Development of a pipeline and protocols for next generation sequencing of blood and formalin-fixed, paraffin-embedded tumour DNA samples

Submitted for the degree of Doctor of Philosophy at Cardiff University

Marc Naven

2015

Summary

Using existing software and six novel scripts, we developed a pipeline for variant calling using exome re-sequencing data of germline blood deoxyribonucleic acid (DNA) samples. We observed >99% (6,723/6,739) concordance between calls from our pipeline and previous genotyping. We identified >93% of single nucleotide variants (SNVs) and >94% of insertion/deletions called by a commercial partner using the same sequencing reads. Using a subset of genes, we showed that around half of predicted pathogenic variants could be validated *in vitro*. Together, these data showed that our pipeline was reliable for variant calling.

Next generation sequencing (NGS) of DNA from formalin-fixed, paraffin-embedded (FFPE) tumours is technically challenging. We sought to determine the sensitivity of NGS to detect known somatic hotspot mutations (n=25) in 19 FFPE advanced colorectal cancers and to optimise it for the identification of novel somatic variants. Analysis using Illumina's MiSeq software identified 100% of somatic mutations with 93% specificity, whereas the Genome Analysis Tool Kit's (GATK) HaplotypeCaller identified 80-92% of somatic mutations with 100% specificity. We investigated the background mutator profile and found that normal FFPE DNA had 3-fold more SNVs than matched blood DNA, with an excess of FFPE-associated C:G>T:A substitutions (27.1 vs. 5.3%). Uracil DNA glycosylase treatment reduced this excess. Only ~5% of variants were consistently called in replicate runs and were likely to be genuine somatic variants.

We detected potential candidates for genetic biomarkers of cetuximab-resistance in colorectal cancers that were previously shown to be wild type for *KRAS*, *NRAS*, *BRAF* and *PIK3CA*. We found that 7/21 (33%) of the CRCs analysed harboured mutations at either codon 12 or 61, whileother CRCs carried truncating *KRAS* mutations. *NRAS* also carried a codon 12 mutation in 1 CRC, and *PIK3CA* possessed mutations at codons 542 and 545. *PTEN* was found to harbour 5 truncating mutations at codons 71, 140, 155, 178 and 189, potentially leading to loss of function.

Acknowledgements

I thank the following people:

Professor Jeremy Cheadle and Dr Kevin Ashelford for their supervision and guidance throughout this PhD.

My mum, dad, brother and other family members for their continuous support; not only throughout this PhD but my life so far.

My many friends and colleagues that I've had the pleasure of knowing. Also, I thank the people of Wales for making me feel at home here.

Cancer Research Wales and their donors for funding this work.

The patients that gave consent for their samples to be used in this research.

Abbreviations

°C	Degrees Celcius
µl	Microlitre
µm	Micrometre
3’	3 prime end of a DNA molecule
5’	5 prime end of a DNA molecule
5-foU	5-formyl uracil
5-FU	5-fluorouracil
5-hmU	5-hydroxymethyl uracil
8-oxoG	7,8-dihydro-8-oxoguanine
A	Adenine
ABI	Applied Biosystems
ACD1	Assay control deoxyribonucleic acid 1
ACP1	Assay control oligonucleotide pool
aCRC	Advanced colorectal cancer
AP	Apurinic/apyrimdinic
APC	Adenomatous polyposis coli
ATP	Adenosine triphosphate
BAM	Binary/alignment mapping
BAQ	Per-base alignment quality
BER	Base excision repair
BFAST	BLAT-like fast accurate search tool
BGI	Beijing Genomics Institute
BLAST	Basic local alignment search tool
BLAT	BLAST-like alignment tool
BMP	Bone morphogenetic protein
bp	Base pairs
BWA	Burrows-Wheeler aligner
C	Cytosine

CAT	Custom amplicon oligonucleotides
CCD	Charge-coupled device
CCDS	Consensus coding sequence
CIN	Chromosome instability
cm	Centimetre
CNV	Copy number variation
COIN	Continuous vs. intermittent
COSMIC	Collection of somatic mutations in cancer
CpG	Cytosine-phosphate-guanine
CPU	Central processing unit
CRC	Colorectal cancer
D	Aspartate
dATP	Deoxyadenosine triphosphate
ddNTP	Dideoxynucleotide triphisphate
dH2O	Deionised water
DNA	Deoxyribonucleic acid
DNase	Deoxyribonuclease
dNTP	Deoxynucleotide triphosphate
DPX	Distyrene, plasticiser, xylene
Dr	Doctor
dUMP	Deoxyuridine monophosphate
E	Glutamate
EBT	Elution buffer with Tris
EDTA	Ethylenediaminetetraacetic acid
EGFR	Epidermal growth factor receptor
ELM4	Extension-ligation mix 4
emPCR	Emulsion PCR
FAP	Familial adenomatous polyposis
FdUMP	Fluorodeoxyuridine monophosphate

FFPE	Formalin-fixed, paraffin-embedded
FPU	Filter plate unit
G	Guanine
G	Glycine
GATK	Genome analysis tool kit
GB	Gigabyte
Gb	Gigabase
gDNA	Germline deoxyribonucleic acid
GRCh37	Genome Reference consortium human 37
GS FLX	Genome Sequencer FLX (trademark)
GWAS	Genome-wide association studies
H	Histidine
H_2O	Water
HNPCC	Hereditary non-polyposis colorectal cancer
HT1	Hybridisation buffer 1
IAP	Index amplification plate
ID	Identity
IGV	Integrated genomics viewer
Indel	Insertion/deletion
K	Lysine
Kb	Kilobases
k-mer	Oligonucleotide molecule of length k
LNA1	Library normalisation additives 1
LNB1	Library normalisation beads 1
LNB2	Library normalisation storage buffer 2
LNP	Library normalisation plate
LNW1	Library normalisation wash 1
mAB	Monoclonal antibody
MAF	Minor allele frequency

MALDI-TOF MS	Matrix-assisted laser desorption/ionisation time of flight mass spectrometry
MAP	MutYH-associated polyposis
Mb	Megabase
MBQ	Minimum base quality
mCRC	Metastatic colorectal cancer
ml	Millilitre
mM	Millimolar
MRC	Medical Research Council
MSI	Microsatellite instability
N	Normality
NaOH	Sodium hydroxide
nFFPE	Normal formalin-fixed paraffin-embedded
ng	Nanogram
NGS	Next generation sequencing
NHS	National Health Service
ORF	Open reading frame
OS	Overall survival
OSH2	Oligonucleotide hybridisation for sequencing 2
PCR	Polymerase chain reaction
PE	Paired-end
PFS	Progression free survival
pH	Measurement of hydrogen ion activity
PhD	Doctor of philosophy
PMM2	Polymerase chain reaction master mix 2
Q	Glutamine
R	Arginine
RCT	Randomised controlled trial
RECIST	Response evaluation criteria in solid tumours
RNA	Ribonucleic acid

RPM	Revolutions per minute
rsID	rs identifier used in dbSNP
SAM	Sequence/alignment mapping
SBH	Sequencing-by-hybridisation
SBL	Sequencing-by-ligation
SBS	Sequencing-by-synthesis
SOLiD	Sequencing by Oligonucleotide Ligation and Detection
SNP	Single nucleotide polymorphism
SNV	Single nucleotide variant
SOAP	Short oligonucleotide analysis package
SW1	Stringent wash 1
T	Thymine
TB	Terabyte
TDP1	TruSeq deoxyribonucleic acid polymerase 1
tFFPE	tumour formalin-fixed paraffin-embedded
TSCA	TruSeq custom amplicon
U	Uracil
U	Units
UB1	Universal buffer 1
UCSC	University of California, Santa Cruz
UDG	Uracil deoxyribonucleic acid glycosylase
UK	United Kingdom
US	United States
USA	United States of America
V	Valine
VCF	Variant call format
VQSR	Variant quality score recalibration
WT	Wild-type
x g	Times gravity

Contents

Chapter 1 – General introduction

1.1 DNA sequencing

Deoxyribonucleic acid (DNA) sequencing can be defined as determining the order of bases along a DNA molecule. There are several methods of achieving this, and most methods can be classified into one of two groups: traditional (Sanger and Maxam-Gilbert) and next generation sequencing (NGS). Many of the methods have advantages and disadvantages, making them suitable for particular purposes and unsuitable for others.

1.1.1 Sanger sequencing

Sanger sequencing has been the global staple method for determining the order of bases in a DNA molecule since its first publication (Sanger, et al. 1977). Sanger sequencing was used to determine the first genome sequence of an organism, a bacteriophage known as ϕX174 (Sanger, et al. 1977). At the time, sequencing was a labour intensive process with little to no automation, and DNA was sequenced at the rate of only a few hundred nucleotides per experiment. This practice dominated the sequencing landscape until several years later when semi-automated sequencers were made available, with higher throughput capabilities (Smith et al. 1986; Swerdlow & Gesteland 1990). Further technological advances were achieved upon the arrival of capillary array electrophoresis (Kim et al. 1996; Ueno & Yeung 1994; Huang et al. 1992).

These developments led to production of the single capillary sequencers such as the Applied Biosystems (ABI) Prism 310, which were widely used in the DNA sequencing community. In the late nineteen nineties, commercial 96 capillary sequencers such as the ABI Prism 3700 DNA Analyzer allowed for higher throughput in sequencing experiments.

Sanger sequencing is still commonly used today, and is heavily based on the original technique. This required obtaining multiple copies (often many millions) of the template DNA molecule to be sequenced, and can be achieved with relative ease via cloning or amplification through polymerase chain reaction (PCR). Each DNA strand has the complementary strand synthesised using short upstream priming oligonucleotides (‘primers’), deoxynucleotide triphosphates (dNTPs), dideoxynucleotide triphosphates (ddNTPs) and a DNA polymerase. Due to lack of a hydroxyl group, ddNTP incorporation into the growing complementary strand causes non-reversible chain-termination during strand extension. The result is a multitude of DNA strands of many different sizes; after denaturation and washing away of unbound primers and nucleotides, the newly synthesised strands can be sorted by their molecular weights. This is easily achieved via electrophoresis, and as their weights are directly linked to their length, shorter molecules traverse faster than longer ones. Originally, this electrophoresis step was performed on gels, using radioactively labelled ddNTPs. It was later performed in capillaries (Swerdlow & Gesteland 1990; George et al. 1997) using ddNTPs labelled with four different fluorophores (Smith et al. 1986).

Automation of Sanger sequencing has been attempted, using micro-fluidic devices that are capable of performing DNA extraction, *in vitro* amplification and sequencing (Blazej et al. 2006; Mariella 2008; Roper et al. 2007; Emrich et al. 2002). This automation, however, is not widely used for several reasons. Currently, high-throughput Sanger sequencers rely on 384 capillaries, and are therefore capable of producing up to 384 parallel sequences (Emrich et al. 2002; Shibata et al. 2000) with read lengths between 600 - 1,000 base pairs (bp) (Shendure et al. 2011; Hert et al. 2008) (Table 1.1). It is possible to obtain 90 Kilobases (Kb) of sequence data per hour using 384 capillary sequencers (Shibata et al. 2000).

Prior to production of the draft human genome (Lander et al. 2001; Venter et al. 2001), Sanger sequencing could produce a sequence at a per-base cost of approximately United States (US) $10 (Shendure et al. 2004). The efforts of producing a draft sequence pushed costs of Sanger sequencing down to less than US $0.09 per finished base (Collins et al. 2003). Current costs are around $500 per Megabase (Mb) (Table 1.1) (Kircher & Kelso 2010). Figure 1.1 shows that prior to 2007, the cost of sequencing a human genome was falling steadily. With the advent of NGS, however, there was a sharp decline in cost; this further decline meant the technique suddenly became a lot more attractive to many laboratories.

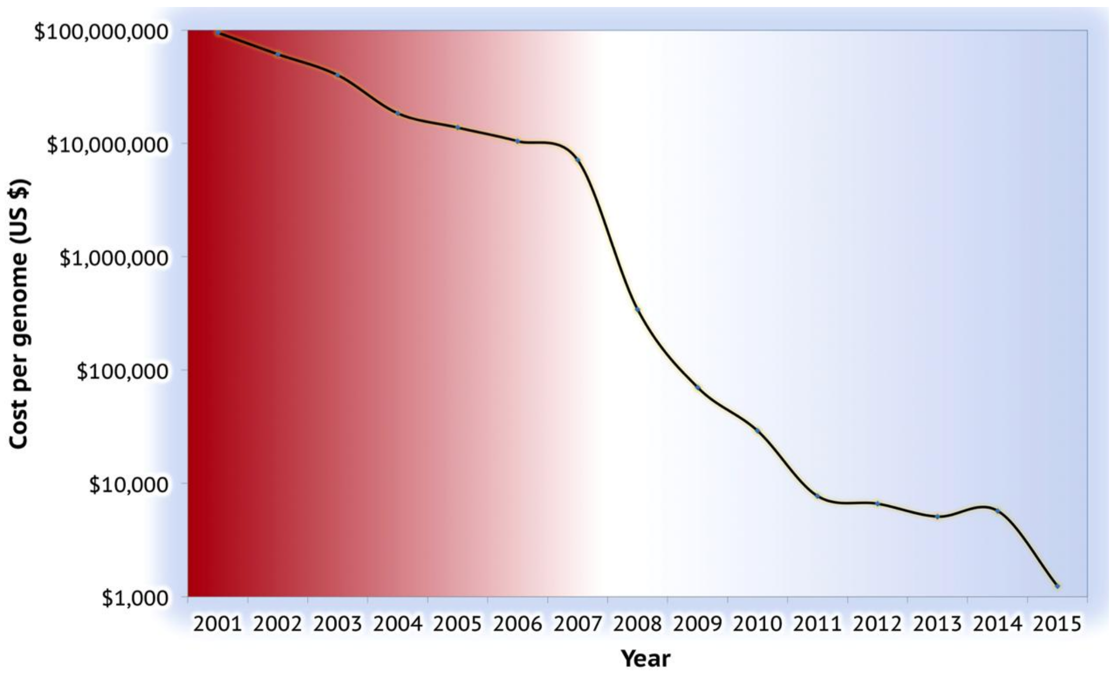


Figure 1.1. Sequencing costs for human genomes since 2001. There is a sharp drop from 2007, when the first NGS platforms became readily available on the market. Data are taken from the National Human Genome Research Institute (http://www.genome.gov/sequencingcosts/, accessed 22/07/2015).

Table 1.1. Metrics of NGS sequencing platforms vs. Sanger sequencing. Daily data generation, typical read lengths and error sources and rates are given for comparison. NGS platforms typically have a higher error rate than Sanger sequencing does, however the advantageous amount of data generated and the cost to do so outweigh these increased error rates.

Technology	**Data generated (Mb/day)**	**Typical read lengths (bp)**	**Error source(s)**	**Error rate**	**Cost per Mb (US $)**
Sanger sequencing	2 (384 capillary sequencers)	<1000 (although longer reads are possible)	• template amplification • sample contamination • polymerase slippage • low signal intensities • reduced electrophoretic separation	$10^{-4} – 10^{-5}$	500
NGS platforms					
Pyrosequencing / sequencing-by-synthesis	750	300 - 500	• template amplification • multiple templates per amplification bead • problematic signal:noise ratios for homopolymers • increasing cycle number • Signal bleed-over • phasing (uni-directional)	$10^{-3} – 10^{-4}$	20

Sequencing-by-ligation	5,000	25 - 75	• template amplification • random bead distribution (bleed-over) • dust/particle contamination • signal decline over longer reads • fluorescent label retention • poor probe hybridisation	$10^{-2} – 10^{-3}$	0.5
Reversible terminator chemistry	5,000	100 – 200 (approx)	• template amplification • Closely spaced clusters (signal interference) • phasing (bi-directional) • dust/particle contamination • increasing cycle number	$10^{-2} – 10^{-3}$	0.5
Virtual terminator chemistry	4,150	24 - 70	• weak signal emission • incorporation errors • accidental washing of hybridised strands • premature termination	>1%	0.3

1.1.2 Next generation sequencing

The demand for high-throughput, low-cost sequencing has inevitably led to the development of many ‘next generation’ sequencing techniques (Bentley et al. 2008; Clarke et al. 2009; Harris et al. 2008; Margulies et al. 2005; Shendure et al. 2005; Korlach et al. 2008). Traditional ‘first-generation’ Sanger sequencing has been outperformed by these technologies by a factor of 100 – 1000 fold, and the cost of sequencing a Mb of DNA has been reduced to between 0.1 and 4% of the cost incurred via Sanger sequencing. Current sequencing methods are based on several key techniques: capillary sequencing, pyrosequencing or sequencing-by-synthesis (SBS), sequencing-by-hybridisation (SBH), sequencing-by-ligation (SBL), reversible terminator chemistry and virtual terminator chemistry.

1.1.3 Emulsion PCR

Two major themes in template preparation for NGS are fixation of template molecules to a solid surface (i.e. a flow cell) or a bead in an emulsion PCR (emPCR) reaction. Both of these allow for millions of reactions to occur at the same time (hence the term massively parallel sequencing). In emPCR, a fragment library is created (e.g. by shearing the original DNA molecules into certain range of lengths) and universal primers are ligated to the ends of the fragments. The double stranded DNA is then separated into single strands and can be captured by beads expressing adapters complementary to one of the primer sequences. Typically, only one DNA molecule is attached to a bead, with only a single bead per emulsion reaction bubble. PCR is then performed, resulting in beads that contain many copies of their original DNA molecule. Prior to sequencing, the beads can be immobilised in a polyacrylamide gel or on a glass surface, or deposited into wells in a picotiter plate (Metzker 2010).

1.1.4 Pyrosequencing

The first NGS technology available was pyrosequencing (Ronaghi et al. 1996), and in 2008 Roche/454 Life Sciences released the GS FLX Titanium sequencer, designed to harness the new sequencing technology. In pyrosequencing, single stranded DNA templates are complemented base-by-base, and the signal emitted is read after each nucleotide incorporation (Figure 1.2). This process is often referred to as SBS. The advantage of such a technique is that the electrophoresis step required by some other techniques is no longer necessary.

A DNA library is created before pyrosequencing can take place. Each DNA template in the library is ligated to two different pre-synthesised sequence adapters; one at the 5’ end and another at the 3’ end of the template molecule (Margulies et al. 2005). One adapter is complementary to a short DNA molecule cross linked to a bead to be used in sequencing, allowing for hybridisation of DNA to the sequencing beads. The other adapter is complementary to a DNA molecule on a set of capture beads. The capture beads are used to isolate template DNA-bead complexes from beads that have no template DNA bound, so that only beads with template DNA hybridised are used in the following sequencing reaction.

