5;6)(q32-33;q22)	CD74	5q33.1		ROS1	6q22.2	
t(16;17)(q21;p13)	CDH11	16q21		USP6	17p13.2	
t(7;11)(q21;q23)	CDK6	7q21.2	FRA7E (common, apc)	MLL	11q23.3	FRA11B (rare, folic acid), FRA11G (common,
t(5;7)(q35;q21)	CDK6	7q21.2	FRA7E (common, apc)	TLX3	5q35.1	FRA5G (rare, folic acid)
t(5;11)(q12;q23)	CENPK	5q12.3		MLL	11q23.3	FRA11B (rare, folic acid), FRA11G (common,
t(8;9)(p12;q33)	CEP110	9q33.2		FGFR1	8p12	

t(4;12)(q12;p13)	CHIC2	4q12	FRA4B (common, BrdU)	ETV6	12p13.2	
t(4;19)(q35;q13)	CIC	19q13.2	FRA19A (common, 5-aza)	DUX4	4q35.2	
t(2;17)(p23;q23)	CLTC	17q23.1	FRA17B (common, apc)	ALK	2p23.2-p23.1	
t(X;17)(p11;q23)	CLTC	17q23.1	FRA17B (common, apc)	TFE3	Xp11.23	
t(2;22)(p23;q11)	CLTCL1	22q11.21		ALK	2p23.2-p23.1	
t(3;17)(q21;p13)	CNBP	3q21.3		USP6	17p13.2	
t(17;22)(q21;q13)	COL1A1	17q21.33		PDGFB	22q13.1	FRA22A (rare, folic acid)
t(17;17)(p13;q21)	COL1A1	17q21.33		USP6	17p13.2	
t(7;8)(q21;q12)	COL1A2	7q21.3		PLAG1	8q12.1	
t(X;6)(q22;q13-14)	COL4A5	Xq22.3		COL12A1	6q13-q14.1	FRA6D (common, BrdU)
t(1;2)(p13;q37)	COL6A3	2q37.3	FRA2J (common, apc)	CSF1	1p13.3	
t(8;12)(p12;q15)	CPSF6	12q15		FGFR1	8p12	
t(11;19)(q21;p13)	CRTC1	19p13.11	FRA19B (rare, folic acid)	MAML2	11q21	
t(11;15)(q21;q26)	CRTC3	15q26.1		MAML2	11q21	
t(3;8)(p22;q12)	CTNNB1	3p22.1		PLAG1	8q12.1	
t(3;9)(q27;p24)	DMRT1	9p24.3		BCL6	3q27.3	FRA3C (common, apc)
t(1;1)(p36;q41)	DUSP10	1q41		PRDM16	1p36.32	FRA1A (common, apc)
t(5;12)(q33;q14)	EBF1	5q33.3		LOC204010	12q14.3	
t(X;21)(q25;q22)	ELF4	Xq25		ERG	21q22.2	
t(9;14)(q34;q32)	EML1	14q32.2		ABL1	9q34.12	
inv(2)(p21p23), del(2)(p21p23)*	EML4	2p21		ALK	2p23.2-p23.1	
t(5;12)(q33;p13)	ERC1	12p13.33		PDGFRB	5q33.1	
t(10;12)(q11;p13)	ERC1	12p13.33		RET	10q11.21	FRA10G (common, apc)
t(9;12)(q34;p13)	ETV6	12p13.2		ABL1	9q34.12	
t(1;12)(q25;p13)	ETV6	12p13.2		ABL2	1q25.2	
t(5;12)(q31;p13)	ETV6	12p13.2		ACSL6	5q31.1	FRA5C (common, apc)
t(1;12)(q21;p13)	ETV6	12p13.2		ARNT	1q21.2	FRA1F (common, apc)
t(12;12)(p13;q13)	ETV6	12p13.2		BAZ2A	12q13.3	
t(12;13)(p13;q12)	ETV6	12p13.2		CDX2	13q12.2	
t(3;12)(q26;p13)	ETV6	12p13.2		EVI1	3q26.2	
t(4;12)(p16;p13)	ETV6	12p13.2		FGFR3	4p16.3	
t(12;13)(p13;q12)	ETV6	12p13.2		FLT3	13q12.2	
t(6;12)(q22;p13)	ETV6	12p13.2		FRK	6q22.1	
t(10;12)(q24;p13)	ETV6	12p13.2		GOT1	10q24.2	FRA10A (rare, folic acid)
t(9;12)(p24;p13)	ETV6	12p13.2		JAK2	9p24.1	
t(3;12)(q26;p13)	ETV6	12p13.2		MDS1	3q26.2	
t(1;12)(p36;p13)	ETV6	12p13.2		MDS2	1p36.11	FRA1A (common, apc)
t(12;15)(p13;q25)	ETV6	12p13.2		NTRK3	15q25.3	
t(4;12)(q12;p13)	ETV6	12p13.2		PDGFRA	4q12	FRA4B (common, BrdU)

t(5;12)(q33;p13)	ETV6	12p13.2		PDGFRB	5q33.1	
t(12;17)(p13;p13)	ETV6	12p13.2		PER1	17p13.1	
inv(12)(p13q15)	ETV6	12p13.2		PTPRR	12q15	
t(12;21)(p13;q22)	ETV6	12p13.2		RUNX1	21q22.12	
t(6;12)(q23;p13)	ETV6	12p13.2		STL	6q23	
t(9;12)(q22;p13)	ETV6	12p13.2		SYK	9q22.2	
t(12;22)(q13;q12)	EWSR1	22q12.2	FRA22B (common, apc)	ATF1	12q13.13	FRA12A (rare, folic acid)
t(2;22)(q33;q12)	EWSR1	22q12.2	FRA22B (common, apc)	CREB1	2q33.3	FRA2I (common,apc)
t(12;22)(q13;q12)	EWSR1	22q12.2	FRA22B (common, apc)	DDIT3	12q13.3	
t(21;22)(q22;q12)	EWSR1	22q12.2	FRA22B (common, apc)	ERG	21q22.2	
t(7;22)(p21;q12)	EWSR1	22q12.2	FRA22B (common, apc)	ETV1	7p21.2	
t(17;22)(q21;q12)	EWSR1	22q12.2	FRA22B (common, apc)	ETV4	17q21.31	
t(2;22)(q35;q12)	EWSR1	22q12.2	FRA22B (common, apc)	FEV	2q35	
t(11;22)(q24;q12)	EWSR1	22q12.2	FRA22B (common, apc)	FLI1	11q24.3	
t(9;22)(q31;q12)	EWSR1	22q12.2	FRA22B (common, apc)	NR4A3	9q31.1	
inv(22)(q12q12)	EWSR1	22q12.2	FRA22B (common, apc)	PATZ1	22q12.2	FRA22B (common, apc)
t(6;22)(p21;q12)	EWSR1	22q12.2	FRA22B (common, apc)	POU5F1	6p21.33	FRA6H (common, apc)
t(2;22)(q31;q12)	EWSR1	22q12.2	FRA22B (common, apc)	SP3	2q31.1	FRA2G (common, apc)
t(11;22)(p13;q12)	EWSR1	22q12.2	FRA22B (common, apc)	WT1	11p13	FRA11E (common, apc)
t(12;22)(p13;q12)	EWSR1	22q12.2	FRA22B (common, apc)	ZNF384	12p13.31	
t(5;7)(q31;q34)	FCHSD1	5q31.3		BRAF	7q34	
t(6;8)(q27;p12)	FGFR1OP	6q27		FGFR1	8p12	
del(4)(q12q12)*	FIP1L1	4q12	FRA4B (common, BrdU)	PDGFRA	4q12	FRA4B (common, BrdU)
t(4;17)(q12;q21)	FIP1L1	4q12	FRA4B (common, BrdU)	RARA	17q21.2	
t(2;13)(q36;q14)	FOXO1A	13q14.11		PAX3	2q36.1	
t(X;11)(q13;q23)	FOXO4	Xq13.1		MLL	11q23.3	FRA11B (rare, folic acid), FRA11G (common,
t(12;16)(q13;p11)	FUS	16p11.2		ATF1	12q13.13	FRA12A (rare, folic acid)
t(11;16)(p11;p11)	FUS	16p11.2		CREB3L1	11p11.2	
t(7;16)(q34;p11)	FUS	16p11.2		CREB3L2	7q33-q34	
t(12;16)(q13;p11)	FUS	16p11.2		DDIT3	12q13.3	
t(16;21)(p11;q22)	FUS	16p11.2		ERG	21q22.2	
t(2;16)(q35;p11)	FUS	16p11.2		FEV	2q35	
t(3;12)(q27;p13)	GAPDH	12p13.31		BCL6	3q27.3	FRA3C (common, apc)
t(5;12)(q33;q24)	GIT2	12q24.11	FRA12E (common, apc)	PDGFRB	5q33.1	
t(10;14)(q11;q32)	GOLGA5	14q32.12		RET	10q11.21	FRA10G (common, apc)
del(6)(q21q22)*	GOPC	6q22.2		ROS1	6q22.2	
del(8)(q12q24)*	HAS2	8q24.13	FRA8C (common, apc), FRA8E (rare, dist A)	PLAG1	8q12.1	
t(8;19)(p12;q13)	HERV-K (LOC113386)	19q13.43	FRA19A (common, 5-aza)	FGFR1	8p12	
t(5;7)(q33;q11)	HIP1	7q11.23	FRA7J (common, apc)	PDGFRB	5q33.1	
t(3;6)(q27;p22)	HIST1H4I	6p22.1		BCL6	3q27.3	FRA3C (common, apc)