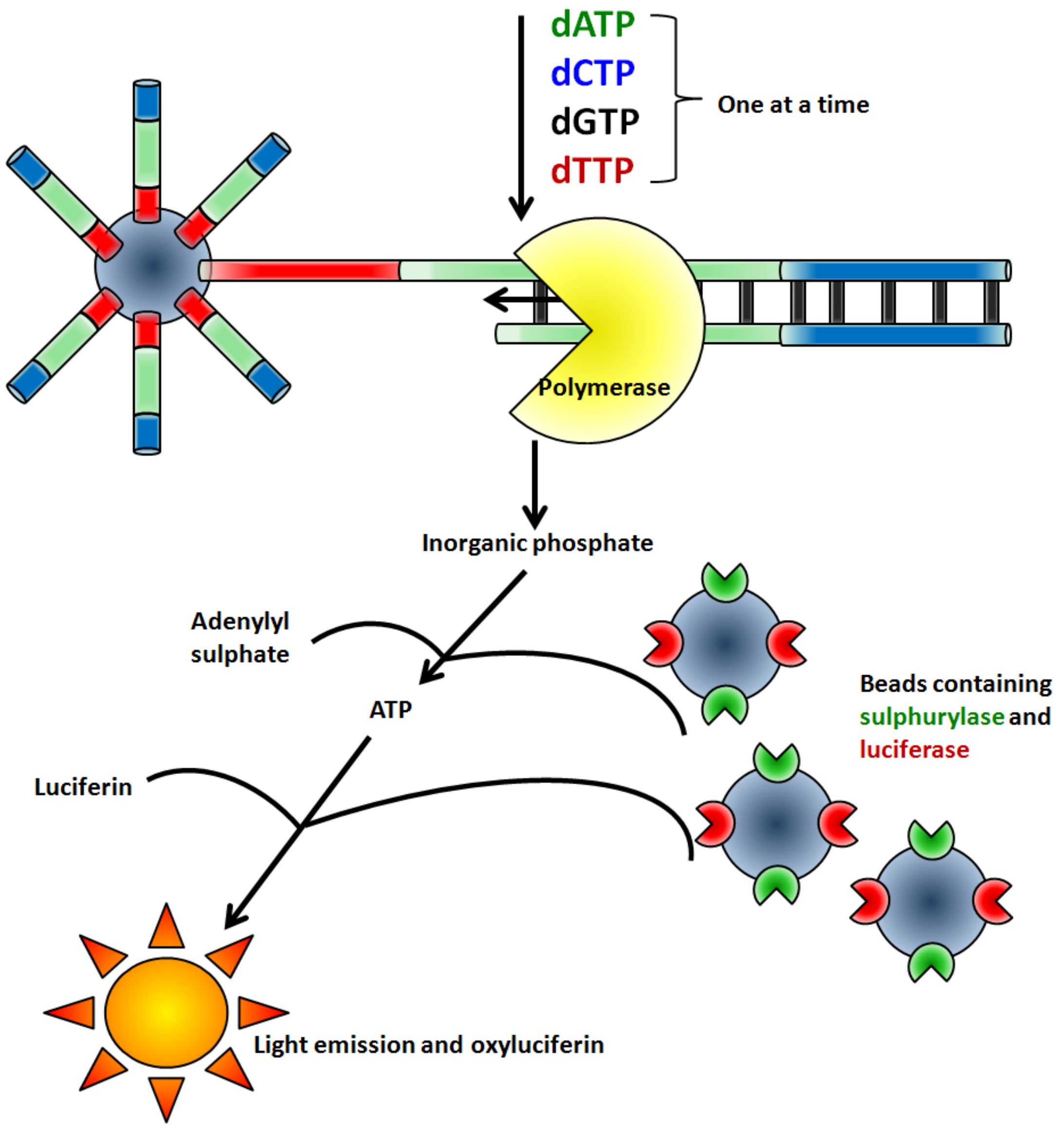


Figure 1.2. Pyrosequencing reactions are carried out on beads which are usually dispersed in picotiter plate wells. A primer attached to the beads is hybridised to the free end of each clonally amplified template strand. In each cycle, a specific dNTP is washed over the wells. If the nucleotide is complementary to the next base in the template strand, then the nucleotide is incorporated into the extending strand. This is achieved via the action of a DNA polymerase. Nucleotide incorporation results in the release of organic phosphate. Along with adenylyl sulphate and sulphurylase enzymes attached to additional beads in the well, this inorganic phosphate is used to drive production of ATP. The ATP is then used as a substrate in a reaction catalysed by bead-bound luciferase, along with luciferin, to generate oxyluciferin. The by-product of this final reaction is light emission, which is measured for each well using a charge-coupled device camera. The knowledge of which nucleotide is added per cycle allows for determination of template sequences on each bead.

During pyrosequencing, multiple copies of the sequencing template are exposed to specific nucleotides. If the nucleotide is complementary to the next available base on the template strand, then a polymerase enzyme incorporates the nucleotide into the extending strand. During the incorporation process, a pyrophosphate molecule is released and an adenosine triphosphate (ATP) sulphurylase converts this to ATP. This ATP is then used in a reaction involving luciferases, which ultimately emit a light signal, which can be measured. As the deoxyadenosine triphosphate (dATP) normally supplied for the strand extension may be used in the reaction with the luciferase, deoxyadenosine-5'-(a-thio) triphosphate is supplied instead. Luciferase cannot use this molecule as a substrate, but it can still be used in the extension step catalysed by the polymerase (Kircher & Kelso 2010). Standard versions of the other dNTPs are used as these are not normally luciferase substrates. After the light intensity emitted by the luciferase reaction has been measured, any unincorporated nucleotides are washed away and the next nucleotide is introduced so that the cycle can repeat. Light emission signals are captured in high definition using a specialised camera known as a charge-coupled device (CCD).

This reaction has been parallelised on picotiter plates (Margulies et al. 2005); these plates are composed of around two million wells, each of them the correct size to accommodate a bead 28 micrometre (µm) in diameter. The beads are coated with single stranded sequence templates, and smaller beads are also added which harbour the luciferase and ATP sulphurylase enzymes. The sequencing cycles are carried out as described, with nucleotides being washed over the plate to allow for the parallelisation to be effective.

Using the GS FLX Titanium sequencer, it is possible to sequence reads from around 1.5 million beads, with each read being 300 – 500 bases in length (Table 1.1). The final length of the sequencing reads is determined by the number of sequencing cycles performed, along with the base composition of the template molecules. Average read lengths over 400 bp may suffer from higher error rates due to decreasing enzyme efficiencies over the course of the experiment. Roughly 750 Mb of DNA can be sequenced per day via pyrosequencing (Table 1.1), incurring costs of approximately $20 per Mb (Table 1.1) (Kircher & Kelso 2010).

1.1.5 Reversible terminator chemistry

Illumina have released several next generation sequencers to date, and have adopted reversible terminator chemistry. Reversible terminator chemistry is similar to SBS used in pyrosequencing in that each sequencing cycle involves adding a labelled nucleotide to a growing DNA strand and imaging to determine the identity of the newly incorporated nucleotide (Turcatti et al. 2008; Bentley et al. 2008).

This type of sequencing relies upon generation of a DNA library, often by amplification and immobilisation of template DNA (Fedurco et al. 2006). The 5' and 3' ends of each DNA molecule have sequencing adapters ligated to them, which allow for sequencing primers to recognise and amplify the templates (Figure 1.3). The DNA is then denatured into single strands using sodium hydroxide (NaOH).

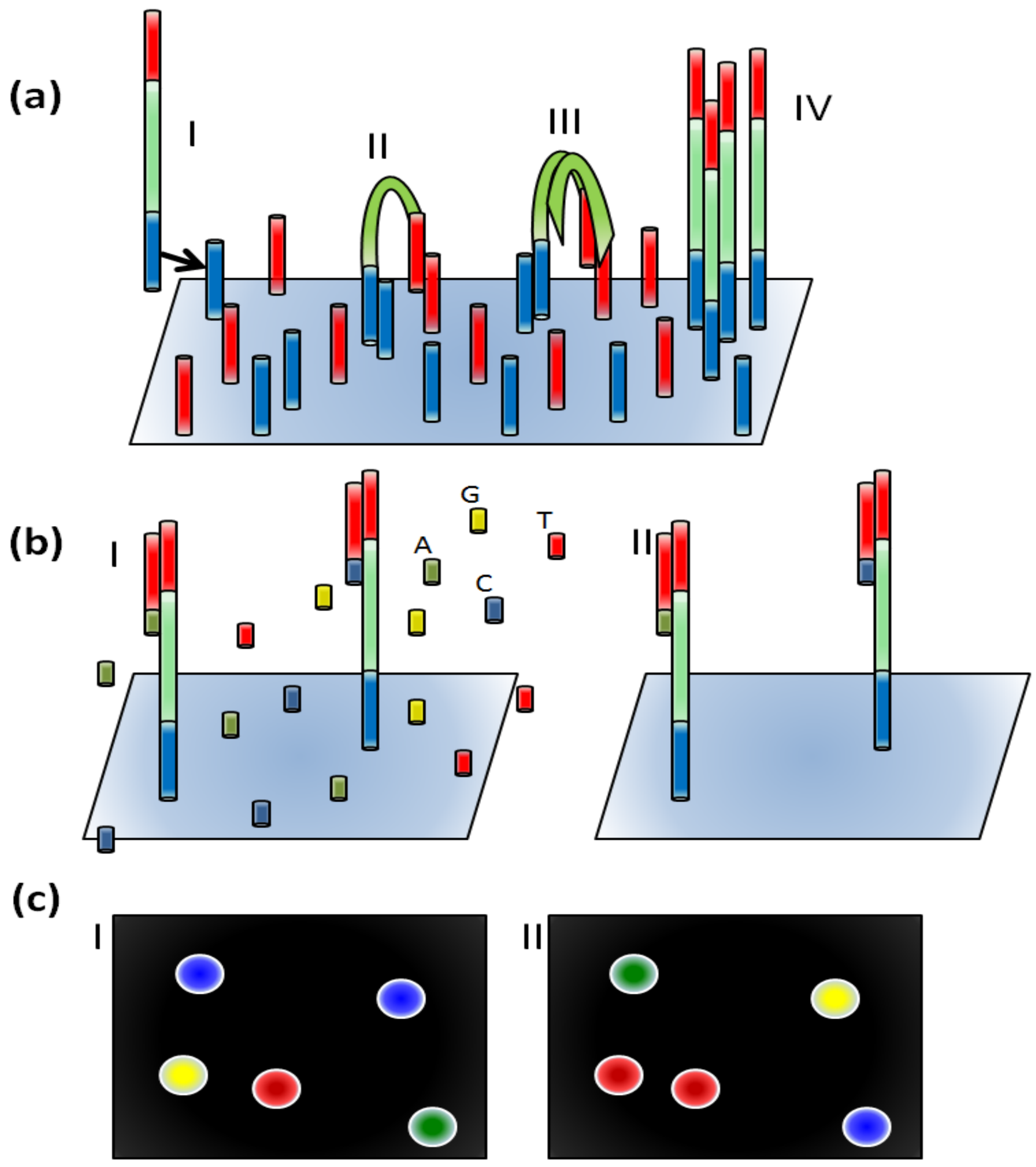


Figure 1.3. Bridge amplification and reversible terminator sequencing. (a) To create a DNA library for sequencing via reversible chain termination, bridge amplification is employed. Here, a single stranded DNA molecule is primed with universal sequencing adapters and hybridised with a sequencing adapter immobilised on a flow cell surface (I). The free end of the template molecule has the freedom to bend over and stochastically hybridise complementary adapter sequences, forming a bridge (II). Amplification is performed to generate the complementary strands (III) which is covalently linked to the flow cell in clusters of identical templates. One of the strands is cleaved, leaving multiple copies of single stranded templates arranged in clusters oriented in a single direction (IV). (b) Reversible termination sequencing relies upon incorporation of fluorescently labelled nucleotides in each cycle of sequencing. By using 4 different fluorophores, all 4 nucleotides can be added per cycle (I). Unincorporated nucleotides are washed away (II), imaging is performed, and the fluorophores and terminating groups are cleaved from the newly extended strand. (c) Each cluster will give a unique signal depending on which nucleotide was used in the extension. Each cluster generates a signal from cycle (I) to cycle (II). In paired end sequencing, the strand synthesised during the sequencing reaction is washed away, bridge amplification is repeated and this time the opposite strand is cleaved, leaving clusters oriented in the reverse direction to the previous round of cluster generation. Sequencing is then performed from the opposite end of the templates.

On platforms such as Illumina's Genome Analyzer II, the single stranded DNA is then introduced to the channels in a flow cell used by the sequencer. On the surface of the flow cell are two populations of short DNA oligonucleotides, which are complementary to the sequencing adapters at the ends of the DNA templates. Hybridisation occurs between the immobilised oligonucleotides and the adapter sequences. A strand complementary to the template is then synthesised, starting from the end hybridised to the oligonucleotide; once complete, this new strand will be immobilised to the surface of the flow cell. In a process known as bridge amplification, the newly synthesised single stranded DNA molecules are able to bend over and the adapter sequence on the free end (the end not attached directly to the flow cell's surface) can hybridise to a complementary immobilised oligonucleotide. Another new strand is synthesised that is identical to the original template DNA that is covalently linked to the flow cell at the second adapter's end. Bridge amplification is repeated multiple times, creating clusters of DNA molecules in close proximity that represent thousands of forward and reverse copies of the original template (Bentley et al. 2008; Fedurco et al. 2006). Before sequencing of these clusters can occur, either the forward or reverse strands are selectively cleaved at the immobilised oligonucleotides attached to the flow cell. All remaining strands on the flow cell then contain copies of the template in the same orientation. Hybridising a sequencing primer then allows for the sequencing reaction to begin.

This sequencing method uses a modified DNA polymerase to direct incorporation of a single reversible terminator (a 3'-O-azidomethyl 2'-deoxynucleoside triphosphate) into a growing DNA strand. These nucleotides are labelled with a different fluorophore, which can be removed after imaging. All four nucleotides are included in each cycle rather than a single nucleotide at a time, thus allowing for incorporation of the optimal nucleotide into each growing strand. Nucleotides that are not incorporated are washed away. After each incorporation step, imaging is performed to determine the identity of each newly inserted base. Imaging is carried out by exciting the fluorophore on each base using a laser. Tris(2-carboxyethyl)phosphine is then added to remove the fluorescent dye and its side arm that was attached to the incorporated base. This step also regenerates the 3’ hydroxyl group necessary for the next cycle. These cycles are repeated so that the growing strand’s sequence is determined base by base.

In the early days of this technology, millions of reads could be sequenced but with only very short read lengths, boasting a modest maximum length of 36 bp. Current flow cell densities on the Genome Analyzer allow for more than 200 million clusters to be sequenced in a single run, with a read length of up to around 100 bp. Further improvements to the system allowed the sequencing of the DNA strands in both directions; post-sequencing of the first strand, the newly synthesised strand is washed away. Bridge amplification is performed for a second time to regenerate strands in the opposite orientation before the previously sequenced strand is cleaved at the covalently bound oligonucleotide adapter. The sequencing adapters on the end of the strand to be sequenced are recognised by a sequencing primer, and sequencing is performed. Sequencing both ends of a template in such a way is known as paired end sequencing, and generates approximately twice the amount of data compared to sequencing in a single direction. Illumina sequencers such as the Genome Analyzer are capable of producing more than 5,000 Mb per day (Table 1.1) (Kircher & Kelso 2010)

1.1.6 Sequencing-by-ligation

The major conceptual difference between SBL and other NGS technologies is that SBL relies on ligase enzymes rather than polymerases. Single stranded templates in a DNA library are hybridised to sequencing primers and 8-mer probes are added, which are competitively ligated to the sequencing primer. Each probe contains a fluorescent label, the identity of which encodes the two most 3' nucleotides in the probe. The identity of the probe is revealed in imaging and by association so are the 3' nucleotides. From the 5' end, three nucleotides and the fluorescent label are cleaved from the probe. At this point, the primer has been extended by five nucleotides, and further ligation is possible due to presence of a free 5' phosphate. Up to 10 cycles of this are performed typically, giving the identity of two nucleotides out of every five in the extended strand. The extended strand is then chemically melted and washed away. A new sequencing primer, shifted in length by one base in comparison to the previous primer, is then annealed to the template. The ligation and imaging cycles are repeated, with the ligation points shifted by a single base in the sequence. Repetition using a further three sequencing primers (each shifted by a single base in comparison to the last), is performed. The fluorescence signals detected throughout these cycles can then be converted to a sequence, revealing the original template DNA's sequence (Figure 1.4). One caveat of this technique is that the first base of the sequence must be known *a priori*, and so the last base of the library adapter is used. Without a reference sequence, errors in fluorescence interpretation can cause incorrect base calling of the downstream sequence.

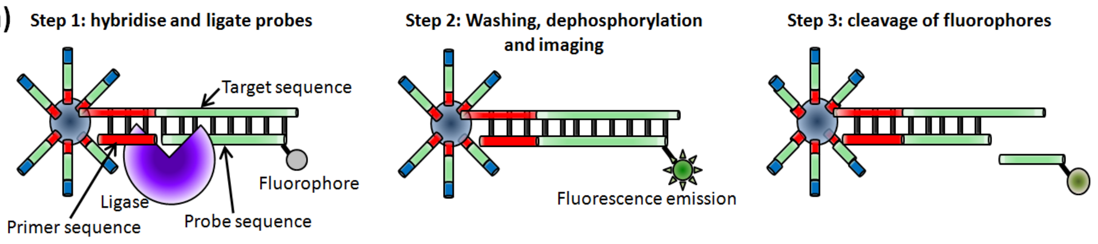
The Sequencing by Oligonucleotide Ligation and Detection (SOLiD) platform enables sequencing parallelisation in a similar fashion to pyrosequencing, in that beads are coated with multiple copies of the same template. Instead of using a picotiter plate, however, a glass slide is used as a flow cell. After modification of the 3' ends of the sequences on the beads, the sequences can be covalently bound to the flow cell.

1.1.7 Virtual terminator chemistry

Not all NGS platforms require preparation of amplified sequence libraries. The HeliScope sequencer (developed by Helicos) has the capability to sequence individual molecules without the need for *in vitro* amplification. This has a major advantage over other technologies in that amplification and other library preparation induced errors are absent from the resulting data. This platform may also be able to detect non-standard nucleotides and modifications in the template molecules, which would often be lost during amplification steps.

The technology differs from the other technologies and has been coined asynchronous virtual terminator chemistry (Harris et al. 2008). Starting double stranded DNA templates are fragmented and melted, and each has a poly-A tail synthesised onto the end using a polyadenylate polymerase. The last adenine in this homopolymer run is attached to a fluorescent label.

(a)


Step 1: hybridise and ligate probes
Target sequence
Fluorophore
Ligase
Primer sequence
Probe sequence
Step 2: Washing, dephosphorylation and imaging
Fluorescence emission
Step 3: cleavage of fluorophores

(b)

Primer round	New primer sequence	Emitted fluorescence signals after ligation cycle					Final colour-space sequence
		1	2	3	4	5	
1	3'- **AACGCCGTTG AGAAC**-5'	GATGCCCGAT ATCTT	GATGCCCGAT ATCTT	GATGCCCGAT ATCTT	GATGCCCGAT ATCTT	GATGCCCGAT ATCTT	🟢🔵🔴
2	3'- **AACGCCGTTG AGAA**-5'	**C**GATGCCCGA TATCTT	**C**GATGCCCGA TATCTT	**C**GATGCCCGA TATCTT	**C**GATGCCCGA TATCTT	**C**GATGCCCGA TATCTT	🔴🔵🔴
3	3'- **AACGCCGTTG AGA**-5'	**AC**GATGCCCG ATATCTT	**AC**GATGCCCG ATATCTT	**AC**GATGCCCG ATATCTT	**AC**GATGCCCG ATATCTT	**AC**GATGCCCG ATATCTT	🟡🔴🔴🔵
4	3'- **AACGCCGTTG AG**-5'	**AAC**GATGCCC GATATCTT	**AAC**GATGCCC GATATCTT	**AAC**GATGCCC GATATCTT	**AAC**GATGCCC GATATCTT	**AAC**GATGCCC GATATCTT	🔵🟡🟢🟢
5	3'- **AACGCCGTTG A**-5'	**GAAC**GATGCC CGATATCTT	**GAAC**GATGCC CGATATCTT	**GAAC**GATGCC CGATATCTT	**GAAC**GATGCC CGATATCTT	**GAAC**GATGCC CGATATCTT	🟢🔴🔴🟢

Dinucleotide fluorescence encoding

2nd base \ 1st base	A	C	G	T
A	🔵	🟡	🟢	🔴
C	🟡	🔵	🔴	🟢
G	🟢	🔴	🔵	🟡
T	🔴	🟢	🟡	🔵

Figure 1.4. Sequencing by ligation and decoding of colour-space reads. (a) During SBL, sequencing primers are hybridised to immobilised adapters covalently linked to the sequencing template. SBL uses five rounds of primer binding, and each round has 10 ligation cycles. In each cycle, 8-mer probes are hybridised to the template and ligated to the primer (step 1). Imaging of the probe's fluorophore gives details about the two 3' most nucleotides in the probe sequence, and hence the complementary template nucleotides (step 2). The final step cleaves the three 5' nucleotides and the fluorophore from the probe, and the next cycle is performed. After all the cycles are complete, the next primer binding round is performed. Each consecutive primer round is identical to the preceding round, except that the primer sequence is shifted by 1 base. This results in a knock-on shift in the imaged sequence. (b) Upon completion, colour-space reads are decoded to give a base-space sequence. An example template sequence (5'-GATGCCCGATATCTT-3') is demonstrated here using an example primer sequence (3'-AACGCCGTTGAGAAC-5', highlighted in bold). Adapted from (Metzker 2010; Kircher & Kelso 2010).

Washing the template library over a flow cell containing poly-T oligonucleotide probes allows for hybridisation. Probes that have template sequences bound are determined using fluorescence imaging, followed by removal of the fluorescent label from the 3’ adenine. During the sequencing reaction, a polymerase and a specific fluorescently labelled nucleotide are washed over the flow cell. Starting from the poly-T probes, the polymerase extends the complementary sequence strand. Nucleotide incorporation rate is controlled by the fluorescent labelling, with a maximum of one nucleotide incorporation per sequencing cycle. After each cycle, the polymerases and free nucleotides are washed away, virtually terminating the extension (Zhu & Waggoner 1997; Bowers et al. 2009). After this, imaging of the flow cell is performed again to detect signal emission from newly incorporated nucleotides (Figure 1.5). This is followed by removal of the fluorescent labels, completing the cycle. As not every sequence is extended in each cycle, the process is asynchronous and may result in read lengths of different sizes. Read lengths achieved via the HeliScope range from 24 – 70 bp, with an average around 32 bp (Pushkarev et al. 2009).

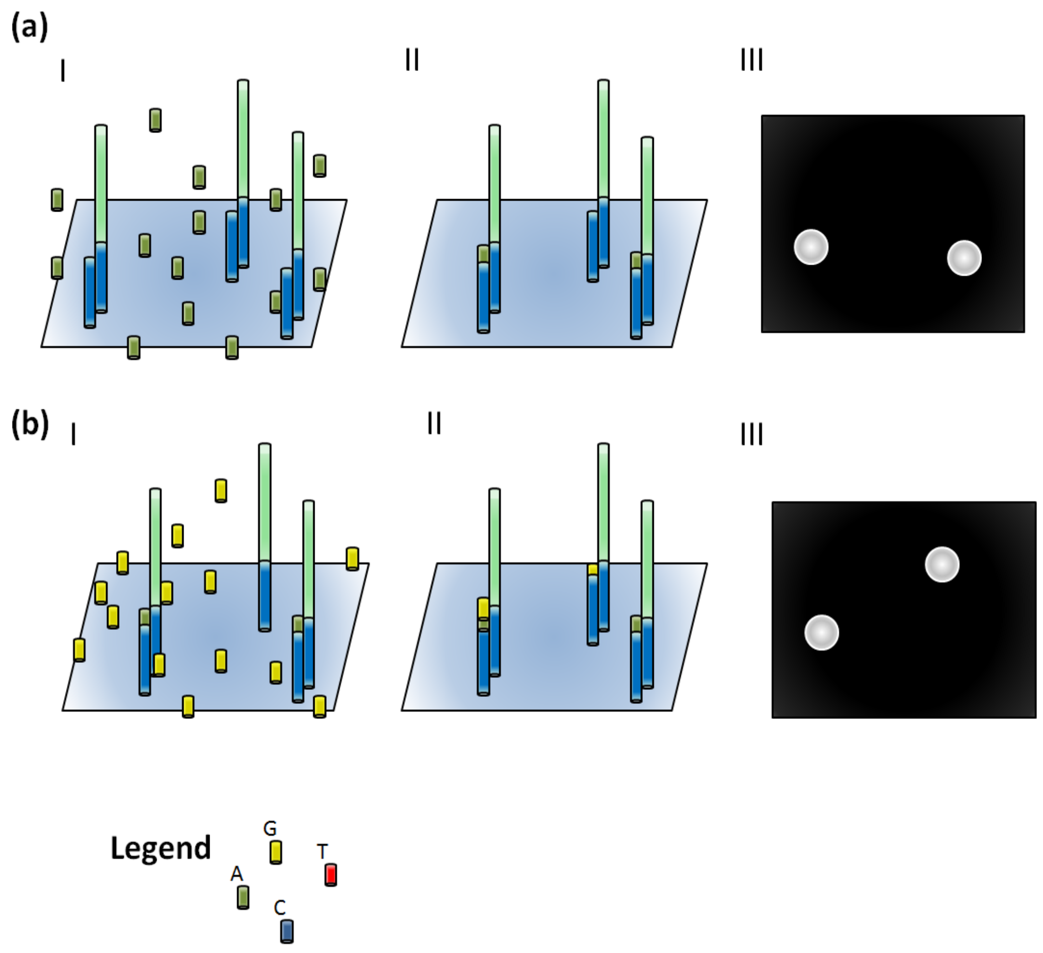


Figure 1.5. Virtual terminator chemistry. A flow cell is prepared, containing poly-T probes. Template sequences ligated to poly-A oligonucleotides are hybridised to the poly-T probes. (a) A single type of nucleotide (adenine [A], cytosine [C], guanine [G] or thymine [T]) is then washed over the flow cell (step I) and a polymerase extends the poly-T tail if the nucleotide is complementary to the next base in the template. Unused nucleotides are washed away, leaving only the hybridised template and the (extended) probes (step II). Single-colour imaging is performed, with the fluorescent dyes on the recently added nucleotides emitting a light signal (step III). The process is then repeated using a different nucleotide (b). Given that the nucleotides are added in a known order, the sequence of the templates can be determined by reading out the fluorescence signals.

1.1.8 Sequencing-by-hybridisation

Studies have demonstrated that sequencing using an SBH approach (Drmanac et al. 1998) can lead to sensitive detection of point mutations in clinically relevant genes (e.g. *HRAS*) (Healey et al. 1997). SBH can be performed in two distinct ways. In the first method, an array of DNA targets of unknown sequence are bound to a substrate and then short labelled oligonucleotides of known sequence are hybridised to the targets. Each round of hybridisation allows for detection of complementary DNA sequences on the templates. By using the reference sequence as a guide, short oligonucleotides that share overlapping sequence can be theoretically assembled to determine the longer template sequence (Figure 1.6). The second method of SBH relies on immobilising the probes of known sequence to a substrate; the probes are then allowed to hybridise to the labelled target DNA whose sequence is to be determined.

Detection of probe-target duplexes can be done with relative ease if the probes are labelled with a fluorophore, for example fluroescein isothiocyanate. Using this approach, duplexes can be detected using fluorescence imaging equipment (Sassolas et al. 2008).

It was theorised (Drmanac et al. 1989) that over 1 Mb of unknown DNA sequence could be determined using mixes of 11 bp long oligonucleotide probes (known as *k*-mers, where *k* is the length of the sequence; in this case 11-mers). Even in the dawn of NGS, it was recognised that extensive computing power would be required for analysis of the data. Fortunately, it was also recognised that the inherent parallelism in production of the data would provide worthy advantages over pre-existing methods of DNA sequencing (Drmanac et al. 1989).

1.1.9 Error rates

1.1.9.1 Sanger Sequencing

Sanger sequencing boasts errors that often arise during *in vitro* amplification of the template DNA, which is much higher than *in vivo*. Sample contamination is another source of sequencing errors, as is polymerase slippage caused by low sequence complexity (e.g. short variable number repeats and homopolymers). Long sequencing reads suffer from errors towards the end of the sequence, caused by lower signal intensities (Kircher & Kelso 2010). It is also known that base miscalls (Ewing et al. 1998) and deletions increase in number with read length, possibly due to reduced separation during electrophoresis. On average, Sanger sequencing has a low error rate, with errors estimated to occur at a rate of one per 10,000 – 100,000 bases (Ewing & Green 1998) (Table 1.1).

1.1.9.2 Pyrosequencing

Single nucleotide substitution error rates for pyrosequencing lie in the range of 10^{-3} – 10^{-4} (Quinlan et al. 2008; Margulies et al. 2005), which means pyrosequencing has a higher error rate than Sanger sequencing, but lower than for other NGS technologies (Table 1.1). The source of many of these errors may be *in vitro* amplification, although this step is a requirement for pyrosequencing on the GS FLX Titanium sequencer.

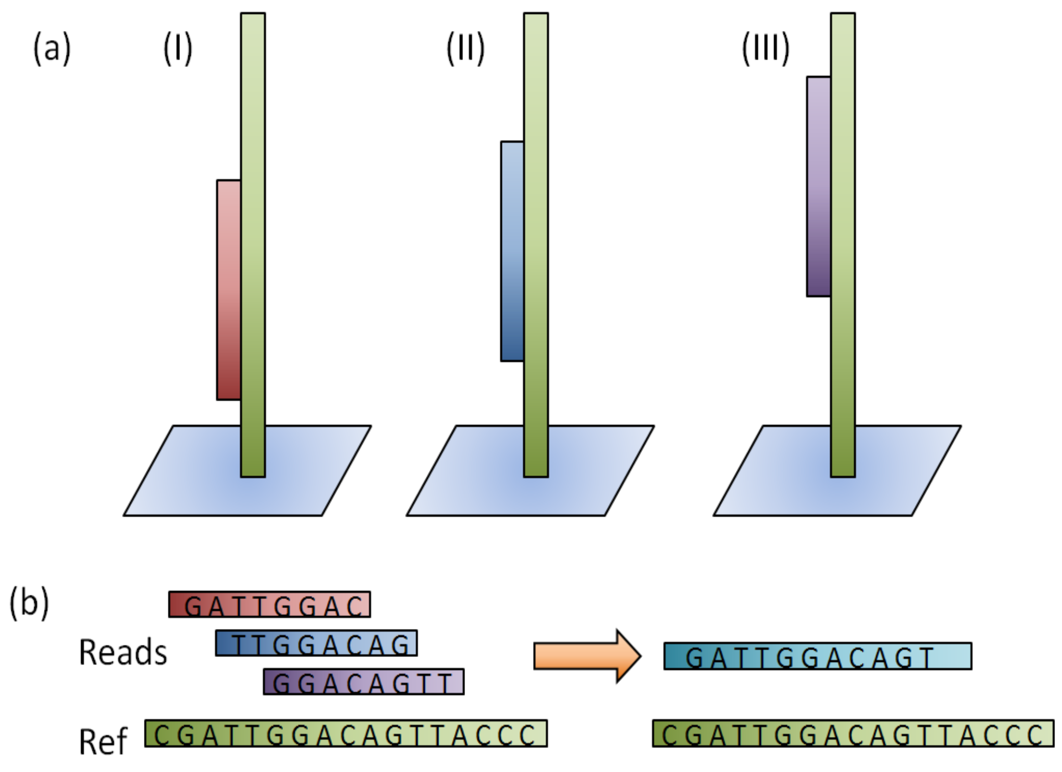


Figure 1.6. SBH is performed by fixing template sequence to a substrate (a). Short oligonucleotides of known sequence (8-mers are shown here) are added one at a time and hybridisation occurs between complementary sequences (I – III). Aligning the oligonucleotides against a reference sequence (b) reveals overlapping regions between the oligonucleotides; these can be assembled into a contig that represents the sequence of the original template molecule.

During amplification using emPCR some sequencing beads may end up carrying more than one unique template molecule. During the sequencing reaction, this results in extraneous light emission, giving a false sequence as nucleotides are incorporated into each template in different cycles. It is practically impossible to distinguish which template the signal was generated by, although in theory it may be possible to filter out these sequencing reads using post-sequencing software (Kircher & Kelso 2010).

Another source of error in pyrosequencing arises from miscalling homopolymer runs, and the signal-to-noise ratio which may lead to phantom mutations, particularly insertions/deletions (indels) (Quinlan et al. 2008). Using higher sequencing coverage, these errors may become insignificant. Longer homopolymer runs of over 10 nucleotides, on the other hand, may still be an issue at higher coverage (Quinlan et al. 2008; Wicker et al. 2006; Green et al. 2008). Furthermore, high-intensity signals from one well in the plate may affect neighbouring cells (i.e. signal bleed-over), potentially creating phantom mutations in the affected wells. Sequence artefacts caused by bleed-over may be corrected by downstream computational analyses (Green et al. 2006).

The position in the sequencing read may also affect the error rate, with more errors typically appearing towards the end of the read. In pyrosequencing this is often due to enzyme efficiency reduction, which ultimately results in inhibition of sequence extension (where polymerases are affected) or lower signal intensities (in the case of luciferases and sulphurylases). Increased phasing may also be encountered; a phenomenon in which not all of the amplified (clonal) template molecules are extended at the same rate. For example, it is possible that the correct nucleotide is not incorporated into the extending strand in a given cycle, leading to the preceding base re-emitting a signal.

1.1.9.3 Reversible terminator chemistry

Average error rates of around 10^{-2} – 10^{-3} are observed using reversible terminator chemistry (Table 1.1) (Dohm et al. 2008; Kircher et al. 2009), and are probably made higher by *in vitro* amplification steps. Preparation of the flow cell may also result in some clusters than are composed of several template sequences, interfering with signal intensities during imaging. This may lead to incorrect base calls or lower confidence scores in correct calls. Furthermore, unclean flow cells may have contamination from dust, lint and other particles that could be erroneously recognised as clusters and generate ‘ghost’ sequences. These ghost sequences, however are likely to be of low quality due to the typical low sequence complexity observed (Kircher & Kelso 2010). Error rates increase towards the ends of the reads, mainly due to increased signal noise created by phasing. Phasing using reversible terminator chemistry can occur in two ways; the first as described for pyrosequencing, and the second by incorporation of more than one nucleotide in the same sequencing cycle. It is also documented that the signal intensity declines during increasing cycle numbers (Kircher et al. 2009; Erlich et al. 2008; Rougemont et al. 2008).

1.1.9.4 Sequencing-by-ligation

For SBL, higher densities can be achieved by using a glass flow cell instead of picotiter plates, although the beads are distributed randomly across the flow cell surface which can lead to problems identifying bead positions. This is especially true for unclean flow cells, as similar to flow cells used in reversible terminator chemistry, dust and other particles impose problems. Signal bleed-over from neighbouring beads can also lead to errors in base calling.

Other errors are caused by *in vitro* amplification, as with other amplification-dependent library preparation protocols. Beads that contain multiple template sequences also suffer from signal noise, similar to that experienced when beads are spaced too near one another. Signal decline along the length of the read, and failure to remove fluorescent labels also results in errors. Where labels are not removed, there is a possibility for them to be removed in the next ligation reaction, creating a sequencing artefact similar to that encountered during phasing. Phasing itself, however, is not a major problem for the SOLiD platform, as permanent termination is ensured by the act of phosphatases efficiently removing the phosphate group from strands that were not extended in the previous cycle (Kircher & Kelso 2010). Poor hybridisation of probes to the template sequence also leads to signal decline over the course of many cycles. Where a reference genome sequence is available, typical error rates fall between 10^{-3} – 10^{-4}; in cases where no reference sequence is available, error rates are between 10^{-2} and 10^{-3} (Kircher & Kelso 2010). Without a reference genome sequence, assembly and consensus base calling can be performed based on the fluorescent signals obtained (so called ‘colour space’), before the signals are converted to nucleotide sequence (‘base space’).