inv(6)(p21q21)	HMGA1	6p21.31	FRA6H (common, apc)	LAMA4	6q21	FRA6F (common, apc)
t(12;14)(q14;q11)	HMGA2	12q14.3		CCNB1IP1	14q11.2	
t(8;12)(q22;q14)	HMGA2	12q14.3		COX6C	8q22.2	
t(2;12)(q37;q14)	HMGA2	12q14.3		CXCR7	2q37.3	FRA2J (common, apc)
t(5;12)(q33;q14)	HMGA2	12q14.3		EBF1	5q33.3	
t(12;13)(q14;q13)	HMGA2	12q14.3		LHFP	13q13.3	
t(3;12)(q28;q14)	HMGA2	12q14.3		LPP	3q28	
t(9;12)(p23;q14)	HMGA2	12q14.3		NFIB	9p23-p22.3	
t(12;14)(q14;q24)	HMGA2	12q14.3		RAD51L1	14q24.1	FRA14C (common, apc)
t(7;7)(p15;p21)	HNRPA2B1	7p15.2		ETV1	7p21.2	
t(8;10)(p11;q11)	HOOK3	8p11.21		RET	10q11.21	FRA10G (common, apc)
t(6;16)(p21;q22)	HP	16q22.3		MRPS10	6p21.1	FRA6H (common, apc)
t(3;14)(q27;q32)	HSP90AA1	14q32.31		BCL6	3q27.3	FRA3C (common, apc)
t(3;6)(q27;p21)	HSP90AB1	6p21.1	FRA6H (common, apc)	BCL6	3q27.3	FRA3C (common, apc)
t(1;14)(p22;q32)	IGH@	14q32.33		BCL10	1p22.3	FRA1D (common, apc)
t(2;14)(p16;q32)	IGH@	14q32.33		BCL11A	2p16.1	
t(14;19)(q32;q13)	IGH@	14q32.33		BCL3	19q13.31	FRA19A (common, 5-aza)
t(3;14)(q27;q32)	IGH@	14q32.33		BCL6	3q27.3	FRA3C (common, apc)
t(14;15)(q32;q11-13)	IGH@	14q32.33		BCL8	15q11.2	
t(11;14)(q13;q32)	IGH@	14q32.33		CCND1	11q13.2	FRA11A (rare, folic acid), FRA11H (common,
t(12;14)(p13;q32)	IGH@	14q32.33		CCND2	12p13.32	
t(6;14)(p21;q32)	IGH@	14q32.33		CCND3	6p21.1	FRA6H (common, apc)
t(7;14)(q21;q32)	IGH@	14q32.33		CDK6	7q21.2	FRA7E (common, apc)
t(14;19)(q32;q13)	IGH@	14q32.33		CEBPA	19q13.11	FRA19A (common, 5-aza)
t(14;20)(q32;q13)	IGH@	14q32.33		CEBPB	20q13.13	
t(8;14)(q11;q32)	IGH@	14q32.33		CEBPD	8q11.21	
t(14;14)(q11;q32)	IGH@	14q32.33		CEBPE	14q11.2	
t(14;19)(q32;q13)	IGH@	14q32.33		CEBPG	19q13.11	FRA19A (common, 5-aza)
t(12;14)(q23;q32)	IGH@	14q32.33		CHST11	12q23.3	
t(11;14)(q23;q32)	IGH@	14q32.33		DDX6	11q23.3	FRA11B (rare, folic acid), FRA11G (common,
t(7;14)(q21;q32)	IGH@	14q32.33		ERVWE1	7q21.2	FRA7E (common, apc)
t(12;14)(p13;q32)	IGH@	14q32.33		ETV6	12p13.2	
t(1;14)(q23;q32)	IGH@	14q32.33		FCGR2B	1q23.3	
t(1;14)(q21;q32)	IGH@	14q32.33		FCRL4	1q23.1	
t(4;14)(p16;q32)	IGH@	14q32.33		FGFR3	4p16.3	
t(3;14)(p14;q32)	IGH@	14q32.33		FOXP1	3p14.1	
t(6;14)(p22;q32)	IGH@	14q32.33		ID4	6p22.3	
t(14;22)(q32;q11)	IGH@	14q32.33		IGL@	22q11.22	
t(5;14)(q31;q32)	IGH@	14q32.33		IL3	5q31.1	FRA5C (common, apc)