1.1.9.5 Virtual terminator technology

Disadvantages of virtual terminator technology are that since only single molecules are sequenced, fluorescence signals that are emitted are weak, and the lack of consensus calls using amplified templates means incorporation errors cannot be corrected as easily. The technology also relies on hybridisation to the flow cell instead of covalent bonding preferred by other technologies, meaning that the HeliScope platform is prone to template molecules being accidently removed during washing steps. Premature termination may occur via addition of erroneously synthesised nucleotides (Kircher & Kelso 2010). Error rates are generally high and appears to be biased towards indels (Pushkarev et al. 2009; Kircher & Kelso 2010). At 15x – 20x coverage, around 1% of the sequenced bases may be incorrect, and at 30x coverage, around just 0.1% of the bases sequenced are thought to be identified incorrectly (Pushkarev et al. 2009).

1.2 *De novo* genome assembly

One of the major challenges facing researchers sequencing the genome of an organism for the first time is assembling the data to reconstruct the genome accurately. Early software built to do this could handle smaller, relatively simple bacterial genomes, but the further development of advanced assembly algorithms along with increases in computational power has allowed for more complex genomes to be assembled *de novo* (Imelfort & Edwards 2009; Lin et al. 2011; Zhang et al. 2011). A hurdle in *de novo* assembly is repetitive DNA sequences that can cause ambiguity in the data. A way to bypass this hurdle is to use increased read lengths; however, this is limited by factors in sequencing chemistry and technology. Alternatively, one can use paired end or mate pair sequencing, whereby if the distance between each read is long enough then the repeat ambiguity can be resolved (Lee et al. 2012). Several pieces of software have been developed that are capable of *de novo* assembly.

1.2.1 SSAKE, VCAKE and SHARCGS

SSAKE, published in 2007 (Warren et al. 2007), was one of the first assemblers for short reads to be released. The software does not account for paired end data, and therefore may struggle with genomes with repetitive regions. SSAKE does however possess the ability to collapse multiple repetitive regions into one contig when run in non-strict mode. An extension of SSAKE, VCAKE (Jeck et al. 2007) has been reported to be able to handle erroneous reads (Lee et al. 2012). Both of these pieces of software perform at roughly comparable levels on simpler genomes, but VCAKE outperforms SSAKE for more complicated genomes. However, VCAKE also does not take advantage of paired end data and so may still struggle with repetitive regions (Jeck et al. 2007).

SHARCGS (Dohm et al. 2007) further complements the methods employed by SSAKE and VCAKE. By default, SHARCGS naive assemblies offer a range of filtering levels. Contigs from different rounds of assembly can be merged in order to increase accuracy; these merged contigs are supplied with quality scores for consideration by end users. SHARCGS does not account for paired end data and thus has difficulty creating an unambiguous set of repetitive regions (Lee et al. 2012).

1.2.2 Velvet

In 2008, a collection of NGS assembly algorithms was released, named Velvet (Zerbino & Birney 2008). Since it was first released, Velvet has undergone continuous improvements from the original authors and bioinformatics community. Velvet is able to take advantage of error correction algorithms, which may help maintain a low memory footprint (Lee et al. 2012; Zerbino & Birney 2008). The algorithms utilised by Velvet are able to account for paired end data, which are used to merge contigs and return collections of scaffolds to the user. These algorithms have been updated (Zerbino et al. 2009) and scaffold generation improved. Repetitive region resolution has also been increased, along with error correction and general speed and memory improvements. Larger genomes, however, reportedly take over 1 terabyte (TB) of memory (Lee et al. 2012), making the software unfeasible for standard desktop computers. Updates to software have been applied to increase memory performance and running times. Velvet remains a highly popular assembler, has been used in assembly of commercially important genomes such as wheat (Berkman et al. 2011) and has been cited over 300 times (Lee et al. 2012).

1.3 Aligning NGS reads to a reference sequence

NGS data must be either assembled (for *de novo* genomes or organisms) or aligned against the reference genome. Several NGS read aligners are available, such as Bowtie (Langmead et al. 2009), Burrows-Wheeler Aligner (BWA, Li & Durbin 2009), Maq (H. Li et al. 2008), the SHort Read Mapping Package (SHRiMP, Rumble et al. 2009) and the Short Oligonucleotide Alignment Programs (SOAP, SOAP2 and SOAP3, (R. Li et al. 2008; R. Li et al.

2009; C.-M. Liu et al. 2012) and each have advantages and disadvantages. Bowtie, for example, has a relatively small memory footprint of 1.3 Gigabytes (GB) for the whole human genome (Ruffalo et al. 2011). This makes it computationally less demanding and so can be run on most modern desktop computers. However, this comes at a price; if an exact match between a sequencing read and the reference genome is not found, then it is not guaranteed to find the best quality mapping coordinates for the read. Fortunately, Bowtie offers users the chance to alter usage parameters to configure the software to run as they would like.

1.3.1 BLAT

More traditional alignment tools, such as Basic Local Alignment Search Tool (BLAST) (Altschul et al. 1990) and Smith-Waterman (Smith & Waterman 1981) algorithms are unable to handle the large amounts of data typically generated throughout the course of NGS. The BLAST-like alignment tool (BLAT) (Kent 2002) has been designed for fast alignment of reads to a reference genome. It is commonly used for lookup of genomic locations by sequence similarity; in a reference the size of the human genome (>3 Gigabases [Gb]) it takes just seconds to locate a region with similar sequence identity. This rapid search ability has earned BLAT employment amongst several public genome browsers, for example Ensembl (Flicek et al. 2013) and the University of California, Santa Cruz (UCSC) Genome Browser Database (Karolchik et al. 2003). BLAT maintains a low memory footprint by using indexed genome formats, allowing for the entire human genome to be read into <1 GB of memory. BLAT has been optimised to search for sequences of ≥40 bp that are 95% identical or greater.

For shorter or less similar sequences, BLAST or other algorithms may prove to be better suited for alignment.

1.3.2 BFAST

The BLAT-like Fast Accurate Search Tool (BFAST) (Homer et al. 2009) also uses genome indexing to retain a low memory footprint and rapid alignment speed. BFAST opts for higher alignment accuracies and sensitivities whilst sacrificing alignment speed, although it is generally still faster than BLAT and has lower false-mapping rates (Lee et al. 2012). It is, however, slower than some other aligners, for example Maq, Bowtie and BWA. The gain of speed over BLAT is largely due to the improved use of the reference indexing. Greater accuracy may be obtained by using a Smith-Waterman algorithm (Smith & Waterman 1981) to align query sequences to possible genomic hits. As Smith-Waterman is a local gapped alignment algorithm, BFAST is readily able to account for both single nucleotide polymorphisms (SNPs) and indels in the NGS data.

1.3.3 BWA

Burrows-Wheeler Aligner was the first Burrows-Wheeler algorithm based tool to allow gapped alignments (Li & Durbin 2009). BWA also allows for sequence mismatches, and can accommodate alignments in both base-space and colour-space. The performance of BWA has been shown to be comparable to SOAP2 (Li et al. 2009) and Bowtie (Li & Durbin 2009; Lee et al. 2012), however run-time may be increased due to handling of gapped alignments. The advantage to handling gapped alignments is crucial for longer reads that are more likely to contain indels.

BWA has an advantage over earlier aligners in that it provides mapping quality scores for reads. This allows mapped reads to be judged on these qualities, and decisions can be made about whether or not to include specific reads in downstream analyses.

1.3.4 Maq

For mapping of large numbers (>10 million) of short reads, Maq is able to perform ungapped alignment to a reference genome accurately and efficiently (H. Li et al. 2008). Occasionally ambiguity arises in mapping locations, due to repetitive regions, short read lengths or sequencing errors; Maq handles these cases by inclusion of mapping quality scores. Maq can also take note of paired end data and Phred-scaled base quality to calculate error probabilities, and where ambiguous alignments may be present, reports only the alignment with the lowest error rate. Where multiple ambiguous reads have equally low error probabilities, one of the reads is reported randomly. In such cases, Maq sets the mapping quality to 0.

1.3.5 Bowtie

Bowtie is another aligner based on the Burrows-Wheeler algorithm, allowing fast alignment of short reads. For example, Bowtie can align 25 million reads to the genome per central processing unit (CPU) hour, and still maintain a memory footprint of <2 GB. Parallelisation allows for Bowtie to run on multiple processing cores simultaneously. Bowtie supports sequence mismatches, and updates have added support for paired end reads, and alignment of colour-space reads. Integration and parsing data between Bowtie and other software such as SAMtools (Li et al. 2009) is possible, and has influenced other alignment software such as Tophat (Trapnell et al. 2009) and Cufflinks (Trapnell et al. 2010).

1.3.6 SHRiMP

SHRiMP has been designed to map shorter sequencing reads against a reference, even if there is a large amount of mismatches or polymorphism present (Lee et al. 2012). This package has been designed to work primarily with colour-space reads from SBL platforms. SHRiMP utilises a Smith-Waterman algorithm and searches for *k*-mer matches (where *k* is a positive integer). Identification of SNPs is permitted by allowing adjacent mismatches between the reference sequence and the sequencing read (Rumble et al. 2009). Updates to SHRiMP include SHRiMP2 (David et al. 2011) which allows for analysis of broader data formats, including FASTA and FASTQ input, as well as base-space reads. The updates also account for paired end data and speed improvements are introduced via parallel computing (David et al. 2011).

1.4 Variant calling

1.4.1 Germline variant calling

Variant calling from germline samples is a standard technique for many research and clinical laboratories. As such, there is a demand for accurate variant callers. One such caller is VarScan (Koboldt et al. 2009), which has reported sensitivity rates of ~97% for SNVs. VarScan is capable of handling data from multiple NGS platforms such as the Roche/454 sequencer and Illumina sequencers. Likewise, the Genome analysis tool kit (GATK; DePristo et al. 2011; McKenna et al. 2010) was developed to utilise data from several platforms and provide a suite of tools for users to ensure reliable variant calls. The UnifiedGenotyper tool in GATK has a reported sensitivity of >99% and additionally, GATK also has tools to assess quality of the sequencing data and quality of the variant calls. This is therefore a powerful tool for bioinformaticians working with NGS data.

1.4.2 Somatic variant calling

Detection of somatic mutations is technically challenging; they are often at low frequency due to presence of normal tissue and the heterogeneous nature of tumours, and technical and biological noise makes the identification of genuine somatic events difficult. There is however, software available that is capable of detecting somatic mutations, including MuTect (Cibulskis et al. 2013), SomaticSniper (Larson et al. 2012), JointSNVMix (Roth et al. 2012) and Strelka (Saunders et al. 2012).

MuTect requires both a sequenced tumour and normal sample in order to call somatic variants; calling variants from both samples allows for distinction between variants exclusive to the tumour and those present in the matched normal sample. MuTect is capable of detecting mutations in colorectal cancer (CRC) at low allele frequencies (<1%) with high sensitivity (>95%) (Cibulskis et al. 2013). Furthermore, sensitivity increases with higher depth of coverage.

MuTect is therefore ideal for use with samples that may have contamination from normal tissue or that are of a subclonal nature.

JointSNVMix also requires a matched normal-tumour pair for reliable somatic variant calling. The sensitivity of JointSNVMix is also high (>90%), although this may be lower in samples with copy number variation (CNV) (Roth et al. 2012). The sensitivity of SomaticSniper, which also relies on a matched tumour-normal pair, has a similar sensitivity level (~92%) as revealed when tested on data from COSMIC (Larson et al. 2012; Forbes et al. 2008).

1.5 Molecular analyses of tumours

The majority of archived tumours are only available in formalin-fixed, paraffin-embedded (FFPE) blocks. FFPE is routinely used as a preservation technique as it maintains the structural integrity of physiological tissues (von Ahlfen et al. 2007); however, DNA from FFPE material poses several challenges for NGS. One of the earliest effectors of FFPE sample quality is the time spent in ischaemic anoxia during and after surgery (Hedegaard et al. 2014). Formalin-fixation should ideally take place as soon as possible after resection as anoxia may alter the transcription pattern of some genes (Srinivasan et al. 2002; Bennewith & Dedhar 2011), giving a false representation of the original cancer. Extended times to fixation from surgical clamping of blood vessels have also been shown to affect the number of mitotic cells in resected samples (Cross et al. 1990), where the number after 6 hours dropped to 49% of the number observed at 30 minutes.

Formalin-fixation is known to reduce the efficiency of DNA extraction (Serth et al. 2000). This could be because the formaldehyde in formalin denatures the hydrogen bonds between chains and causes bases to un-stack in AT-rich regions (Srinivasan et al. 2002). Formaldehyde interacts with DNA in a number of ways that could affect downstream processes: apurinic and apyrimidinic sites (collectively referred to as AP sites) are formed which leave a free pyrimidine or purine residue, respectively. This could be a problem for PCR if the AP sites lie in primer-binding regions. Formaldehyde can also react with a free amino group to form a methylol derivative of a base; these methylol derivatives are then able to react to form methylene bridges between the affected bases. The bridges formed may prevent strand denaturation during PCR (Douglas & Rogers 1998).

The temperature of samples during fixation also effects DNA quality. At room temperature, (non-FFPE) tissues have been reported to suffer degradation upon fixation in buffered formaldehyde. At 4°C however, DNA showed no degradation (Tokuda et al. 1990). Fixation in the presence of deoxyribonuclease (DNase) may also contribute to degradation (Tokuda et al. 1990). Further studies have shown that fixation in buffered formaldehyde in the presence of DNAse-neutralising ethylenediaminetetraacetic acid (EDTA) allows for high molecular weight DNA to be extracted (Yagi et al. 1996). The authors claim, however, that a mechanism other than DNAse may be responsible for most of the degradation to the DNA.

Studies from ancient DNA have shown that while it is assumed that DNA extractions from older samples are likely to give lower yield (which itself is expected to contain more DNA damage) than more recent samples, the age of the sample is not the sole determinant, and environmental factors such as temperature and salt concentration play a role (Hofreiter et al. 2001). Fixation time also affects FFPE DNA degradation. Fixation periods of 24 to 48 hours are not uncommon (Srinivasan et al. 2002), although the longer time periods have a more adverse effect on DNA quality. Storage temperature of FFPE tissues after formalin fixation may also determine DNA quality.

Additional steps are required during DNA extraction to remove the paraffin, for example with xylene and 100% ethanol (Kokkat et al. 2013), which increases processing time and costs. There are several commercially available kits capable of extracting DNA from FFPE samples. One study (Okello et al. 2010) compared 9 commercially available nucleic acid extraction kits and a phenol-chloroform based in-house method using 7 FFPE blocks. To avoid sample biases, the authors used the same set of 7 samples for each kit tested. All but two of the commercial protocols used xylene and 100% ethanol to remove and wash away paraffin; the remaining protocols used Q-Solution and d-Limonene. The kits showed varying yields, ranging between 0.02 – 5.47 ng/µl. Three of the commercial kits and their in-house protocol gave consistently high yield, which proved to be readily amplifiable. There was no significant correlation between amplification suitability and sample age (range = 5 to 15 years since embedment in paraffin) or incubation time during the extraction protocol. There may, however, be a statistically significant link between a protocol's incubation time and final yield of DNA (Okello et al. 2010).

1.5.1 DNA from normal tissue preserved in FFPE is useful in genotyping studies

Tumour FFPE (tFFPE) DNA is often the focus of genetic studies in cancer, while the use of normal tissue that is also preserved in FFPE (nFFPE) is less commonly reported. Typically genome-wide genotyping of FFPE DNA is poor, so it is understandable that nFFPE is often overlooked for whole blood samples. Whole blood samples offer greater DNA quality in comparison, but where no blood sample is available, nFFPE can be used in lieu. Concordance has been shown to be high (>99%) between blood and nFFPE samples (Cannon-Albright et al. 2011). Where blood and nFFPE samples are available, comparisons between data from both may lead to identification of FFPE-induced artefacts.

1.5.2 FFPE-induced artefacts

It is a common feature of FFPE samples that they harbour formalin-induced artefacts, primarily C:G>T:A substitutions. These C:G>T:A artefacts likely arise as a result of deamination of cytosine (Do & Dobrovic 2012). Cytosine is deaminated to uracil (U) in ancient DNA (Hofreiter, Serre, et al. 2001; Do & Dobrovic 2012; Hofreiter, Jaenicke, et al. 2001); during PCR an adenine will be incorporated opposite uracil as opposed to a guanine that would normally have been incorporated opposite the cytosine. During the next PCR cycle, a thymine will be incorporated opposite the adenine, leading to a C:G>T:A transition in the

PCR product. The frequency of these artefacts has been reported at around 1 in 500 bases in FFPE DNA (Williams et al. 1999).

1.5.3 Suitability of FFPE tissues in NGS.

Fixation times of 24 – 72 hours were used in one study (Schweiger et al. 2009) and stored at -20°C for 6 months before sequencing using a SBS approach. The authors also prepared snap frozen samples from the same patient and subjected them to the same storage and sequencing conditions; comparisons between the two tissue types revealed similar read numbers, read alignments and chromosome-wise coverage. In fact, older samples (both snap frozen and FFPE) were used in the study that dated back up to 18 years and still successfully sequenced using the same methods. Snap frozen and FFPE DNA samples again revealed similar coverage distribution.

Further work has shown that FFPE DNA is suitable for use in targeted re-sequencing experiments (Kerick et al. 2011). This study showed that 0.5 – 3.0 µg of genomic DNA from FFPE can be used in sequencing of 52 Mb (the whole exome) and 3.9 Mb (custom targets), using a Genome Analyzer IIx (Illumina). The matched snap frozen samples were also sequenced and compared against the FFPE samples. For both tissue types, an average of 75% of the mapped reads were on-target for the exome sequencing, and 99% of the exome was covered by at least one sequencing read. For the custom target panel, 80% of the sequencing reads mapped to the target region, and 98% of the region was covered by at least one read. These results are testament to the suitability for inclusion of FFPE samples in NGS experiments.

1.5.4 Highly accurate variant calls are possible from FFPE DNA

FFPE samples have been used for whole exome sequencing and shown a 98% accuracy level when compared to Affymetrix SNP array data, even at low (10x) sequencing depths (Kerick et al. 2011). The same study compared the FFPE samples against their matched snap frozen counterparts, and found just 179 loci (1.2% of the total tested) were discordant at 20x depth. 149 of these discrepancies were found in FFPE but not snap frozen samples, suggesting they may be false positive artefacts. Indeed, the majority were either C:G>T:A or A:T>G:C substitutions (Kerick et al. 2011; Do & Dobrovic 2012). At 40x coverage, the number of discordant calls between snap frozen and FFPE samples fell to just 12, and decreased to 0 at 80x coverage (Kerick et al. 2011).

1.6 Colorectal cancer

CRC is the third most common cancer diagnosed globally, and is the fourth largest cancer in terms of deaths (Haggar & Boushey 2009). Every year, around 1.4 million people are diagnosed with CRC, and approximately 700,000 die from the disease (Torre et al. 2015). Although the incidence of the disease is low for younger persons, it rises with age and the median age of diagnosis is around 70 years (Brenner et al. 2014). It appears to be more common in North America, Europe and Australia than in Asia and Africa, suggesting it may be a disease of Western lifestyles. However, in Western countries CRC incidence appears to be in decline; this is likely due to increasing preventative measures such as colonoscopy (Amaro et al. 2016; Torre et al. 2015).

CRCs progress through a number of characterised stages (Dukes' stages). Dukes' stages can be described as follows; stage A involves CRCs limited to the bowel wall; stage B sees carcinomas spreading locally beyond the bowel wall but with no nodal involvement; stage C involves spreading beyond the bowel wall and has lymph node involvement; stage D sees the carcinoma metastasising to other organs (Turnbull et al. 1967)**.** Prognosis depends upon the stage of CRC at diagnosis; patients diagnosed at Dukes' stage A have a 90% chance of surviving for five years, patients with Dukes' stage B have a 75-80% chance (Aslam et al. 2015) whereas those diagnoses at Dukes' stage D have only a 5% chance of surviving five years (de la Chapelle 2004).

1.6.1 The development of CRC

Sporadic CRC develops, in general, through one of two pathways: the microsatellite instability (MSI) and chromosomal instability (CIN) pathways.

CRCs with MSI (~15% of cases, Boland & Goel 2010; Webber et al. 2015) are less likely to be associated with *KRAS* or *TP53* mutations. Instead, mismatch repair (MMR) genes (e.g. *MLH1*, *MSH2* and *MSH6*) are likely to be epigenetically silenced (Horvat & Stabuc 2011). Tumours with MSI contain insertion or deletion mutations in dinucleotide repeats (such as CA) and are likely to be located within the proximal colon. MSI can be used as a prognostic marker in CRC (Webber et al. 2015), as treatments such as 5-fluorouracil (5-FU) are not effective (Warusavitarne et al. 2006); the DNA damage induced by 5-FU which normally encourages apoptosis is not recognised by the cell's faulty MMR machinery (Choudhary et al. 2012; Thibodeau et al. 1993).

The CIN pathway is characterised by chromosomal aneuploidy, centromere dysfunction, inactivating mutations in tumour suppressor genes and activating mutations in oncogenes (Pihan 2013; Fearon & Vogelstein 1990). Around 80 - 85% of CRC cases have CIN (Cisyk et al. 2015; Kołos et al. 2015). Mutations in the tumour suppressor gene *adenomatous polyposis coli* (*APC*) are common (Kołos et al. 2015) in sporadic tumours (Figure 1.7). Powell et al. (1992) found that *APC* was mutated in 63% of small benign adenomas, and in 60% of malignant tumours, and others have found *APC* is mutated at similar frequencies (Dow et al. 2015; Samowitz et al. 2007). These data suggest that *APC* mutations occur very early during CRC development. *APC* is known to be hypermethylated in CRC cases (Ding et al. 2015), leading to silencing of the gene and its tumour-supressor activity. Inactivating mutations within tumour suppressor genes are consistent with Knudson's two hit hypothesis (Pannett & Thakker 2001; Knudson 1971), in which biallelic mutations are required for cellular progression to tumourigenesis. Sporadic tumours often have two APC-inactivating mutations, the first of which can be inherited. Oncogenes, such as *KRAS*, *NRAS* and *HRAS*, are (cumulatively) mutated in between 16% and 40% of all CRCs (Van Krieken et al. 2016; Waring et al. 2015) and are found in a similar amount of adenomas over 1 centimetre (cm) in diameter (Bos et al. 1987; Vogelstein et al. 1988). Such mutations are rarer in smaller adenomas, suggesting these mutations may drive growth of larger adenomas. The CIN pathway is often linked with loss of chromosome 17p, which occurs in over 75% of CRCs (Lee et al. 2013), in varying degrees in flat adenomas (Postma et al. 2005), in around 30% of late adenomas and in 10% or less of early adenomas

(Delattre et al. 1989; Vogelstein et al. 1988; Fearon & Vogelstein 1990). This genomic region contains the tumour suppressor *TP53*, and the remaining copy of the gene has been shown to carry inactivating mutations (Olivier et al. 2010; Nigro et al. 1989). These observations indicate that inactivating *TP53* therefore allows for progression from adenomas to carcinomas (Figure 1.7).

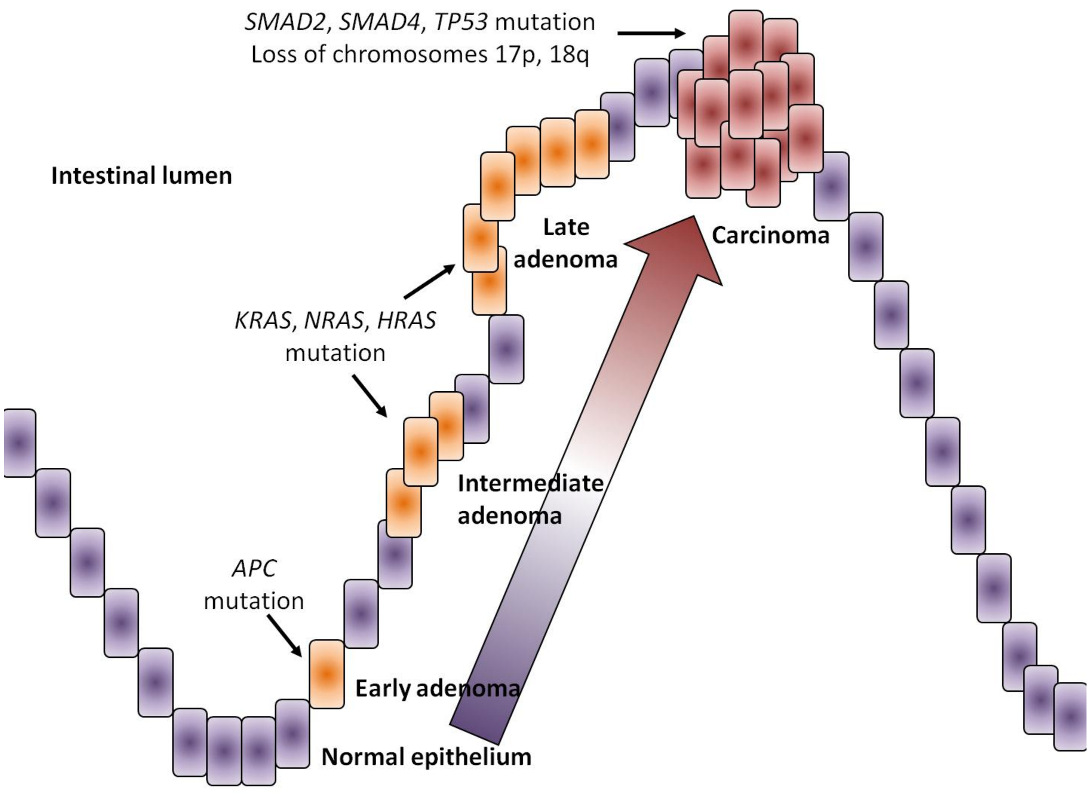


Figure 1.7. Colorectal carcinoma progression from regular epithelium. Progression follows the arrow from the base of the crypt. Mutations in genes such as *APC* allow for benign adenomas or polyps to form. Further mutations, such as those in *EGFR*, *KRAS* and related Ras genes allow for progression of late stage adenomas. Finally, mutations in genes such as *SMAD4*, *PIK3CA* and *TP53*, along with increased chromosomal instability allow for progression of adenomas to malignant cancers (Fodde et al. 2001; Fearon & Vogelstein 1990; Pino & Chung 2010).

1.6.2 Germline mutations that predispose to CRC

Up to ~15% of CRCs are thought to be caused by germline factors, but only a proportion of these are known. Germline mutations in *APC* cause familial adenomatous polyposis (FAP), and patients present with hundreds or thousands of small colorectal adenomas. These adenomas are usually detectable by the patient’s thirties, and if left untreated, some of these will progress to CRC (Phelps et al. 2009; Fearnhead et al. 2001).

Hereditary non-polyposis colorectal cancer (HNPCC) or Lynch syndrome is caused by germline mutations in MMR genes such as *MLH1*, *PMS2*, *MSH2* and *MSH6*; these mutations show a penetrance of around 80% (de la Chapelle 2004). Around 5% of CRC cases are attributed to Lynch syndrome and are usually diagnosed based on age of onset and whether there is a family history. Tumours from patients with HNPCC display MSI.

MutYH-associated-polyposis (MAP) is associated with a colorectal phenotype that is similar to mild forms of FAP. MAP is caused by mutations in the base excision repair gene *MutYH* (Sampson et al. 2005). MutYH is a DNA glycosylase responsible for excision of adenines that are accidentally incorporated opposite 7,8-dihydro-8-oxoguanine (8-oxoG), which is created during oxidative damage (Nakabeppu 2014; Slupska et al. 1999).

1.6.3 Treatment of CRC - Surgery and chemotherapy

One of the front line defences against CRC is surgery. However, even after surgical resection, there is still up to a 50% chance that the patient could relapse. For the majority of these relapsed patients, further surgery is ineffective and increases risk of death (Kelly & Cassidy 2007; Obrand & Gordon 1997). Initial chemotherapies introduced for CRC patients included 5- FU. It has proven to be a success in combination with other treatments, in that it improves overall survival (OS) and progression free survival (PFS) in patients following resection of Dukes' stage C CRCs. However, as a first line treatment it is not as effective as typically less than 15% of CRCs show beneficial response (Johnston & Kaye 2001). Using 5-FU in combination with treatments such as oxaliplatin or irinotecan increase the response rate to around 50% in advanced CRC (Giacchetti et al. 2000; Douillard et al. 2000).

The mechanism of action for 5-FU is based on its structural similarity to uracil; the difference is that 5-FU has a fluorine atom instead of a hydrogen atom at C-5. Inside the cell, 5-FU is converted to active metabolites including fluorodeoxyuridine monophosphate (FdUMP). These active metabolites are able to disrupt ribonucleic acid (RNA) synthesis and the action of thymidylate synthase (Longley et al. 2003) as they compete for the active site with the normal substrate, deoxyuridine monophosphate (dUMP). Inhibiting thymidylate synthase blocks deoxythymidine monophosphate synthesis, subsequently leading to depletion of deoxythymidine triphosphate and imbalances in the levels of other deoxynucleotides. These imbalances cause lethal DNA damage by disrupting DNA repair (Longley et al. 2003).

Capecitabine is a precursor of 5-FU. The final step during the conversion to 5-FU relies on thymidine phosphorylase, which can be found at higher levels in tumour cells than normal cells (Kelly & Cassidy 2007). This ensures tumour cells effectively receive larger doses of the end product than normal cells. Clinical trials of capecitabine have shown that there is a greater tumour response with capecitabine than with 5-FU (Hoff et al. 2001; Van Cutsen et al. 2001).

The topoisomerase I inhibitor irinotecan prevents topoisomerase from unwinding the DNA double helix during replication, leading to strand breakage and ultimately cell death (Kelly & Cassidy 2007). In a study of patients whose CRC progressed after 5-FU treatment, Rothenberg et al. (1999) found that giving intra-venous irinotecan improved OS, but patients also reported gastrointestinal side effects such as diarrhoea and nausea.

1.6.4 EGFR signalling in colorectal cancer

The epidermal growth factor receptor (EGFR) pathway (Figure 1.8) is responsible for cell migration, adhesion and proliferation (Hynes & Lane 2005; Oda et al. 2005; Tian et al. 2013). Expression of EGFR (also known as ERBB1) can be seen in many carcinomas, including colon, lung, breast, kidney, head and neck and pancreas (Krasinskas 2011). Carcinogenesis, survival and response are affected by EGFR expression, gene amplification and mutations. The extracellular domain of EGFR is responsible for ligand binding. EGFR also contains a trans-membrane domain and a cytoplasmic region that contains a tyrosine kinase domain. EGFR is one of four closely related members of the ERBB family (ERBB1-4) that each contain the same structural domains (Tebbutt et al. 2013; Hynes & Lane 2005). Each member of the family is able to form a hetero- or homo-dimer upon ligand binding (Yarden & Sliwkowski 2001).

EGFR signalling, when not regulated, can lead to transformation of benign to malignant tissue and increased cell proliferation, angiogenesis, metastasis and cell survival, all of which contribute to tumour progression (Krasinskas 2011; Mitsudomi & Yatabe 2010; Spano et al. 2005). Over expression of EGFR has been reported in CRCs, with wide ranges in the number of cases showing increased levels (Spano et al. 2005; Goldstein & Armin 2001; McKay et al. 2002; Resnick et al. 2004). Some studies have reported that EGFR over expression is associated with reduced survival (Goldstein & Armin 2001; Resnick et al. 2004); however, this has not be found by others (Spano et al. 2005).

1.6.5 Treatments targeting EGFR and predictive biomarkers

Activation of the EGFR pathway, either by natural ligand binding or by molecular alterations, signals the cell to proliferate. Constitutive activation is problematic and is likely to negatively affect response to anti-EGFR monoclonal antibody (mAB) treatments such as cetuximab. EGFR itself has multiple mutations that affect response to cetuximab (so called ‘predictive biomarkers’). One such mutation is p.T790M (Pao et al. 2005; Kwak et al. 2005; Kobayashi et al. 2005), often seen in lung cancer. p.T790M is capable of activating EGFR by increasing its affinity for ATP, the proposed main mechanism by which drug resistance is conferred (Yun et al. 2008). In addition, the acquired mutation p.S492R has been shown to prevent binding of cetuximab, leading to cetuximab resistance (Montagut et al. 2012). However, EGFR mutations are, in general, rare in CRC; in an analysis of 156 metastatic CRC (mCRC) patients treated with cetuximab, 1.9% (3/156) of tumours carried p.S492R, which was only detectable after treatment (Esposito et al. 2013).

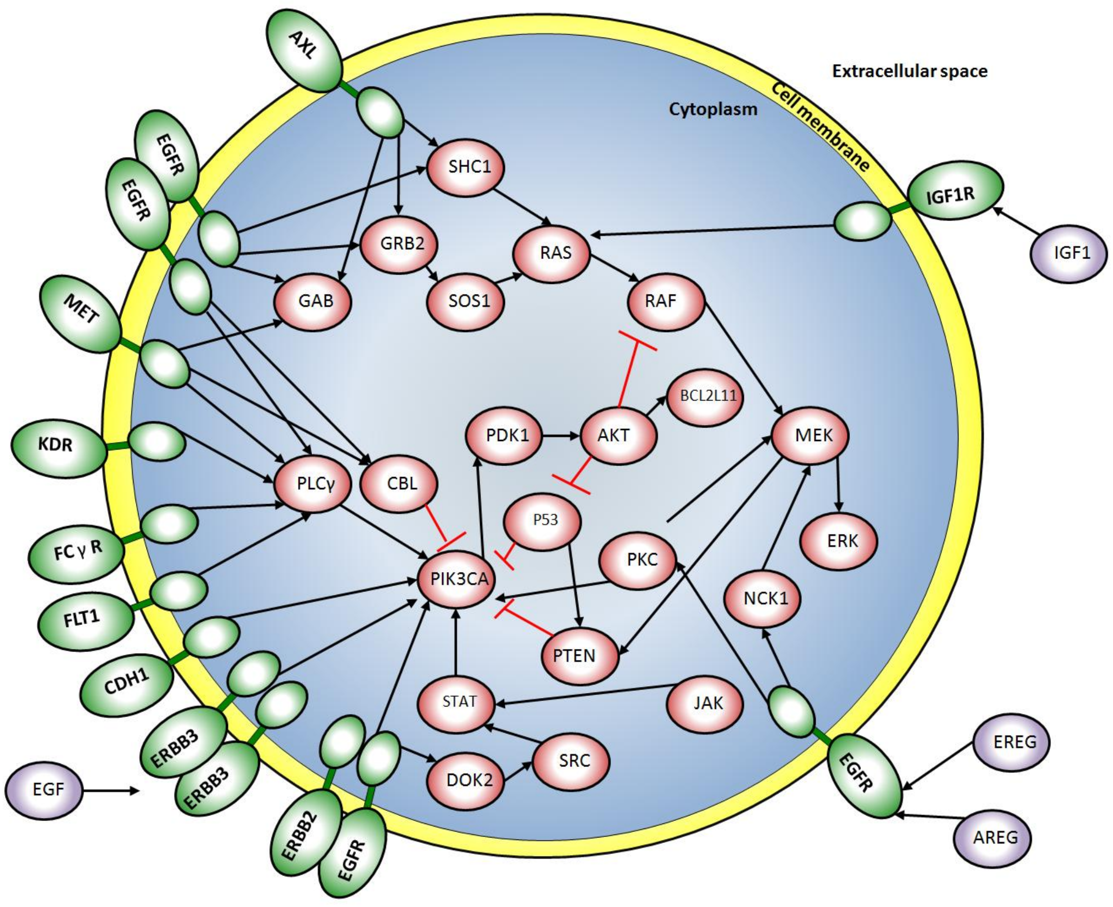


Figure 1.8. EGFR pathway and related signalling. Ligands are shaded purple, receptors are shaded green and intracellular molecules are shaded red. Black arrows convey signal transduction while red lines indicate inhibition of signalling (Li et al. 2013; Parsons et al. 2005; Avraham & Yarden 2011; Organ & Tsao 2011; Aloy et al. 2006; Kanno et al. 2000; Griffioen & Molema 2000; Oda et al. 2005; Holohan et al. 2013; Fruman & Rommel 2014; Luo et al. 2012; Virella &

Lopes-Virella 2012; Schmidt et al. 2014; Choi et al. 2013; Fernandez & Torres-Alemán 2012).

The most well known predictor of clinical response to cetuximab is *KRAS* mutation status. This association was first reported by Lièvre et al. (2006), with *KRAS* mutation leading to lack of cetuximab response (P = 0.0003). Pooling of data from two studies (Lièvre et al. 2006; Moroni et al. 2005), showed a 91% probability of non-response in the presence of a *KRAS* mutation (with a 10-fold higher relative of risk of cetuximab response from *KRAS* wild-type compared to *KRAS* mutant tumours).