t(6;14)(p25;q32)	IGH@	14q32.33		IRF4	6p25.3	
t(1;14)(p35;q32)	IGH@	14q32.33		LAPTM5	1p35.2	
t(1;14)(q25;q32)	IGH@	14q32.33		LHX4	1q25.2	
t(14;16)(q32;q23)	IGH@	14q32.33		MAF	16q23.1	
t(14;20)(q32;q12)	IGH@	14q32.33		MAFB	20q12	
t(14;18)(q32;q21)	IGH@	14q32.33		MALT1	18q21.32	FRA18B (common, apc)
t(1;14)(q22;q32)	IGH@	14q32.33		MUC1	1q22	
t(8;14)(q24;q32)	IGH@	14q32.33		MYC	8q24.21	
t(10;14)(q24;q32)	IGH@	14q32.33		NFKB2	10q24.32	
t(11;14)(q23;q32)	IGH@	14q32.33		PAFAH1B2	11q23.3	FRA11B (rare, folic acid), FRA11G (common,
t(9;14)(p13;q32)	IGH@	14q32.33		PAX5	9p13.2	
t(11;14)(q23;q32)	IGH@	14q32.33		PCSK7	11q23.3	FRA11B (rare, folic acid), FRA11G (common,
t(4;14)(p14;q32)	IGH@	14q32.33		RHOH	4p14	
t(14;19)(q32;q13)	IGH@	14q32.33		SPIB	19q13.33	FRA19A (common, 5-aza)
t(14;14)(q11;q32), inv(14)(q11q32)	IGH@	14q32.33		TRA@	14q11.2	
inv(14)(q11q32)	IGH@	14q32.33		TRD@	14q11.2	
t(4;14)(p16;q32)	IGH@	14q32.33		WHSC1	4p16.3	
t(14;16)(q32;q23)	IGH@	14q32.33		WWOX	16q23.1	FRA16D (common, apc)
t(1;2)(p22;p11)	IGK@	2p11.2	FRA2L (rare, folic acid)	BCL10	1p22.3	FRA1D (common, apc)
t(2;19)(p11;q13)	IGK@	2p11.2	FRA2L (rare, folic acid)	BCL3	19q13.31	FRA19A (common, 5-aza)
t(2;3)(p11;q27)	IGK@	2p11.2	FRA2L (rare, folic acid)	BCL6	3q27.3	FRA3C (common, apc)
t(2;11)(p11;q13)	IGK@	2p11.2	FRA2L (rare, folic acid)	CCND1	11q13.2	FRA11A (rare, folic acid), FRA11H (common,
t(2;12)(p11;p13)	IGK@	2p11.2	FRA2L (rare, folic acid)	CCND2	12p13.32	
t(2;7)(p11;q21)	IGK@	2p11.2	FRA2L (rare, folic acid)	CDK6	7q21.2	FRA7E (common, apc)
t(2;18)(p11;q21)	IGK@	2p11.2	FRA2L (rare, folic acid)	FVT1	18q21.33	FRA18B (common, apc)
t(2;8)(p11;q24)	IGK@	2p11.2	FRA2L (rare, folic acid)	MYC	8q24.21	
t(2;8)(p11;q24)	IGK@	2p11.2	FRA2L (rare, folic acid)	PVT1	8q24.21	
t(2;6)(p11;q25)	IGK@	2p11.2	FRA2L (rare, folic acid)	ZC3H12D	6q25.1	
t(19;22)(q13;q11)	IGL@	22q11.22-q11.23		BCL3	19q13.31	FRA19A (common, 5-aza)
t(3;22)(q27;q11)	IGL@	22q11.22-q11.23		BCL6	3q27.3	FRA3C (common, apc)
t(11;22)(q13;q11)	IGL@	22q11.22-q11.23		CCND1	11q13.2	FRA11A (rare, folic acid), FRA11H (common,
t(12;22)(p13;q11)	IGL@	22q11.22-q11.23		CCND2	12p13.32	
t(6;22)(p21;q11)	IGL@	22q11.22-q11.23		CCND3	6p21.1	FRA6H (common, apc)
t(7;22)(q21;q11)	IGL@	22q11.22-q11.23		CDK6	7q21.2	FRA7E (common, apc)
t(16;22)(q23;q11)	IGL@	22q11.22-q11.23		MAF	16q23.1	
t(8;22)(q24;q11)	IGL@	22q11.22-q11.23		MYC	8q24.21	
t(8;22)(q24;q11)	IGL@	22q11.22-q11.23		PVT1	8q24.21	
t(2;22)(p16;q11)	IGL@	22q11.22-q11.23		REL	2p16.1	
t(16;22)(q23;q11)	IGL@	22q11.22-q11.23		WWOX	16q23.1	FRA16D (common, apc)
t(4;16)(q27;p13)	IL2	4q27		DEXI	16p13.13	

t(4;16)(q27;p13)	IL2	4q27		TNFRSF17	16p13.13	
t(3;16)(q27;p12)	IL21R	16p12.1	FRA16E (rare, dist A)	BCL6	3q27.3	FRA3C (common, apc)
t(5;9)(q33;q22)	ITK	5q33.3		SYK	9q22.2	
t(6;7)(p21;p15)	JAZF1	7p15.2-p15.1		PHF1	6p21.32	FRA6H (common, apc)
t(7;17)(p15;q11)	JAZF1	7p15.2-p15.1		SUZ12	17q11.2	
t(2;17)(p23;q25)	KIAA1618	17q25.3		ALK	2p23.2-p23.1	
t(4;10)(q12;p11)	KIF5B	10p11.22		PDGFRA	4q12	FRA4B (common, BrdU)
t(10;14)(q11;q22)	KTN1	14q22.3		RET	10q11.21	FRA10G (common, apc)
t(12;16)(p13;p13)	LAG3	12p13.31		MYH11	16p13.11	FRA16A (rare, folic acid)
t(1;7)(p35;q34)	LCK	1p35.1		TRB@	7q34	
t(3;13)(q27;q14)	LCP1	13q14.12		BCL6	3q27.3	FRA3C (common, apc)
t(5;8)(p13;q12)	LIFR	5p13.1	FRA5A (common, BrdU)	PLAG1	8q12.1	
del(3)(q27q28)*	LPP	3q28		BCL6	3q27.3	FRA3C (common, apc)
t(7;19)(q34;p13)	LYL1	19p13.13	FRA19B (rare, folic acid)	TRB@	7q34	
t(11;19)(q13;q13.4)	MALAT1	11q13.1	FRA11H (common, apc)	MHLB1	19q13.4	FRA19A (common, 5-aza)
t(6;11)(p21.1;q13)	MALAT1	11q13.1	FRA11H (common, apc)	TFEB	6p21.1	FRA6H (common, apc)
t(3;18)(p21;q21)	MALT1	18q21.32	FRA18B (common, apc)	MAP4	3p21.31	

APPENDIX D

[LICENSES FOR COPYRIGHT MATERIAL USE]

NATURE PUBLISHING GROUP LICENSE
TERMS AND CONDITIONS
Apr 04, 2014

This is a License Agreement between Sheetal Shetty ("You") and Nature Publishing Group ("Nature Publishing Group") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by Nature Publishing Group, and the payment terms and conditions.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

License Number	3357810018516
License date	Mar 28, 2014
Licensed content publisher	Nature Publishing Group
Licensed content publication	Nature Reviews Genetics
Licensed content title	Sequencing technologies [mdash] the next generation
Licensed content author	Michael L. Metzker
Licensed content date	Jan 1, 2010
Volume number	11
Issue number	1
Type of Use	reuse in a dissertation / thesis
Requestor type	academic/educational
Format	electronic
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	5
High-res required	no
Figures	Figure1, Figure2, Figure3, Figure4, Figure5,
Author of this NPG article	no
Your reference number	
Title of your thesis / dissertation	Structural Variant Detection A Novel Approach
Expected completion date	Apr 2014
Estimated size (number of pages)	400
Total	0.00 USD

Terms and Conditions

Terms and Conditions for Permissions

Nature Publishing Group hereby grants you a non-exclusive license to reproduce this material for this purpose, and for no other use,subject to the conditions below:

1. NPG warrants that it has, to the best of its knowledge, the rights to license

reuse of this material. However, you should ensure that the material you are requesting is original to Nature Publishing Group and does not carry the copyright of another entity (as credited in the published version). If the credit line on any part of the material you have requested indicates that it was reprinted or adapted by NPG with permission from another source, then you should also seek permission from that source to reuse the material.