There are several common somatic mutations in *KRAS*, and known mutation hotspots include codons 12, 13, 61 and 146 (Wicki et al. 2010). Codons 12 and 13 mutations account for 90% of *KRAS* mutations (Bazan et al. 2005). It has been suggested that some *KRAS* mutations have different predictive effects for cetuximab. For example, patients with a p.G13D mutation exhibited longer OS and PFS than those with other mutations (De Roock et al. 2010); however, these data require independent validation. Patients with CRCs harbouring mutations at codon 61 in *KRAS* may not benefit from cetuximab, as these patients have significantly lower response rates than those with wild type *KRAS* (De Roock, Claes, et al. 2010). There is conflicting evidence to suggest that codon 146 mutations may significantly lower response to cetuximab in CRCs (De Roock, Claes, et al. 2010; Loupakis et al. 2009).

NRAS is mutated in around 2-4% of CRCs (Meriggi et al. 2014; De Roock, Claes, et al. 2010), and similarly to *KRAS,* mutations in codons 12, 13 are the most common. In *KRAS* wild type tumours, mutations in *NRAS* may cause resistance to anti-EGFR therapies (Di Bartolomeo et al. 2014). *BRAF* mutation (in particular p.V600E) has been shown to associate with lack of clinical response to cetuximab, as well as reduced OS and PFS (Di Nicolantonio et al. 2008). However, other studies have not supported this (Moroni et al. 2005).

PIK3CA encodes the p110α subunit of PI3K, the signalling of which is inhibited by PTEN. Loss of PTEN (seen in ~30% of sporadic cases) therefore allows continuous PI3K signalling and cetuximab-resistance (Frattini et al. 2007). Common mutations in *PIK3CA* occur in exons 9 and 20 (Jhawer et al. 2008; Prenen et al. 2009; Sartore-Bianchi et al. 2009). Cetuximab-resistance can be conferred by *PIK3CA* mutations mutually exclusive of *KRAS* mutations, and patients harbouring somatic *PIK3CA* mutations have worse PFS than those with wild-type *PIK3CA* (Sartore-Bianchi et al. 2009). OS decreases in patients with at least one *PIK3CA* mutation, and appears to be statistically significant when combined with loss of *PTEN* (Sartore-Bianchi et al. 2009).

1.7 Background to this Project and Aims

Patients with colorectal cancer are often treated with the anti-EGFR drug cetuximab. However, many patients show lack of response to cetuximab even though they do not exhibit mutations in genes associated with response. Using samples from Medical Research Council (MRC) COIN trial (Maughan et al.

2011), this project set out to identify potential novel genetic predictors of response to cetuximab.

The aims of this project were to:

(1) Test a bioinformatics pipeline and parameters for germline variant calling from NGS data (Chapter 3).

(2) Develop and optimise protocols for variant calling from NGS of FFPE-DNA (Chapter 4).

(3) Identify novel genetic markers of non-response to cetuximab (Chapter 5).

Chapter 2 - Materials and Methods

2.1 Reagents and equipment.

Table 2.1. Suppliers and reagents used during the course of this PhD.

Supplier (location)	**Reagent**
4titude (Surrey, UK)	96-well non skirted PCR plate
	96-well skirted PCR plate
	PCR adhesive plate seal
Agilent (California, USA)	SureSelect All Exon 50 Mb enrichment kit
	SureSelect All Exon V2 enrichment kit
BDH Laboratory Supplies (Poole, UK)	Gill's haematoxylin
Beckman Coulter Ltd (High Wycombe, UK)	AMPure XP beads
Bio-Rad (California, USA)	Microseal 'A' adhesive film
Biotium (California, USA)	GelRed Nucleic acid gel stain
Illumina (California, USA)	Assay Control DNA 1 (ACD1)
	Assay control oligonucleotide pool (ACP1)
	Adapter collar
	Custom amplicon oligonucleotides (CAT)
	Elution buffer with Tris (EBT)
	Extension-ligation mix 4 (ELM4)
	Filter plate & lid
	Genome Analyser IIx
	HiSeq 2000
	Hybridisation buffer 1 (HT1)
	i5 primers
	i7 primers

	Incorporation buffer
	Library normalisation additives 1 (LNA1)
	Library normalisation beads 1 (LNB1)
	Library normalisation storage buffer 2 (LNB2)
	Library normalisation wash 1 (LNW1)
	MiSeq reagent cartridge v2 (300/500 cycle)
	Oligonucleotide hybridisation for sequencing 2 (OHS2)
	PE MiSeq flow cell
	PCR master mix 2 (PMM2)
	Stringent wash 1 (SW1)
	TruSeq DNA polymerase 1 (TDP1)
	Universal buffer 1 (UB1)
Mettler Toledo (Ohio, USA)	GPS-L10 (0.1 - 20 µl) pipette tips
	GPS-L1000 (20 - 1000 µl) pipette tips
	PR-1000 (100 - 1000 µl) pipette
	PR-2 (0.1 - 2 µl) pipette
	PR-20 (0.5 - 20 µl) pipette
	PR-200 (20 - 200 µl) pipette
	SS-L250 (2 - 20 µl) pipette tips
Motic (Hong Kong, China)	B3 Professional Series Digital Microscope
New England Biolabs (Massachusetts, USA)	Gel loading dye, orange (6x)
	uracil DNA glycosylase (UDG)
QIAGEN (Manchester, UK)	2 ml collection tubes
	Buffer AL
	Buffer ATE
	Buffer ATL

	Buffer AW1
	Buffer AW2
	MinElute column
	Proteinase K
Roche Nimblegen (Wisconsin, USA)	SeqCap EZ Exome capture kit V2
Sigma-Aldrich (Missouri, USA)	1.5 ml microcentrifuge tubes
	DPX mountant
	Eosin Y solution
	Ethidium bromide (10 mg/ml diluted to 0.5 µg/ml)
	Ethylenediaminetetraacetic acid
Thermo Fisher Scientific (Massachusetts, USA)	96-well 0.8 ml MIDI plate
	Adhesive aluminium foil seal
	Agarose
	Dimethyl sulfoxide
	Ethanol
	Mineral oil
	NaOH
	Superfrost glass microscope slides
	Swann-Morton scalpel blades
	Tris base
	Xylene
VWR International (Pennsylvania, USA)	Acetic acid
	Cover glasses

2.2 Patients and samples

Patients used in this project had advanced or metastatic colorectal cancer and were enrolled within the MRC trials COIN (ISRCTN27286448) or COIN-B (ISRCTN38375681). Patients within COIN were randomised 1:1:1 into 3 arms; Arm A patients received continuous oxaliplatin and fluoropyrimidine chemotherapy, Arm B patients received continuous chemotherapy plus cetuximab, and Arm C patients were given chemotherapy intermittently. Patients in the COIN-B trial were given intermittent chemotherapy and randomised 1:1 into 2 arms; Arm D patients were also given intermittent cetuximab, whereas Arm E patients were also given cetuximab continuously. Informed consent was obtained from all patients for their samples to be used in bowel cancer research, as approved by REC [04/MRE06/60].

2.3 Identification of normal and tumour tissue within FFPE sections

2.3.1 FFPE block sectioning

Ten 5 µm sections of each normal and each tumour FFPE block were cut using a microtome and clean cutting blade. Each section was flattened by floating in warm water and then fixed to a microscope slide by air drying overnight.

2.3.2 Haematoxylin and eosin staining of FFPE sections

One slide per FFPE block was selected for staining. Slides were submerged in xylene for 5 minutes, and then in fresh xylene for a further 5 minutes. Slides were then dipped in 100% ethanol for 5 minutes, and repeated using fresh 100% ethanol. Slides were then submerged in 70% ethanol for 5 minutes, followed by 50% ethanol for a further 5 minutes. Slides were bathed in Milli-Q treated H_2O for 2 minutes, and for a further 2 minutes in fresh Milli-Q treated H_2O. The slides were then immersed in Gill's haematoxylin for 30 seconds, and rinsed using H_2O for 3 minutes. The slides were dipped in eosin Y solution for 10 seconds, and rinsed with H_2O for 3 minutes. The stained slides were submerged in 50% ethanol for 1 minute, followed by 70% ethanol for 1 minute, 100% ethanol for 1 minute, and fresh 100% ethanol for 1 minute. Finally, slides were dipped in xylene for 1 minute, followed by fresh xylene for 1 minute. A cover slip was fixed to the slide using DPX mountant and allowed to air dry.

2.3.3 Identification of tumour and normal tissue from FFPE sections

Using a Motic B3 series digital microscope, tumour tissue in stained FFPE sections was identified by Dr. Christopher Smith, and was highlighted with a marker pen. Tumour tissue in each unstained slide was identified by overlaying with the stained section. Normal FFPE (nFFPE) tissue was confined using a digital microscope.

2.3.4 Macrodissection of tissue from FFPE sections

Tumour tissue was macrodissected using a scalpel blade and collected in a 1.5 ml microcentrifuge tube (1 per FFPE block). Normal tissue was macrodissected into clean 1.5 ml microcentrifuge tubes.

2.4 DNA extraction from FFPE sections.

A QIAamp® DNA FFPE tissue kit (QIAGEN) was used for extraction of DNA from FFPE tissues. 1 ml xylene was added to each microcentrifuge tube containing FFPE tissue and vortexed for 10 seconds. The samples were centrifuged at 13,000 RPM for 2 minutes and the supernatant discarded. 1 ml 100% ethanol was added to each tube and vortexed. Samples were centrifuged at 13,000 RPM for 2 minutes and the supernatant was discarded.

The samples were left at room temperature for 10 minutes or until the residual ethanol had evaporated. The pellet in each tube was re-suspended in 180 µl buffer ATL, and 20 µl proteinase K was added. Samples were vortexed and incubated at 56°C for 1 hour. Samples were then incubated at 90°C for a further hour.

The samples were centrifuged briefly and 200µl buffer AL was added, followed by vortexing. 200µl 100% ethanol was added to each sample and thoroughly vortexed. Each sample was transferred to a QIAamp MinElute column, placed inside 2 ml collection tubes, and centrifuged at 6,000 x g for 1 minute. The supernatant was discarded and the MinElute columns were placed inside clean 2 ml collection tubes. 500 µl buffer AW1 was added to the MinElute column, which was centrifuged at 6,000 x g for 1 minute.

The supernatant was discarded and the MinElute columns placed inside clean 2 ml collection tubes. 500 µl buffer AW2 was added to the columns and centrifuged at 6,000 x g for 1 minute. The columns were placed inside clean 2 ml collection tubes and centrifuged at 20,000 x g for 3 minutes. The MinElute columns were placed inside clean 1.5 ml microcentrifuge tubes, 100 µl buffer ATE was added and incubated at room temperature for 1 minute. Samples were centrifuged at 20,000 x g for 1 minute and DNA was stored at -20°C.

2.5 PCR and Sanger sequencing

PCR was performed in 25-50 µl reaction volumes containing AmpliTaq Gold, 10-20 ng DNA and 10µM of primers For Sanger sequencing, single stranded DNA and excess dNTPs were removed using ExoSAP and reactions were cycle-sequenced using Big Dye Mix, purified using Montage sequencing clean-up plates and analysed on an ABI Prism 3100.

2.6 Mutation analysis of *KRAS*, *BRAF*, *NRAS* and *PIK3CA* hotspots

The mutation status at hotspots found in CRCs (deposited in COSMIC (Forbes et al. 2008)) was determined in all tumour samples analysed in this project. These hotspots included codons 12, 13 and 61 of *KRAS*, codons 12 and 61 of *NRAS*, codons 594 and 600 of *BRAF*, and codons 542, 545, 546 and 1047 of *PIK3CA*. Mutation status was determined either by pyrosequencing or Sequenom assays as previously described (Maughan et al. 2011; Smith, Fisher, et al. 2013).

2.7 UDG treatment

A minimum of 250 ng of tFFPE derived DNA was treated with 10U of uracil-DNA glycosylase (UDG) and 1x UDG buffer at 37°C. UDG was heat inactivated (95°C for 10 minutes) and after cooling to room temperature, products were used in library preparation.

2.8 NGS protocols

2.8.1 Exome sequencing

Germline DNA samples for patients 1-40 were shipped to Beijing Genomics Institute (BGI) (Shenzhen, China) where whole exome sequencing was performed. 30/40 (75%) exomes were captured using a SureSelect All Exon V2 enrichment kit (Agilent) and sequenced using a HiSeq 2000 (Illumina). These 30 exomes were sequenced to a depth of ≥30x. A further 10/40 (25%) of the exomes were captured using a SureSelect All Exon 50 Mb enrichment kit (Agilent) and sequenced to ≥50x depth on a HiSeq 2000. Germline samples for patients 41 - 50 were shipped to the University of North Carolina (USA) and their exomes captured using a SeqCap EZ Exome capture kit V2 (Roche Nimblegen) and sequenced on an Illumina Genome Analyzer IIx (Illumina) to a depth of ~30x.

2.8.2 Targeted sequencing

2.8.2.1 Targeted sequencing panel design

We designed a TruSeq custom amplicon (TSCA) kit (Illumina) that targeted four genes: *KRAS*, *BRAF*, *NRAS* and *PIK3CA* (panel 1). Amplicons targeting the coding exons of the 4 genes were designed using Illumina's DesignStudio software (http://bioinformatics.illumina.com/informatics/experimental-design/designstudio.html). A second panel, targeting the coding regions of 49 genes (listed in Table 2.2) was also designed. All amplicons were designed to average 175 bp in length.

2.8.2.2 DNA library preparation

The TSCA library preparation guide ('revision B' or later) was followed to prepare a DNA library for use in NGS. Firstly, 5µl of the positive control sample ACD1 (Illumina), and 5µl dH_2O were added to an empty well in a 96-well PCR plate (the HYP plate). A minimum of 250 ng of DNA from nFFPE and tFFPE, and 50 ng of gDNA was used for library preparation according to the manufacturer's specifications (Illumina).

5 µl of CAT was added to each well containing sample DNA, and 5 µl ACP1 was added to the well containing ACD1. 35 µl OSH2 was added to each sample and mixed. The plate was sealed with adhesive aluminium foil and centrifuged at 1,000 x g for 1 minute. The plate was incubated at 95°C for 1 minute in a pre-heated block. With the plate still on the block, the temperature was set to 40°C and the plate incubated for a further 80 minutes.

The filter plate unit (FPU) was assembled by placing the adapter collar over a clean 96-well 0.8 ml MIDI plate and then placing a filter plate over the collar. 45 µl of SW1 was added to each well in the filter plate and the lid was placed over the FPU, followed by centrifugation at 2,400 x g for 10 minutes. After incubation, the HYP plate was centrifuged at 1,000 x g for 1 minute.

The contents of each well in the HYP plate were transferred to the corresponding wells in the filter plate. The lid was placed on the filter plate and centrifuged at 2,400 x g for 2 minutes. 45 µl of SW1 was added to each sample and centrifuged at 2,400 x g for 2 minutes, and the flow through discarded. 45 µl of UB1 was added to each sample in the filter plate. The lid was placed back on the filter plate, which was then centrifuged at

2,400 x g for 2 minutes. 45 µl ELM3 was added to each well, the plate was sealed using an aluminium foil seal and incubated at 37°C for 45 minutes. After incubation, the aluminium seal was removed and the filter plate was centrifuged at 2,400 x g.

25 µl of 50 mM NaOH was added to each well and pipette mixed 5 - 6 times. The filter plate was incubated at room temperature for 5 minutes, during which time a PCR master mix was made using 2.8 ml PMM2 and 56 µl TDP1 for every 96 samples. 4 µl of Illumina's i5 Index primers were added to each column of a new indexed amplification (IAP) plate and 4 µl of i7 primers were added to each row of the IAP plate, so that each well contained a unique i5/i7 index combination. 22 µl of the master mix was added to each well of the IAP plate. 20 µl of each sample eluted from the filter plate were transferred to the IAP plate and pipette mixed with the PCR master mix. The IAP plate was sealed using microseal 'A' and centrifuged at 1,000 x g for 1 minute.

PCR was started at 95°C for 3 minutes, followed by either 24 cycles (germline samples) or 29 cycles (FFPE samples) of 95°C for 30 seconds, 66°C for 30 seconds and 72°C for 1 minute, after which a final extension step of 72°C for 5 minutes was performed. 5 µl of PCR product was visualised on a 4% agarose gel.

The IAP plate was centrifuged at 1,000 x g for 1 minute, and 60 µl of AMPure XP beads were added to wells in a new MIDI plate. The entire contents of the wells from the IAP plate were transferred to the new plate, which was then sealed and shaken at 1,800 RPM for 2 minutes.

Table 2.2. Targeted gene panels for use in NGS.

Panel	**Targeted genes**
1	*KRAS*, *NRAS*, *BRAF*, *PIK3CA*
2	*AKT1, AKT2, AKT3, ARAF, AREG, AXL, BCL2L11, BRAF, CBL, CDH1, DOK2, EGF, EGFR, ERBB2, ERBB3, EREG, FCGR2A, FCGR3A, FLT1, GAB1, GRB2, HRAS, IGF1, IGF1R, JAK1, JAK2, KDR, KRAS, MAP2K1, MAP2K2, MAPK1, MAPK3, MET, NCK1, NRAS, PDK1, PIK3CA, PLCG1, PLCG2, PRKCA, PRKCD, PTEN, RAF1, SHC1, SOS1, SRC, STAT1, STAT3, TP53*

The plate was stored without shaking at room temperature for 10 minutes, after which it was placed on a magnetic stand until the supernatant had cleared. With the plate still on the stand, the supernatant was removed using a pipette. Keeping the plate on the magnetic stand, an ethanol wash was performed by adding 200 µl 80% fresh ethanol to each well and the supernatant allowed to clear. The supernatant was removed and the ethanol wash step repeated. The supernatant was removed and excess ethanol removed via a pipette. The plate was removed from the stand and air dried for 10 minutes.

30 µl EBT was added to each well. The plate was sealed and shaken again at 1,800 RPM for 2 minutes. A two minute incubation at room temperature was performed without shaking, and the plate was placed back on the magnetic stand. Once the supernatant had cleared, 20 µl from each well was transferred to a library normalisation MIDI plate (LNP), which was centrifuged at 1,000 x g for 1 minute.