2. Permission granted free of charge for material in print is also usually granted for any electronic version of that work, provided that the material is incidental to the work as a whole and that the electronic version is essentially equivalent to, or substitutes for, the print version.Where print permission has been granted for a fee, separate permission must be obtained for any additional, electronic re-use (unless, as in the case of a full paper, this has already been accounted for during your initial request in the calculation of a print run).NB: In all cases, web-based use of full-text articles must be authorized separately through the 'Use on a Web Site' option when requesting permission.

3. Permission granted for a first edition does not apply to second and subsequent editions and for editions in other languages (except for signatories to the STM Permissions Guidelines, or where the first edition permission was granted for free).

4. Nature Publishing Group's permission must be acknowledged next to the figure, table or abstract in print. In electronic form, this acknowledgement must be visible at the same time as the figure/table/abstract, and must be hyperlinked to the journal's homepage.

5. The credit line should read:
   Reprinted by permission from Macmillan Publishers Ltd: [JOURNAL NAME] (reference citation), copyright (year of publication)
   For AOP papers, the credit line should read:
   Reprinted by permission from Macmillan Publishers Ltd: [JOURNAL NAME], advance online publication, day month year (doi: 10.1038/sj.[JOURNAL ACRONYM].XXXXX)

   Note: For republication from the *British Journal of Cancer*, the following credit lines apply.
   Reprinted by permission from Macmillan Publishers Ltd on behalf of Cancer Research UK: [JOURNAL NAME] (reference citation), copyright (year of publication)For AOP papers, the credit line should read:
   Reprinted by permission from Macmillan Publishers Ltd on behalf of Cancer Research UK: [JOURNAL NAME], advance online publication, day month year (doi: 10.1038/sj.[JOURNAL ACRONYM].XXXXX)

6. Adaptations of single figures do not require NPG approval. However, the adaptation should be credited as follows:

   Adapted by permission from Macmillan Publishers Ltd: [JOURNAL NAME]

(reference citation), copyright (year of publication)

Note: For adaptation from the *British Journal of Cancer*, the following credit line applies.
Adapted by permission from Macmillan Publishers Ltd on behalf of Cancer Research UK: [JOURNAL NAME] (reference citation), copyright (year of publication)

7. Translations of 401 words up to a whole article require NPG approval. Please visit http://www.macmillanmedicalcommunications.com for more information.Translations of up to a 400 words do not require NPG approval. The translation should be credited as follows:

   Translated by permission from Macmillan Publishers Ltd: [JOURNAL NAME] (reference citation), copyright (year of publication).

   Note: For translation from the *British Journal of Cancer*, the following credit line applies.
   Translated by permission from Macmillan Publishers Ltd on behalf of Cancer Research UK: [JOURNAL NAME] (reference citation), copyright (year of publication)

We are certain that all parties will benefit from this agreement and wish you the best in the use of this material. Thank you.
Special Terms:
v1.1

OXFORD UNIVERSITY PRESS LICENSE
TERMS AND CONDITIONS
Apr 04, 2014

This is a License Agreement between Sheetal Shetty ("You") and Oxford University Press ("Oxford University Press") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by Oxford University Press, and the payment terms and conditions.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

License Number	3361850838425
License date	Apr 04, 2014
Licensed content publisher	Oxford University Press
Licensed content publication	Nucleic Acids Research
Licensed content title	The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants:
Licensed content author	Peter J. A. Cock, Christopher J. Fields, Naohisa Goto, Michael L. Heuer, Peter M. Rice
Licensed content date	04/01/2010
Type of Use	Thesis/Dissertation
Institution name	
Title of your work	Structural Variant Detection A Novel Approach
Publisher of your work	n/a
Expected publication date	Apr 2014
Permissions cost	0.00 USD
Value added tax	0.00 USD
Total	0.00 USD
Total	0.00 USD
Terms and Conditions	

STANDARD TERMS AND CONDITIONS FOR REPRODUCTION OF MATERIAL FROM AN OXFORD UNIVERSITY PRESS JOURNAL

1. Use of the material is restricted to the type of use specified in your order details.

2. This permission covers the use of the material in the English language in the following territory: world. If you have requested additional permission to translate this material, the terms and conditions of this reuse will be set out in clause 12.

3. This permission is limited to the particular use authorized in (1) above and does not allow you to sanction its use elsewhere in any other format other than specified above, nor does it apply to quotations, images, artistic works etc that have been reproduced from other sources which may be part of the material to be used.

4. No alteration, omission or addition is made to the material without our written consent. Permission must be re-cleared with Oxford University Press if/when you decide to reprint.

5. The following credit line appears wherever the material is used: author, title, journal, year, volume, issue number, pagination, by permission of Oxford University Press or the sponsoring society if the journal is a society journal. Where a journal is being published on behalf of a learned society, the details of that society must be included in the credit line.

6. For the reproduction of a full article from an Oxford University Press journal for whatever purpose, the corresponding author of the material concerned should be informed of the proposed use. Contact details for the corresponding authors of all Oxford University Press journal contact can be found alongside either the abstract or full text of the article concerned, accessible from www.oxfordjournals.org Should there be a problem clearing these rights, please contact journals.permissions@oup.com

7. If the credit line or acknowledgement in our publication indicates that any of the figures, images or photos was reproduced, drawn or modified from an earlier source it will be necessary for you to clear this permission with the original publisher as well. If this permission has not been obtained, please note that this material cannot be included in your publication/photocopies.

8. While you may exercise the rights licensed immediately upon issuance of the license at the end of the licensing process for the transaction, provided that you have disclosed complete and accurate details of your proposed use, no license is finally effective unless and until full payment is received from you (either by Oxford University Press or by Copyright Clearance Center (CCC)) as provided in CCC's Billing and Payment terms and conditions. If full payment is not received on a timely basis, then any license preliminarily granted shall be deemed automatically revoked and shall be void as if never granted. Further, in the event that you breach any of these terms and conditions or any of CCC's Billing and Payment terms and conditions, the license is automatically revoked and shall be void as if never granted. Use of materials as described in a revoked license, as well as any use of the materials beyond the scope of an unrevoked license, may constitute copyright infringement and Oxford University Press reserves the right to take any and all action to protect its copyright in the materials.

9. This license is personal to you and may not be sublicensed, assigned or transferred by you

to any other person without Oxford University Press's written permission.

10. Oxford University Press reserves all rights not specifically granted in the combination of (i) the license details provided by you and accepted in the course of this licensing transaction, (ii) these terms and conditions and (iii) CCC's Billing and Payment terms and conditions.

11. You hereby indemnify and agree to hold harmless Oxford University Press and CCC, and their respective officers, directors, employs and agents, from and against any and all claims arising out of your use of the licensed material other than as specifically authorized pursuant to this license.

12. Other Terms and Conditions:

v1.4

NATURE PUBLISHING GROUP LICENSE
TERMS AND CONDITIONS
Apr 04, 2014

This is a License Agreement between Sheetal Shetty ("You") and Nature Publishing Group ("Nature Publishing Group") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by Nature Publishing Group, and the payment terms and conditions.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

License Number	3361601448677
License date	Apr 03, 2014
Licensed content publisher	Nature Publishing Group
Licensed content publication	Nature Biotechnology
Licensed content title	What is dynamic programming?
Licensed content author	Sean R Eddy
Licensed content date	Jul 1, 2004
Volume number	22
Issue number	7
Type of Use	reuse in a dissertation / thesis
Requestor type	academic/educational
Format	print and electronic
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
High-res required	no
Figures	Figure 1
Author of this NPG article	no
Your reference number	
Title of your thesis / dissertation	Structural Variant Detection A Novel Approach
Expected completion date	Apr 2014
Estimated size (number of pages)	400
Total	0.00 USD

Terms and Conditions

Terms and Conditions for Permissions

Nature Publishing Group hereby grants you a non-exclusive license to reproduce this material for this purpose, and for no other use,subject to the conditions below:

1. NPG warrants that it has, to the best of its knowledge, the rights to license

reuse of this material. However, you should ensure that the material you are requesting is original to Nature Publishing Group and does not carry the copyright of another entity (as credited in the published version). If the credit line on any part of the material you have requested indicates that it was reprinted or adapted by NPG with permission from another source, then you should also seek permission from that source to reuse the material.

2. Permission granted free of charge for material in print is also usually granted for any electronic version of that work, provided that the material is incidental to the work as a whole and that the electronic version is essentially equivalent to, or substitutes for, the print version.Where print permission has been granted for a fee, separate permission must be obtained for any additional, electronic re-use (unless, as in the case of a full paper, this has already been accounted for during your initial request in the calculation of a print run).NB: In all cases, web-based use of full-text articles must be authorized separately through the 'Use on a Web Site' option when requesting permission.

3. Permission granted for a first edition does not apply to second and subsequent editions and for editions in other languages (except for signatories to the STM Permissions Guidelines, or where the first edition permission was granted for free).

4. Nature Publishing Group's permission must be acknowledged next to the figure, table or abstract in print. In electronic form, this acknowledgement must be visible at the same time as the figure/table/abstract, and must be hyperlinked to the journal's homepage.

5. The credit line should read:
   Reprinted by permission from Macmillan Publishers Ltd: [JOURNAL NAME] (reference citation), copyright (year of publication)
   For AOP papers, the credit line should read:
   Reprinted by permission from Macmillan Publishers Ltd: [JOURNAL NAME], advance online publication, day month year (doi: 10.1038/sj.[JOURNAL ACRONYM].XXXXX)

   Note: For republication from the *British Journal of Cancer*, the following credit lines apply.
   Reprinted by permission from Macmillan Publishers Ltd on behalf of Cancer Research UK: [JOURNAL NAME] (reference citation), copyright (year of publication)For AOP papers, the credit line should read:
   Reprinted by permission from Macmillan Publishers Ltd on behalf of Cancer Research UK: [JOURNAL NAME], advance online publication, day month year (doi: 10.1038/sj.[JOURNAL ACRONYM].XXXXX)

6. Adaptations of single figures do not require NPG approval. However, the adaptation should be credited as follows:

   Adapted by permission from Macmillan Publishers Ltd: [JOURNAL NAME]

(reference citation), copyright (year of publication)

Note: For adaptation from the *British Journal of Cancer*, the following credit line applies.
Adapted by permission from Macmillan Publishers Ltd on behalf of Cancer Research UK: [JOURNAL NAME] (reference citation), copyright (year of publication)

7. Translations of 401 words up to a whole article require NPG approval. Please visit http://www.macmillanmedicalcommunications.com for more information.Translations of up to a 400 words do not require NPG approval. The translation should be credited as follows:

   Translated by permission from Macmillan Publishers Ltd: [JOURNAL NAME] (reference citation), copyright (year of publication).

   Note: For translation from the *British Journal of Cancer*, the following credit line applies.
   Translated by permission from Macmillan Publishers Ltd on behalf of Cancer Research UK: [JOURNAL NAME] (reference citation), copyright (year of publication)

We are certain that all parties will benefit from this agreement and wish you the best in the use of this material. Thank you.
Special Terms:
v1.1

APPENDIX E

[TOOL SCRIPTS]

Extraction algorithm

```
#!/usr/bin/perl -w

use strict;
use warnings;

my $infile_name=shift(@ARGV);
my $outfile1_name=$infile_name;
my $outfile2_name=$infile_name;
# check file extension is .sam or .bam
if($infile_name=~/.(s|b)am$/){
   $outfile1_name=~s/.(b|s)am$/.extraction2.sam/;  # add  .extraction.sam suffix to output file
   $outfile2_name=~s/.(b|s)am$/.extraction2.fa/; # add .extraction.fq suffix to output file
}else{
   die "Error: input file needs to be in bam/sam format.\n"; # if file extension not .sam and .bam file
then exit
}

my $isbam=($infile_name =~ /.bam$/)? 1:0; # if .bam file then return 1(true)

if($isbam){
   open(INFILEHDL, "samtools view $infile_name |") or die "$0: can't open ".$infile_name.":$!\n"; #
open bam file
}else{
   open INFILEHDL, "<".$infile_name or die "$0: can't open ".$infile_name.":$!\n"; #open sam file
}
open(OUTFILEHDL1, ">$outfile1_name") or die "$0: can't write in the output: $outfile1_name :$!\n";
# open file in overwite mode
open(OUTFILEHDL2, ">$outfile2_name") or die "$0: can't write in the output: $outfile2_name :$!\n";
# open file in overwite mode

my $readsum = 0;
my $readcount = 0;
my $prevline = "";
my $prevfld0 = "";
my $prevfld1 = "";
my $prevfld5 = "";
my $prevfld9 = "";
my $currentfld0 = "";
my $currentfld1 = "";
my $currentfld5 = "";
my $currentfld8 = "";
my $currentfld9 = "";
my $first = 0;
my $pattlen =0;
my $lpattern="";
my @meanlist =('99','163','147','83');
my @matchlist =
('73','133','89','121','165','181','101','117','153','185','69','137','77','141','67','131','115','179','81','161'
,'97','145','65','129','113','177');
while (my $LINE=<INFILEHDL>) # read line till EOF
```

```
{
  # chomp my $LINE;  # removes trailing whitespace

   my @L=split(/\t+/,$LINE); # split on white space,  + will merge multiple whitespace
   #my @L=split;
   if ($L[0]=~/^@/) # current line first character is @ then print that line and skip that line
   {
      print OUTFILEHDL1 $LINE;
      next;
   }

   $currentfld0 = "";
   $currentfld1 = "";
   $currentfld5 = "";
   $currentfld8 = "";
   $currentfld9 = "";

   if ( scalar(@L)>= 11) # array length is atleast 11
   {
      $currentfld0 = $L[0]; #Identifier
      $currentfld1 = $L[1]; #FLAG
      $currentfld5 = $L[5]; #CIGAR
      $currentfld8 = $L[8]; #TLEN
      $currentfld9 = $L[9]; #SEQUENCE

                        if ($first == 0) # build search pattern only once
                        {
                                $pattlen = int(length($currentfld9)*.30);  #30% or less based on decimal
point of identifier length
                                $Ipattern = "N" x $pattlen;
                                $first = 1;
                                print $Ipattern."\n" ; # for testing purpose
                        }

      if  (grep{$currentfld1 eq $_} @meanlist)
      {
        $readsum = $readsum + $currentfld8 ;
        $readcount = $readcount + 1 ;
      }

      if ( ($currentfld0 eq $prevfld0) and ( (grep{$currentfld1 eq $_} @matchlist) || (grep{$prevfld1 eq
$_} @matchlist) || ($currentfld5=~/S/ || $prevfld5=~/S/))  )
      {

         print OUTFILEHDL1 $prevline."\n";
         print OUTFILEHDL1 $LINE."\n";

                                if (index($prevfld9,$Ipattern)==-1 and index($currentfld9,$Ipattern)==-1 )
# if N Pattern not found in Read 1 & Read 2 then write to FA file
                                {
                                        print OUTFILEHDL2 ">".$prevfld0."<RID>1"."\n";
                                        print OUTFILEHDL2 $prevfld9."\n";
```

```
                                        print OUTFILEHDL2 ">".$currentfld0."<RID>2"."\n";
                                        print OUTFILEHDL2 $currentfld9."\n";
                                }
        }

    }  # if scalar

    $prevline = $LINE ;
    $prevfld0 = $currentfld0;
    $prevfld1 = $currentfld1;
    $prevfld9 = $currentfld9;

} # while

close OUTFILEHDL1;
close OUTFILEHDL2;
close INFILEHDL;
```