800 µl of LNB1 was re-suspended by pipette mixing and added to 4.4 ml LNA1. 45 µl of the LNA1/LNB1 mix was added to each well in the LNP. The plate was sealed and shaken at 1,800 RPM for 30 minutes, after which it was placed on a magnetic stand until the supernatant cleared. Once clear, the supernatant was removed via pipette and the plate removed from the magnetic stand. The remaining beads in each of the wells were washed as follows; 45 µl LNW1 was added to each well, the plate was sealed and shaken at 1,800 RPM for 5 minutes. The plate was placed on the magnetic stand for 2 minutes and the supernatant was removed. This was step was then repeated a second time.

The plate was removed from the magnetic stand and 30 µl of freshly prepared 0.1 N NaOH was added to each well. The plate was sealed and shaken at 1,800 RPM for 5 minutes. The LNP was placed back on the magnetic stand until the supernatant was clear. 30 µl of LNS2 was added to the wells of a new 96-well PCR plate, followed by transferral of the contents of the LNP to the new plate. The new plate was sealed and stored at -20°C for later use or was used immediately for library pooling (Chapter 2, Section 2.8.2.3).

2.8.2.3 Library pooling

The final plate containing the prepared library (Chapter 2, Section 2.8.2.2) was centrifuged at 1,000 x g for 1 minute and if the plate was frozen prior to use, the contents of each well were resuspended by pipette mixing. 5 µl of DNA from each well to be sequenced was pooled into a 1.5 ml centrifuge tube, which was vortexed briefly. 6 µl of the pooled library was diluted in 594 µl of HT1, followed by vortexing thoroughly.

The diluted library was heated for 2 minutes at 96°C on a heat block, and immediately after the tube was placed in an ice-water bath for 5 minutes.

2.8.2.4 Next generation sequencing

At least five hours prior to loading the library onto the MiSeq, a MiSeq reagent cartridge was removed from -20°C storage and thawed in a water bath at room temperature. Sequencing was initially carried out by loading the dilute amplicon library (Chapter 2, Section 2.8.2.3) into the sample reservoir in a thawed 300- or 500-cycle MiSeq reagent cartridge (v2). Cartridges were loaded onto one of two independent Illumina MiSeq platforms (Illumina, USA); MiSeq 1 used the 500-cycle cartridge and 175 cycles of paired-end sequencing were performed whereas MiSeq 2 used the 300-cycle cartridge and 150 cycles of paired-end sequencing. Sequencing reads from MiSeq 1 were trimmed using Trimmomatic (Bolger et al. 2014). All subsequent runs were carried out using the 300 cycle kit (150 cycles).

2.9 Bioinformatics and data analyses

A bioinformatics pipeline for variant calling was established using BWA version 0.5.9 (Li & Durbin 2009), SAMtools version 0.1.18 (H. Li et al. 2009), GATK version 1.3-145 (DePristo et al. 2011; McKenna et al. 2010) and SnpEff version 2.04 (Cingolani et al. 2012). A custom Perl script was written to facilitate data parsing and analysis of pipeline output (Custom Script 1). All analysis was performed on a workstation running an Ubuntu operating system (version 11.10 or later). All software was run using the Unix command line, using default parameters unless specified differently. The data flow is shown in Figure 2.1.

2.9.1 Genome alignment

BWA was used to first create an index of release GRCh37 of the human reference genome, using the *bwa index* command. Sequencing reads were then mapped to the reference genome using BWA's *aln* and *sampe* commands. In this step, each aligned read was tagged with the sample ID and the sequencing platform (Illumina). The output from this step was in sequence/alignment mapping (SAM) format.

2.9.2 Data reformatting and base quality score recalibration

SAMtools' *view* command was used to convert the SAM files to a smaller binary/alignment mapping (BAM) format. The BAM files were sorted by read name using *samtools sort* so that read mate coordinates and insert size could be added with *samtools fixmate*. The files were then resorted by genomic coordinates, again using the *samtools sort* command. For whole exome sequencing (Chapter 3), PCR duplicated reads were removed via the *samtools rmdup* command. SAMtools' *index* command was used to create an index for faster downstream access to each BAM file.

Local realignment was performed on the sequencing reads using GATK's RealignerTargetCreator and IndelRealigner tools. BAM files generated at this stage were used for calculating depth of coverage and data metrics. The following steps were taken to recalibrate base quality scores for each of the exomes, as recommended in the GATK best practice documentation. GATK's *CountCovariates* tool was first used to generate a table of recalibrated quality scores. This tool suspects all sites which mismatch the reference genome are sequencing errors, so a list of known variants from dbSNP version 132 was used. Several factors were considered by this tool including the sequencing cycle, read group, quality score and dinucleotide covariates. The

TableRecalibration tool was then used to recalibrate the original base quality scores in the BAM file.

2.9.3 Variant calling and quality score recalibration

Variant calling was primarily performed using GATK's Unifed Genotyper or HaplotypeCallertool. Both single nucleotide variants (SNVs) and insertions/deletions (indels) were called. dbSNP version 132 or later was used to supply rsIDs for known variants. Variants were output in variant call format (VCF) files.

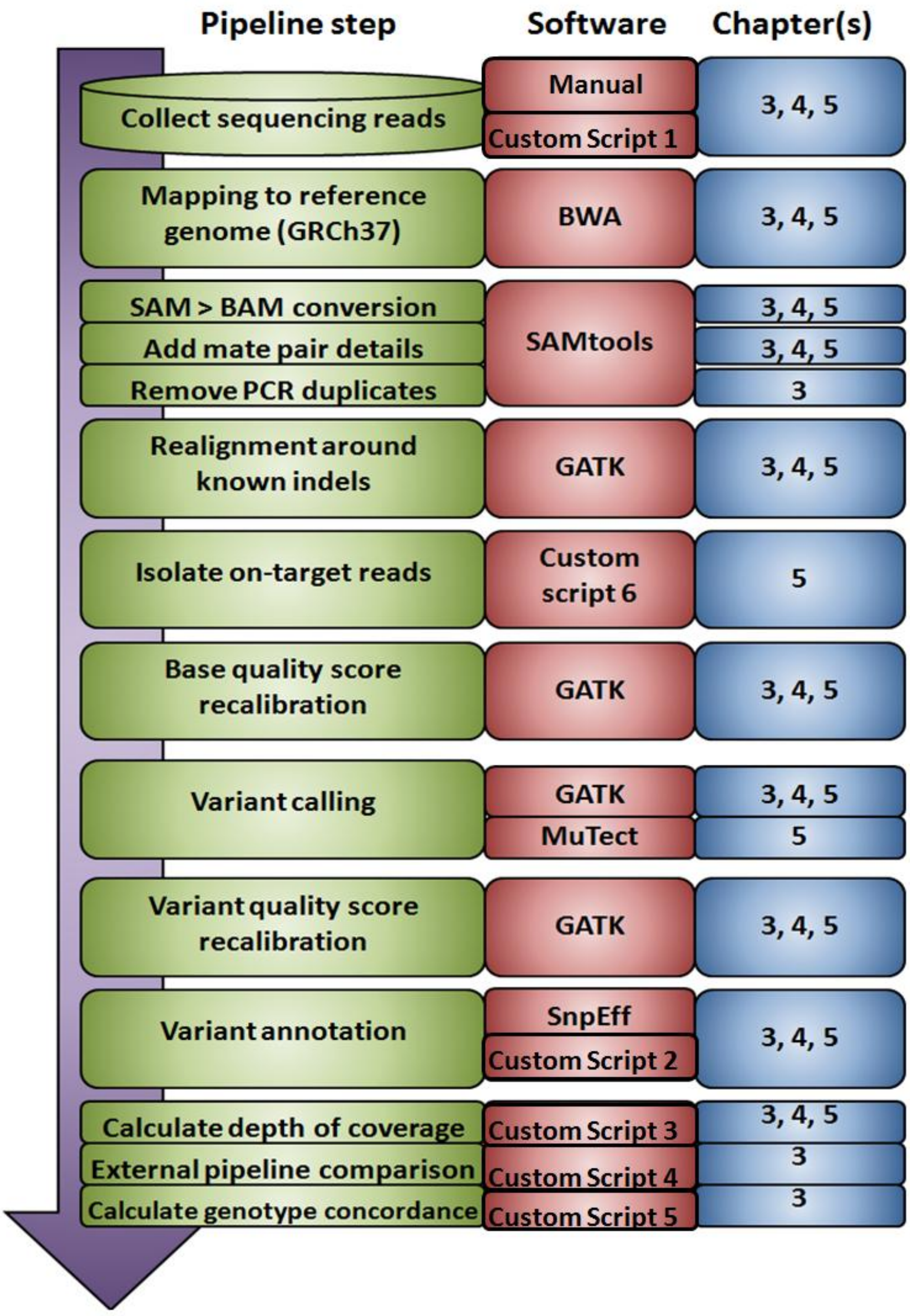


Figure 2.1. Flow of data through our bioinformatics pipeline. The purple arrow indicates the flow of data. Green boxes indicate individual steps in the pipeline, whereas red

boxes indicate the software used for each step. Blue boxes direct to the results chapter that each step was involved in.

As recommended in the GATK best practices, variant quality score recalibration (VQSR) was performed. GATK's *SelectVariants* was used to separate SNPs and indels into two separate files. GATK's *VariantRecalibrator* was employed to recalibrate variant scores for SNPs, using training sites from HapMap release 3.3, Illumina's Omni2.5M SNP microarray and dbSNP. Prior probability values specified for each of these training datasets were 15, 12 and 8, respectively. VariantRecalibrator was used with the *mG 4* and *percentBad 0.05* arguments specified as each exome was analysed individually, as recommended by GATK's best practices. The SNPs and their recalibrated scores were added to new VCF files using GATK's *ApplyRecalibration* tool, and the recalibrated SNPs were recombined with the indels from the original VCF file using *CombineVariants*.

For somatic variant calling, we also used MuTect (Cibulskis et al. 2013). MuTect was used with default parameters from the command line, and given the reference human genome (GRCh37) and files containing variants in dbSNP and COSMIC. Only variants that passed MuTect's variant confidence filters were considered genuine, as determined by a ‘PASS’ label in the filter column of the output variant file.

2.9.4 Variant annotation

SnpEff (version 2.0.4 or later) was utilised, with default parameters and reference data from GRCh37.64, to perform variant annotation for variants after VQSR. GATK's *VariantAnnotator* was used to add the resulting annotations into the final VCF file for each exome. Further annotation was performed using Custom Script 2, and allele frequencies from dbSNP, 1000 Genomes and COSMIC were added for each variant. The number of sequenced samples each variant was detected in was calculated, as well as presence in germline and/or tumour tissue.

2.9.5 Depth of coverage calculations

The exome capture kits used by BGI and University of North Carolina targeted exons within CCDS (Pruitt et al., 2006) and RefSeq (Pruitt, Tatusova, & Maglott, 2007). To calculate the coverage across a gene, coordinates of coding exons were extracted from CCDS release 9 or later (ftp://ftp.ncbi.nlm.nih.gov/pub/CCDS/current_human/, data made public 9/7/2011) and RefSeq Genes datasets (http://genome.ucsc.edu/, accessed 16/5/2012). Gene coordinates from CCDS were considered if the gene had a 'public' status. For genes with multiple transcripts, the transcript with the most coding bases was chosen. The longest transcript of each gene in RefSeq was also chosen, providing a transcript for the gene had not already been obtained from CCDS.

To calculate the average sequencing depth across a gene's open reading frame (ORF), a custom perl script (Custom Script 3) was written that integrated SAMtools' *mpileup* command. The script examined BAM files and extracted bases in realigned sequencing reads that mapped to an ORF of interest. The '-A', '-E' and '-Q 0' arguments were given to SAMtools so that anomalous reads were included, per-base quality scores were calculated at run time, and bases were excluded if they had a quality of ≤0. The total coverage for each coding base of an ORF was summed and divided by the length of the ORF to give the average coverage. The percent of the ORF reaching specified coverage depths was also calculated. Average depth across each exome was calculated by summing the average ORF coverage and dividing by the number of ORFs.

2.9.6 Bioinformatics pipeline comparison

Using their own bioinformatics pipeline, BGI supplied variants calls for 10 exomes. For the same 10 exomes (using the same set of raw sequencing reads), we compared variants called by our pipeline to those called by BGI. A custom script (Custom Script 4) was written for this task.

For each comparison, variants were classed as concordant if the genomic coordinates, reference allele, variant allele and the allele zygosity agreed between both call sets in the comparison.

Variants called by our pipeline were broken down into four different subcategories including all SNPs, novel SNPs, all indels and novel indels. Variants were classed as novel (‘potentially pathogenic’) if they were not observed in dbSNP version 132. Concordance rates between call sets were calculated for each of the four subcategories separately. To increase the overlap between our variant call set and BGI's call set, we performed the variant calling step using GATK's Unified Genotyper with a minimum base quality (MBQ) parameter = 10 and a stand_emit_conf parameter = 10.

2.9.7 Calculating concordance with prior genotyping data

Prior genotyping data was previously obtained using the Illumina GoldenGate assay for 153 SNPs in 47/50 (94%) exomes. The genotypes called by our bioinformatics pipeline were used in a comparison against this data. To do this, Custom Script 5 was written. The rsID for each prior genotype were used to query dbSNP (version 135) to retrieve the genomic coordinates and known variant alleles for each assayed SNP.

GATK's Unified Genotyper was used to call alleles at each of the 153 variant sites in the 47 exomes. Three parameter sets were used when running Unified Genotyper, the first being the default settings (minimum base quality = 17, variant score = 30). The second set of parameters tested used minimum base quality = 10 and variant score = 10. The final set of tested parameters was identical to the second set, plus a per-base alignment quality (BAQ) calculation.

2.9.8 Removal of off-target sequences

Off-target reads were removed by isolating on target reads to new BAM files using Custom Script 6 (Chapter 5). Amplicon targets from the TruSeq Custom Amplicon Panel were used to define target regions. For targets that overlapped, a contiguous region covering the targets affected was created. Coordinates for the targets or contiguous regions were then used to isolate sequencing reads from individual BAM files; this was performed using SAMtools view. A sequencing read was designated as ‘on target’ if one of the terminal (start or end) mapping coordinates matched one of the targets’ terminal coordinates. The opposite end of the read had to lie closer to the other terminus. Reads matching these criteria were isolated by printing to a new BAM file. Any reads that did not match these criteria was labelled as ‘off target’ and ignored.

2.9.9 Online resources

COSMIC data: http://cancer.sanger.ac.uk/cosmic.

CCDS data:ftp://ftp.ncbi.nlm.nih.gov/pub/CCDS/current_human/, data made public 9/7/2011).

RefSeq data obtained from: http://genome.ucsc.edu/, accessed 16/5/2012).

Chapter 3 – Developing and testing a pipeline for variant calling from germline exome re-sequencing data

3.1 Details of work performed

In this chapter, exome sequencing was performed by BGI. Bioinformatics analyses were performed by myself, and *in vitro* validation (including PCR and Sanger sequencing) were performed by myself and Dr Christopher Smith. Genotyping of samples prior to exome sequencing was performed by Dr Christopher Smith.

3.2 Introduction

Sequencing the exome allows for sensitive and specific detection of common and rare variants. Ng et al. (2009) showed that in Mendelian disease, exome sequencing can identify candidate genes in small numbers of individuals. Timmermann et al. (2010) were amongst the first to show that the exome can be sequenced in primary colon cancers as well as normal colon tissue from the same patient. Furthermore, they showed that rare (MAF <1%) or novel SNVs can be identified via this approach. Previously established methods, such as genotyping and Sanger sequencing, have developed to be highly specific, but may lack the power required to detect a large number of low frequency alleles simultaneously within a sample. NGS overcomes these draw backs, but many factors such as bioinformatic parameters may lead to inaccurate variant calls.

Most bioinformatics pipelines follow a simple formula; this usually consists of sequencing read alignment, data formatting, variant calling and annotation. The HugeSeq pipeline (Lam et al. 2012) follows such a formula, using software such as BWA, SAMtools, Picard and GATK. Throughout this pipeline several steps are taken to ensure higher variant calling accuracy, including removal of PCR duplicates, local alignment around known indels and recalibration of base qualities. This pipeline and its underlying software have proven suitable for Illumina sequencing data (Lam et al. 2012).

Other efforts have resulted in software suites accessible online such as Galaxy (Goecks et al. 2010; Giardine et al. 2005) that enable access to bioinformatics tools and high performance computing clusters. These suites enable users to upload data files, have it analysed on external hardware and have the output files returned.

As our grand aim was to detect low frequency somatic mutations (Chapters 4 and 5), we designed a workflow to ensure our pipeline parameters and protocols were rigourous enough to produce a highly sensitive call set. In this chapter, we sought to adopt a pipeline and adapt parameters for highly accurate variant calling using exome re-sequencing data from 50 germline samples from patients with CRC. We then designed a series of custom scripts to measure metrics from our sequencing efforts.

To ensure the exome re-sequencing data had sufficient coverage for reliable variant calling, we developed a script that calculated coverage over the exome. To measure the pipeline's ability to call commonly-predicted and therefore genuine variants, we developed a script that compared variant calls to those of BGI's pipeline. To test the accuracy of the predicted genotype for each variant call, a script was developed that compared predicted genotypes to previous genotyping results obtained on the Illumina GoldenGate platform. We then identified a series of potential disease causing variants from genes that were likely to play a role in colorectal tumourigenesis and determined whether these predicted variants were genuine by laboratory validation.

3.3 Materials and methods

3.3.1 Samples

We prepared blood DNA samples from unrelated sporadic patients with aCRC from the UK-national, multi-centre, randomised controlled trials (RCTs) COIN (Maughan et al. 2011) and COIN-B (Adams et al. 2011) (Chapter 2, Section 2.2).

3.3.2 Exome re-sequencing

DNA samples from 40 patients were processed using Agilent's SureSelect Target Enrichment platform (10 with 50 Mb and 30 with 44 Mb capture kits) and sequenced on an Illumina HiSeq2000 (10 at ≥50x coverage and 30 at ≥30x coverage) by BGI (Hong Kong) (Chapter 2, Section 2.8.1). DNA samples from a further 10 patients were processed using the Roche Nimblegen SeqCap EZ Exome capture kit and sequenced on an Illumina Genome Analyser (at ~30x coverage) (Chapter 2, Section 2.8.1).