De-duplication algorithm

```
#!/usr/bin/perl -w

# Assumptions
# psl file needs to be sorted by identifier,chromosome,start position

use strict;
use warnings;

my $infile_name=shift(@ARGV);
my $outfile1_name=$infile_name;
if (not defined $infile_name)
{
  die "Error: .psl filename missing \n";
}
# check file extension is psl
if($infile_name=~/.(p)sl$/){
   $outfile1_name=~s/.(p)sl$/.sml/;  # add  .sml suffix to output file
}else{
   die "Error: input file needs to be in psl format.\n"; # if file extension not .psl then exit
}
open(OUTFILEHDL1, ">$outfile1_name") or die "$0: can't write in the output: $outfile1_name :$!\n";
# open file in overwite mode
open INFILEHDL, "<".$infile_name or die "$0: can't open ".$infile_name.":$!\n"; #open sam file
my @read1;
my @read2;
my $prevline = "";
my $prevfld9 = "";
my $prevfld13 = "";
my $prevfld15 = "";
my $prevfld16 = "";
my $prevfld17 = "";
my $prevrange = 0;
```

```
my $currentfld9 = "";
my $currentfld13 = "";
my $currentfld15 = "";
my $currentfld16 = "";
my $currentfld17 = "";
my $currrange = 0;


my $iline = 1;
my $processline = 0;
my $i=0;
my $j=0;
my $lastmaxrange = 0;
my @cid;
my @pid;
while (my $LINE=<INFILEHDL>) # read line till EOF
{
  my @L=split(/\t/,$LINE); # split on tab space
   $processline += 1;
   if (scalar(@L)>= 18) # array length is at least 17
   {
                        $currentfld9  = $L[9];  # Identifier
                        if(substr($L[13],0,3) eq "chr")
                        {
                                $currentfld13 = substr($L[13],3); # Chromosome
                        }
                        else
                        {
                                $currentfld13 = $L[13]; #Chromosome
                        }
                        $currentfld15 = $L[15]; # Start
                        $currentfld16 = $L[16]; # End
                        $currentfld17 = $L[17]; # Blockcount
                        #$currrange = GetRange$currentfld15,$currentfld16();
            $currrange = GetRange($currentfld15,$currentfld16);
                        @cid = split("<RID>",$currentfld9);
                        @pid = split("<RID>",$prevfld9);


                        if ( (scalar(@cid) >1)   )
                        {
                                if ($currentfld9 ne $prevfld9) # identifier not match then save
                                {
                                         $lastmaxrange = 0;          # if identifier changes then reset last
max range
                                        if ($currentfld17 eq "1") # blockcount = 1 then add to array
                                        {
                                                InsertRead();

                                        }
                                }
```

```
                        else # if same identifier then check chr range and blockcount
                        {
                                if ( $currentfld13 ne $prevfld13 ) # chromsome diff then save
                                {
                                        $lastmaxrange = 0; # if chromosome changes then reset
last max range
                                        if ($currentfld17 eq "1") # blockcount = 1 then add to
array
                                        {
                                                InsertRead();

                                        }
                                        else # if chromosome same then check range
                                        {
                                                if ( $currrange != $prevrange) # if range diff then
save
                                                {
                                                        if ($currentfld17 eq "1") # blockcount =
1 then add to array
                                                        {
                                                                InsertRead();

                                                        }
                                                }
                                        }
                                }
                        }
                        }
        } #scalar L

} # eof

close OUTFILEHDL1;
close INFILEHDL;

sub GetRange
{
        my $rangespan = 1000;
        my $grange = 0;
        my $st = 0;
        my $ed = 0;
        $st = $_[0];
        $ed = $_[1];
        my $midpoint = int(($ed+$st)/2);

        if ($midpoint < $rangespan)
        {
                $grange = 1;
        }
        else
        {
         $grange = int($midpoint/$rangespan) + 1;
```

```
        }
        return $grange ;
}

sub InsertRead
{
        if ($currrange >$lastmaxrange)
        {
                print OUTFILEHDL1
$currentfld9."\t".$currentfld13."\t".$currentfld15."\t".$currentfld16."\t".$currrange."\n" ;
                $lastmaxrange = $currrange ;

                $prevline  = my $LINE;
                $prevfld9  = $currentfld9;
                $prevfld13 = $currentfld13;
                $prevfld15 = $currentfld15;
                $prevfld16 = $currentfld16;
                $prevfld17 = $currentfld17;
                $prevrange = $currrange;
        }
}
```

Create_matrix algorithm

```
#!/usr/bin/perl -w


# Assumption
# sml file is ordered  by Idenitifier,chromosome

use strict;
use warnings;

my $infile_name=shift(@ARGV);
my $outfile1_name=$infile_name;
if (not defined $infile_name)
{
  die "Error: .sml filename missing \n";
}
# check file extension is sml
if($infile_name=~/.(s)ml$/){
   $outfile1_name=~s/.(s)ml$/.unsort/;  # add  .unsort suffix to output file
}else{
   die "Error: input file needs to be in sml format.\n"; # if file extension not .sml then exit
}
open(OUTFILEHDL1, ">$outfile1_name") or die "$0: can't write in the output: $outfile1_name :$!\n";
# open file in overwite mode
open INFILEHDL, "<".$infile_name or die "$0: can't open ".$infile_name.":$!\n"; #open sml file
my @read1;
my @read2;
my $prevline = "";
```

```
my $prevfld1 = "";
my $prevfld2 = "";
my $prevfld3 = "";
my $prevfld4 = "";
my $prevfld5 = "";
my $currentfld1 = "";
my $currentfld2 = "";
my $currentfld3 = "";
my $currentfld4 = "";
my $currentfld5 = "";
my $matfilename = $outfile1_name;

my $processline = 0;
my $i=0;
my $j=0;
my @cid;
my @pid;
$matfilename =~s/.(u)nsort$/.mat/;
while (my $LINE=<INFILEHDL>) # read line till EOF
{
   chomp($LINE) ;
   my @L=split(/\t/,$LINE); # split on tab space
   $processline += 1;
   if (scalar(@L)>= 5) # array length is at least 5

   {
                    $currentfld1  = $L[0];  # Identifier
                    $currentfld2 = $L[1]; # Chromosome
                    $currentfld3 = $L[2]; # Start
                    $currentfld4 = $L[3]; # End
                    $currentfld5 = $L[4]; # range

                    @cid = split("<RID>",$currentfld1);
                    @pid = split("<RID>",$prevfld1);
                    if ( (scalar(@cid) >1) and (scalar(@pid) >1)  )
                    {
                              if ($pid[0] eq $cid[0]) # identifier match
                              {
                                        InsertRead();
                              }
                              else # if different match then save array to file
                              {
                                        CreateMatrix();
                                        @read1 =(); #   $#read1 = -1 # clear array
                                        @read2 =(); # clear array

                                        InsertRead();
                    }
                    }
                    if (($processline == 1) and (scalar(@cid) >1) ) # save first line to array
                    {
                              InsertRead();
```

```
                }
        } #scalar L
        $prevline  = $LINE;
        $prevfld1  = $currentfld1;
        $prevfld2 = $currentfld2;
        $prevfld3 = $currentfld3;
        $prevfld4 = $currentfld4;
        $prevfld5 = $currentfld5;
} # eof

CreateMatrix(); #save array to file remaining ones
close OUTFILEHDL1;
close INFILEHDL;

# do unix sort for field 1,Field3,Field2,Field4  here on the mat file
 system ("sort -k1n,1 -k3n,3 -k2n,2 -k4n,4 $outfile1_name > $matfilename");



sub CreateMatrix
{
        my @ln1;
        my @ln2;

        my $cnt = 1;
        foreach my $r1(@read1) # r1 and r2 combination
        {
                foreach my $r2(@read2)
                {
                @ln1 = split(/\t/,$r1);
                        @ln2 = split(/\t/,$r2);
                        if ($ln1[0] eq $ln2[0] and $ln1[1] eq $ln2[1]) # if chromosome and offset is
same then do not print
                        {
                                # do not print
                        }
                        else
                        {
                                 if (PivotFile($ln1[0],$ln2[0],$ln1[1],$ln2[1]) == 2)
                {
                     print OUTFILEHDL1 $r1."\t".$r2."\n";
                }
                else
                {
                     print OUTFILEHDL1 $r2."\t".$r1."\n";
                }
                        }
                }

        }

        for(my $x=0; $x < scalar(@read1); $x++) # r1 unique combination
```

```
        {
                for(my $j=$cnt; $j < scalar(@read1); $j++)
                {
            @ln1 = split(/\t/,$read1[$x]);
            @ln2 = split(/\t/,$read1[$j]);
            if ($ln1[0] eq $ln2[0] and $ln1[1] eq $ln2[1]) # if chromosome and offset is same then do
not print
            {
                # do not print
            }
            else
            {
                if (PivotFile($ln1[0],$ln2[0],$ln1[1],$ln2[1]) == 2)
                                {
                                        print OUTFILEHDL1 $read1[$x]."\t".$read1[$j]."\n";
                                }
                                else
                                {
                                        print OUTFILEHDL1 $read1[$j]."\t".$read1[$x]."\n";
                                }
            }

                }
                $cnt++;
        }
        $cnt = 1;
    for(my $x=0; $x < scalar(@read2); $x++) # r2 unique combination
    {
        for(my $j=$cnt; $j < scalar(@read2); $j++)
        {
            @ln1 = split(/\t/,$read2[$x]);
            @ln2 = split(/\t/,$read2[$j]);
            if ($ln1[0] eq $ln2[0] and $ln1[1] eq $ln2[1]) # if chromosome and offset is same then do
not print
            {
                # do not print
            }
            else
            {
                                if (PivotFile($ln1[0],$ln2[0],$ln1[1],$ln2[1]) == 2)
                {
                    print OUTFILEHDL1 $read2[$x]."\t".$read2[$j]."\n";
                }
                else
                {
                    print OUTFILEHDL1 $read2[$j]."\t".$read2[$x]."\n";
                }
            }

        }
        $cnt++;
    }
```

```
}

sub InsertRead
{
        if($cid[1] eq "1") # read 1
        {
           push @read1,  $currentfld2."\t".$currentfld5;
        }
        if($cid[1] eq "2") # read 2
        {
           push @read2,  $currentfld2."\t".$currentfld5;
        }
}

sub PivotFile
{
    my $arg1 ="";
    my $arg2 ="";
        my $arg3 ="";
        my $arg4 ="";
    $arg1 = $_[0];  # chr1
    $arg2 = $_[1];  # chr2
        $arg3 = $_[2];  #  offset1
        $arg4 = $_[3];  #  offset2
    my $rw;
    my $co;
    if (uc($arg1) eq 'X' or uc($arg1) eq 'Y')
    {
        $rw=0;
    }
    else
    {
        $rw = int($arg1);
    }
    if (uc($arg2) eq 'X' or uc($arg2) eq 'Y')
    {
        $co=0;
    }
    else
    {
        $co = int($arg2);
    }

    my $pvot = 2;  # default 2 = no 1 = yes

    if ($rw == 0 and $co == 0) # both row and col are x or y
    {
        if ($arg1 gt $arg2) # row greater than col then pivot
        {
            $pvot = 1;
        }
    }
```

```
    else
    {
        if ($rw == 0 and $co > 0)  # row is x or y , col is number then  pivot
        {
            $pvot = 1;
        }
    }

    if ($rw > 0 and $co > 0)       # row and col both are numbers
    {

        if ($rw > $co)  # row is greater than col then pivot
        {
            $pvot = 1;

        }
    }

        if ($arg1 eq $arg2)  # if and row and col is same
        {

                if (int($arg3) > int($arg4)) # if row offset is greater than col offset
                {
                        $pvot = 1;
                }

        }

 return $pvot;
}
```

Write_count algorithm

```
#!/usr/bin/perl -w
# Assumption
# mat file is ordered by field1,field3,field2,field4
use strict;
use warnings;

my $infile_name=shift(@ARGV);
my $outfile1_name=$infile_name;
if (not defined $infile_name)
{
  die "Error: .sml filename missing \n";
}
# check file extension is mat
if($infile_name=~/.(m)at$/){
   $outfile1_name=~s/.(m)at$/.chr/;  # add  .chr suffix to output file
}else{
   die "Error: input file needs to be in mat format.\n"; # if file extension not .mat then exit
}
```

```
open(OUTFILEHDL1, ">$outfile1_name") or die "$0: can't write in the output: $outfile1_name :$!\n";
# open file in overwite mode
open INFILEHDL, "<".$infile_name or die "$0: can't open ".$infile_name.":$!\n"; #open mat file
my $prevline = "";

my $iline = 1;
my $cnt = 0;
my $first = 1;
while (my $LINE=<INFILEHDL>) # read line till EOF
{
      chomp($LINE);
            if ($first == 1)
            {
               $prevline = $LINE;
               $first = 0;
            }
            if ($prevline eq $LINE)
            {
                        $cnt = $cnt + 1;
      }
            else
            {
                        WriteCount();
                        $cnt = 1;
            }
            $prevline  = $LINE;
} # eof

WriteCount(); #save  remaining ones
close OUTFILEHDL1;
close INFILEHDL;

sub WriteCount
{
         print OUTFILEHDL1 $prevline."\t".$cnt."\n";
}
```

Get Hi-C Score algorithm

```
#!/usr/bin/perl -w
use strict;
use warnings;

# Assumptions chr file is sorted by field1,field3,field2,field4
# Hi -c file  range is 1000000 and they are tab delimited


my $infile_name = shift(@ARGV);
my $hicfile_path = shift(@ARGV);

my $outfile1_name = $infile_name;
# check file extension is chr
```

```
if($infile_name=~/.(c)hr$/){
   $outfile1_name=~s/.(c)hr$/.score/;  # add  .score suffix to output file
}else{
   die "Error: input file needs to be in chr format.\n"; # if file extension not .chr then exit
}

if (not defined $hicfile_path )
{
  die "Error: HIC file path not found.\n";
}

open(OUTFILEHDL1, ">$outfile1_name") or die "$0: can't write in the output: $outfile1_name :$!\n";
# open file in overwite mode
open INFILEHDL, "<".$infile_name or die "$0: can't open ".$infile_name.":$!\n"; #open MAT file
my $currR1fld9 = "";
my $currR1fld13 = "";
my $currR1fld15 = "";
my $currR1fld16 = "";
my $currR2fld9 = "";
my $currR2fld13 = "";
my $currR2fld15 = "";
my $currR2fld16 = "";
my $prevR1fld13 = "";
my $prevR2fld13 = "";

my $currhicfile = "";
my $prevhicfile = "";
my $hicfile = "";
my @filearray ;
my $colmidrange = 0;
my $rowmidrange = 0;
my $rowpos = 0;
my $colpos = 0;
my $r1start =0;
my $r1end =0;
my $r2start =0;
my $r2end =0;
my $currCount="";
my $score = "";
my $pivot = 2;
while (my $LINE=<INFILEHDL>) # read line till EOF
{
        chomp($LINE);
        my @L=split(/\t/,$LINE); # split on tab space
        $currR1fld9 = "";
        $currR1fld13 = "";
        $currR1fld15 = "";
        $currR1fld16 = "";
        $currR2fld9 = "";
        $currR2fld13 = "";
        $currR2fld15 = "";
        $currR2fld16 = "";
```

```
	$currCount = "";
	$r1start=0;
	$r1end=0;
	$r2start=0;
	$r2end=0;
	$score = "" ;
	$pivot = 2;

	if (scalar(@L)>= 5) # array length is at least 7
  {
		$currR1fld9  = $L[0]; # Chromosome
		$currR1fld13 = $L[1]; # Column Offset
		$currR2fld9  = $L[2]; # Chromosome
		$currR2fld13 = $L[3]; # Column Offset
		$currCount = $L[4];  # Chromosome Count


		if ($pivot == 1) # if row chr greater than col chr then swap chr position
HIC_gm06690_chr2_chr18_1000000_pearson.txt
		{
			$currhicfile =
"HIC_gm06690_chr".uc($currR2fld9)."_"."chr".uc($currR1fld9)."_1000000_pearson.txt";  # file name
will chr21_chr22.hic
		}
		else
		{
			$currhicfile =
"HIC_gm06690_chr".uc($currR1fld9)."_"."chr".uc($currR2fld9)."_1000000_pearson.txt";  # file name
will chr21_chr22.hic
		}

		if ($currhicfile ne $prevhicfile)  # if previous file not same as current file open file
		{
		 openHiCFile(); #open file
		}
		$rowpos = 0;
		$colpos = 0;
		$rowmidrange = CalcMidPosition($currR1fld13);
		$colmidrange = CalcMidPosition($currR2fld13) ;


		if ($pivot == 1)
		{
			$rowpos = GetFilePosition($colmidrange);
			$colpos = GetFilePosition($rowmidrange);
		}
		else
		{
	$rowpos = GetFilePosition($rowmidrange);
	$colpos = GetFilePosition($colmidrange);
		}
```

```
                $score = GetHiCScore(); # get hic score
                CalcRange($currR1fld13,$currR2fld13);

                if ($score gt 0 )  # score greater than 0 then print
                {
                        if ($currCount > 1) # count is greater than 1 then print
                        {
                                print OUTFILEHDL1
"chr".$currR1fld9."\t".$r1start."\t".$r1end."\t"."chr".$currR2fld9."\t".$r2start."\t".$r2end."\t".$curr
Count."\t".$score."\n";
                        }
                }

        }

        $prevR1fld13 = $currR1fld13;
        $prevR2fld13 = $currR2fld13;
        $prevhicfile = $currhicfile;
}

close OUTFILEHDL1 ;
close INFILEHDL ;


# open file and load into array
sub openHiCFile
{
        $hicfile =  $hicfile_path.$currhicfile;
        @filearray = ();
        open INFILEHIC, "<".$hicfile or die "$0: can't open ".$hicfile.":$!\n"; #open hi-c file
        @filearray = <INFILEHIC>;
        shift(@filearray);
        close INFILEHIC;
}

# get HI-C Score base on row and col position
sub GetHiCScore
{
        my @cline;
        my $scr = "Not found";
        if (scalar(@filearray) > $rowpos) # check if that range exists or not
        {
                @cline = split(/\t/,$filearray[$rowpos]);

#print "score col 0  ".$cline[0]." col 1 ".$cline[1]."\n";

                if      (scalar(@cline) > $colpos) # check if range exist or not
                {
                        $scr = $cline[$colpos];
                }
                else
                {
```

```
                                print "column offset not found
".$currhicfile."\t".$rowpos."\t".$colpos."\n";
                        }
                }
                else
                {
                        print "row offset not found ".$currhicfile."\t".$rowpos."\t".$colpos."\n";
                }
                return $scr;
}

# Calculate Position based on range (not used)
sub GetPosition
{
                my $readpos = $_[0];
                my $start = 0;
                my $end = 999999;
                my $rfactor = 1000000;  # hi-c file range span by million
                my $pos = 0;
                for (my $i = 0; $i < 1000; $i++)
                {
                        if (($readpos >= $start) and ($readpos <= $end))
                        {
                                $pos = $i ;
                                last;
                        }
                        $start += $rfactor;
                        $end += $rfactor
                }
                return $pos;
}


sub GetFilePosition
{
                my $readpos = $_[0];
                my $colspan =1000000;   #hi-c file span by million
                my $pos = 0;
                if ($readpos < $colspan)
                {
                        $pos = 1;
                }
                else
                {
                        $pos = int($readpos/$colspan) + 1;

                }
                return $pos ;
}

sub CalcMidPosition
{
```

```
	my $coloff = $_[0];
	my $chrspan = 1000;   # assuming chr file column offset span is 1000 , change this if needed
	my $cpos = 0;
	my $mend = ($coloff * $chrspan)-1;
	my $mstart = $mend - ($chrspan-1);

	$cpos = int(($mend+$mstart)/2);

	return $cpos;
}

sub CalcRange
{
	my $r1col = $_[0];
	my $r2col = $_[1];
	my $chrspan = 1000;  # assuming chr file column offset span is 1000, change this if needed

	$r1end = ($r1col * $chrspan)-1;
	$r1start = $r1end-($chrspan-1);
	$r2end = ($r2col * $chrspan)-1;
	$r2start = $r2end-($chrspan-1);

}

sub PivotFile
{
	my $arg1 ="";
	my $arg2 ="";
	$arg1 = $_[0];  # row
	$arg2 = $_[1];  # col
	my $rw;
	my $co;
	if (uc($arg1) eq 'X' or uc($arg1) eq 'Y')
	{
		$rw=0;
	}
	else
	{
		$rw = int($arg1);
	}
	if (uc($arg2) eq 'X' or uc($arg2) eq 'Y')
    {
	$co=0;
    }
	else
	{
		$co = int($arg2);
	}

	my $pvot = 2;  # default 2 = no 1 = yes

	if ($rw == 0 and $co == 0) # both row and col are x or y
```

```
	{
		if ($arg1 gt $arg2) # row greater than col then pivot
		{
			$pvot = 1;
		}
	}
	else
	{
		if ($rw == 0 and $co > 0)  # row is x or y , col is number then  pivot
		{
			$pvot = 1;
		}
	}

	if ($rw > 0 and $co > 0)	# row and col both are numbers
	{

		if ($rw > $co)  # row is greater than col then pivot
		{
			$pvot = 1;

		}
	}
	return $pvot;
}
```


180