3.3.3 Bioinformatics analyses

Our default pipeline used the backtrack algorithm in BWA version 0.5.9 (Li & Durbin 2009) to align the paired-end reads to the reference genome and alignments were outputted in SAM format (Li et al. 2009) (Chapter 2, Section 2.9.1). SAMtools version 0.1.18 (Li et al. 2009) was used for to sort reads in each BAM file by genomic coordinates and to remove PCR duplicated reads (Chapter 2, Section 2.9.2). Multiple BAM files for a single patient were merged into a single file. The UnifiedGenotyper tool in the Genome Analysis Toolkit (GATK) version 1.3 (McKenna et al. 2010; DePristo et al. 2011) was used for with default parameters for variant calling. UnifiedGenotyper's default parameters require bases to have a minimum base quality = 17 for inclusion in variant calling, and only outputs variants if they have a quality score ≥ 30. Variant quality score re-calibration was performed using GATK's VariantRecalibrator and ApplyRecalibration tools with their default parameters (Chapter 2, Section 2.9.3). SnpEff (Cingolani et al. 2012) was used to annotate variants and extra annotation was added with a custom Perl (Chapter 2, Section 2.9.4). Extra annotation included the allele frequencies from the 1000 Genomes Project (1000 Genomes Project Consortium et al. 2015) and Exome Sequencing Project (EVS 2014)

We optimised our pipeline parameters by relaxing the strictness of variant calling. For this, we lowered the minimum base Phred quality score considered by UnifiedGenotyper to 10. We also lowered the standard emitting quality score of variants to be included in the output VCF files to 10.

Coverage across each exome (18,441 genes with public status in the CCDS (Pruitt et al. 2009) dataset and 4,871 genes in RefSeq (Pruitt et al. 2007)) and for the 1,138 genes considered likely to play a role in colorectal tumourigenesis was calculated using SAMtools in combination with a custom Perl script (Custom Script 3, Chapter 2, Section 2.9.5).

To compare calls made by sequencing to genotypes produced on the GoldenGate platform, the UnifiedGenotyper tool was applied to the re-calibrated BAM files to emit all 153 analysed SNP sites regardless of genotype (Chapter 2, Section 2.9.7). Three parameter sets were tested. The first set used UnifiedGenotyper’s default settings and the second set used a minimum base quality = 10 and variant score = 10. The final parameters were identical to the second set with the addition of a per-base alignment quality calculation.

3.3.4 BGI pipeline

BGI's pipeline aligns sequencing reads to the reference human genome (version hg19) using SOAP/SOAP2 (R. Li et al. 2008; R. Li et al. 2009). A maximum of 3 mismatches per read were allowed, and the parameters –a, -b, -D, -o, -u, -p, -2, -m, -x, -s 40, -l 35 and –v 3 were specified for this step. Reads in the resulting BAM files were sorted by genomic coordinates and duplicate reads were removed with Picard (http://picard.sourceforge.net). Where multiple BAM files were present for a single patient, the BAM files were merged. GATK's RealignerTargetCreator and IndelRealigner tools (McKenna et al. 2010; DePristo et al. 2011) were used to realign reads mapping to the exome target region. GATK's UnifiedGenotyper tool was used for variant calling with and variant filtering was performed with GATK's VariantFiltration tool. BGI used default parameters for all GATK tools.

3.3.5 PCR and Sanger sequencing

To validate called variants, PCR and Sanger sequencing was carried out as previously described (Chapter 2, Section 2.5).

3.4 Results

3.4.1 Whole exome sequencing

We exome re-sequenced blood DNA from 50 unrelated sporadic patients with aCRC that had been recruited into the UK-national, multi-centre RCTs COIN and COIN-B (Table 3.1). Exomes were sequenced using two different NGS platforms (a GenomeAnalyser IIx and a Hiseq2000) and DNA enrichment kits. At 30x coverage, the GenomeAnalyser IIx yielded an average of 78,387,002 reads per exome, whereas the HiSeq2000 sequenced 26,782,135 reads per exome. Exomes sequenced on the GenomeAnalyser IIx were sequenced to an average depth of 30.4x, whereas the Hiseq2000 sequenced to an average of 24.5x and 44.5x using the SureSelect Human All Exon V2 and 50 Mb kits, respectively.

On average, 46,179,764 paired-end sequencing reads were obtained per exome. From these, an average of 97.3% reads were mapped, of which 95.2% reads were successfully mapped as a correct mate pair (Table 3.1). A pipeline integrating BWA, SAMtools, GATK and SnpEff was established to produce a variant call set. A custom script (Custom Script 1) was written to manage data parsing throughout the pipeline.

3.4.2 Sequencing coverage

Custom Script 3 was written and used in conjunction with SAMtools to calculate coverage across each exome. From CCDS and RefSeq, 18,441 and 4,871 coding genes were identified respectively, giving a total of 23,312 genes used in exome coverage calculations. Across all 50 samples sequenced, a mean depth of 30.2x (range 22.2–52.5x) was observed in coding regions within these genes. Average coverage was calculated across the ORF of each gene at varying depths. At 10x depth, 69.6% of each ORF was covered. At depths of 20x and

30x, 56.1% and 42.6% of ORFs were covered on average, respectively (Table 3.1).

Table 3.1. Summary of sequencing metrics for each exome. All platforms produced paired-end reads that mapped well and in correct pairs. Key: All Exon 50Mb = SureSelect All Exon 50 Mb capture kit; All Exon V2 = SureSelect All Exon V2 capture kit; GAIIx = GenomeAnalyser IIx; SeqCap EZ = SeqCap EZ exome V2 capture kit.

Sample ID	**Exon capture kit**	**NGS platform**	**Depth**	**Sequencing reads**	**Mapped reads (%)**	**Correct mate pair (%)**	**Average ORF (%) covered**		
							≥10x	**≥20x**	**≥30x**
1	All Exon V2	Hiseq 2000	25.3x	25,996,910	98.8	98.1	68.3	51.0	34.8
2	All Exon V2	Hiseq 2000	25.3x	26,018,786	98.9	98.3	69.0	51.7	35.3
3	All Exon V2	Hiseq 2000	25.1x	25,775,084	98.9	98.3	67.8	50.2	34.3
4	All Exon V2	Hiseq 2000	24.9x	29,112,742	98.3	97.4	67.3	49.6	33.9
5	All Exon V2	Hiseq 2000	25.2x	25,654,782	98.9	98.2	68.4	50.9	34.7
6	All Exon V2	Hiseq 2000	27.5x	24,227,224	98.2	97.4	68.6	52.2	36.9
7	All Exon V2	Hiseq 2000	47.7x	52,186,484	98.5	96.4	76.0	69.7	62.5
8	All Exon V2	Hiseq 2000	25.1x	25,567,114	98.8	98.2	67.6	50.0	34.3
9	All Exon V2	Hiseq 2000	25.1x	25,416,220	99.0	98.4	67.9	50.4	34.5
10	All Exon V2	Hiseq 2000	25.6x	24,145,266	97.6	96.6	67.3	49.3	33.7

11	All Exon V2	Hiseq 2000	25.1x	26,246,846	98.8	98.1	68.5	50.9	34.5
12	All Exon V2	Hiseq 2000	25.3x	26,101,224	99.0	98.4	68.2	50.9	34.9
13	All Exon V2	Hiseq 2000	25.2x	25,945,522	98.9	98.3	68.9	51.6	35.1
14	All Exon V2	Hiseq 2000	25.2x	26,094,882	99.0	98.4	69.1	52.4	35.4
15	All Exon V2	Hiseq 2000	25.1x	25,655,404	99.0	98.5	68.7	51.2	34.7
16	All Exon V2	Hiseq 2000	25.2x	25,394,638	99.0	98.4	69.0	51.7	35.1
17	All Exon V2	Hiseq 2000	25.0x	26,080,956	99.0	98.5	68.9	51.4	34.7
18	All Exon V2	Hiseq 2000	25.0x	24,872,560	99.0	98.4	68.1	50.5	34.4
19	All Exon V2	Hiseq 2000	23.6x	26,679,426	98.0	94.5	67.3	50.8	34.0
20	All Exon 50Mb	Hiseq 2000	43.6x	66,221,614	96.6	91.5	79.7	72.2	62.8
21	All Exon V2	Hiseq 2000	22.4x	27,109,810	98.0	94.5	66.1	48.7	31.6
22	SeqCap EZ	GAIIx	22.8x	92,339,526	87.2	85.6	59.8	45.7	32.7
23	All Exon 50Mb	Hiseq 2000	47.0x	76,298,952	97.1	93.5	80.6	73.9	65.7
24	All Exon V2	Hiseq 2000	23.2x	27,009,544	98.1	94.6	66.5	50.1	33.5
25	All Exon 50Mb	Hiseq 2000	52.5x	88,964,856	97.8	94.5	80.8	75.3	69.4
26	All Exon 50Mb	Hiseq 2000	46.4x	74,679,802	98.0	95.1	79.4	73.0	65.4
27	All Exon 50Mb	Hiseq 2000	48.7x	79,180,926	97.6	94.4	81.2	75.1	67.7
28	All Exon V2	Hiseq 2000	22.2x	26,069,130	98.0	94.4	65.4	47.4	30.7
29	All Exon V2	Hiseq 2000	25.1x	34,822,078	97.6	93.9	67.7	52.4	36.9
30	All Exon 50Mb	Hiseq 2000	38.6x	54,062,870	97.0	92.6	78.6	69.6	57.7

31	All Exon V2	Hiseq 2000	22.3x	24,955,050	98.2	94.9	66.0	48.6	31.5
32	SeqCap EZ	GAIIx	22.9x	78,608,136	89.9	88.0	62.6	48.8	34.0
33	All Exon V2	Hiseq 2000	23.0x	28,018,264	98.2	94.9	65.4	48.7	32.9
34	All Exon V2	Hiseq 2000	23.6x	29,585,984	98.0	94.5	66.5	50.2	34.1
35	All Exon V2	Hiseq 2000	22.4x	29,853,450	97.6	93.9	65.7	48.3	31.7
36	All Exon V2	Hiseq 2000	24.3x	28,797,442	98.0	94.3	66.8	51.0	35.4
37	All Exon 50Mb	Hiseq 2000	47.1x	77,604,370	97.7	94.8	80.0	73.7	66.2
38	All Exon 50Mb	Hiseq 2000	39.3x	57,691,734	96.8	92.1	78.6	70.1	58.9
39	All Exon V2	Hiseq 2000	23.4x	25,680,424	98.2	94.9	67.0	50.5	33.8
40	SeqCap EZ	GAIIx	37.5x	83,998,846	95.9	95.4	66.0	59.6	52.2
41	All Exon 50Mb	Hiseq 2000	36.6x	51,397,194	96.8	91.9	77.7	68.4	55.6
42	SeqCap EZ	GAIIx	39.8x	93,383,898	96.0	95.4	65.1	58.6	51.9
43	SeqCap EZ	GAIIx	34.1x	75,924,258	93.1	91.9	66.5	59.9	51.4
44	SeqCap EZ	GAIIx	23.5x	64,212,086	94.1	91.7	63.9	49.8	35.0
45	SeqCap EZ	GAIIx	29.4x	63,403,226	98.6	97.9	66.3	56.8	45.3
46	SeqCap EZ	GAIIx	28.7x	62,329,802	98.2	97.3	66.1	56.3	44.3
47	All Exon 50Mb	Hiseq2000	45.5x	70,147,450	97.4	93.8	80.2	73.5	65.1
48	SeqCap EZ	GAIIx	26.3x	78,854,006	92.0	90.3	63.9	52.8	40.1
49	All Exon V2	Hiseq 2000	25.1x	29,795,162	98.2	95.1	68.0	52.8	37.1
50	SeqCap EZ	GAIIx	39.0x	90,816,234	95.7	95.0	65.5	59.0	52.2

Mean	-	-	30.2x	46,179,764	97.3	95.2	69.6	56.1	42.6

3.4.3 Pipeline output comparisons

For ten samples, we compared variant calls made through our pipeline to those that had independently been determined using the same raw sequencing reads through the pipeline at BGI. Using our pipeline with default parameters, we called 3.1-fold more SNPs and 3.2-fold more indels than BGI. Importantly, we detected on average 93.4% (45,407/48,636) and 94.1% (2,062/2,192) of BGI's SNPs and indels respectively. Of the novel variants, we detected on average 59% (1,302/2,209) and 87.5% (673/769) of BGI's SNPs and indels respectively.

We optimised our pipeline's parameter settings to increase the detected number of variants. The pipeline started with read alignment by BWA's backtrack algorithm (Li & Durbin 2009), sorting reads by genomic coordinates and removing PCR duplicates by SAMtools (H. Li et al. 2009) Using optimised parameter settings *(*minimum base quality = 10 and variant quality = 10) we detected 1.6 fold more SNPs and 2.7 fold more novel SNPs than when using our default parameters. Using these optimised parameters, the overlap with BGI's pipeline output increased to 94.3% for SNPs and 94.7% for indels. For novel SNPs and indels, the overlap increased to 63.3% and 88.9%, respectively (Table 3.2, Figure 3.1).

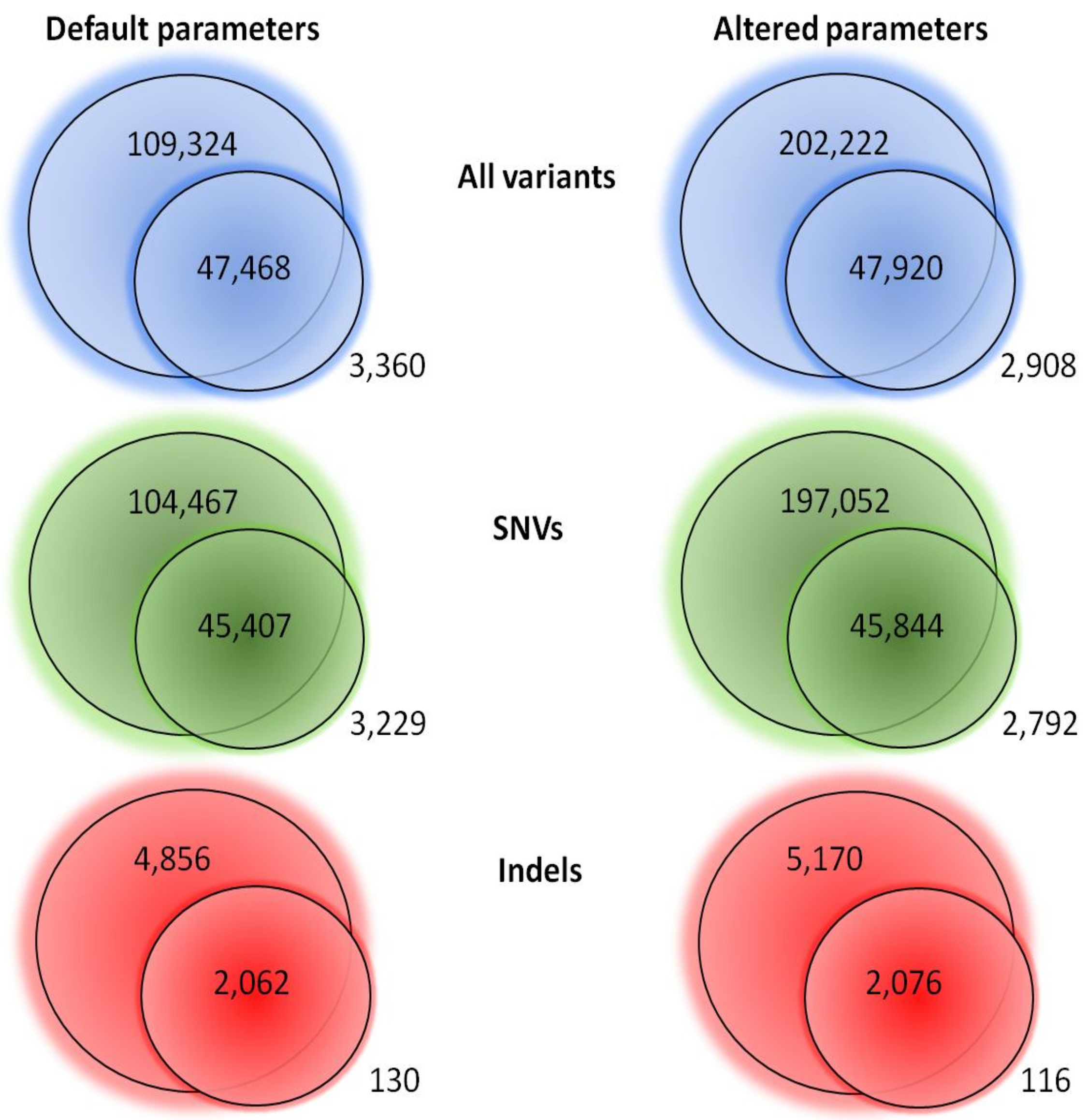


Figure 3.1. Venn diagrams showing the overlap between number of variants called by our pipeline (larger circles) and BGIs pipeline (smaller circles). Altering the parameters slightly reduced the number of variants called exclusively by BGI's pipeline.

Table 3.2. Comparisons between our pipeline's call set and the call set produced by the BGI pipeline. MBQ, minimum base quality; stand_emit_conf, the confidence score used by GATK when emitting variants.

Parameter set	**Sample ID**	**SNPs**						**Indels**					
		All			**Novel**			**All**			**Novel**		
		Our pipeline	**BGI pipeline**	**Overlap (%)**	**Our pipeline**	**BGI pipeline**	**Overlap (%)**	**Our pipeline**	**BGI pipeline**	**Overlap (%)**	**Our pipeline**	**BGI pipeline**	**Overlap (%)**
Default (MBQ 17, stand_emit_conf 10)	20	121,449	48,306	92.9	10,359	2,427	56.9	5,577	2,052	93.7	886	692	86.9
	23	154,359	50,640	92.7	12,515	2,415	56.2	7,345	2,221	94.3	1,203	784	88.9
	25	160,078	52,642	91.7	12,251	2,739	49.9	8,346	2,377	94.8	1,372	859	88.4
	26	157,857	44,126	96.4	12,893	1,113	75.6	8,767	2,435	93.9	1,441	895	87.5
	27	152,419	47,323	96.8	41,005	1,585	80.8	8,109	2,315	94.7	1,442	812	89.0
	30	118,135	46,976	92.8	9,374	2,274	54.7	5,103	1,932	93.4	776	649	85.5
	37	157,362	51,326	91.6	13,030	2,604	51.5	8,039	2,252	94.8	1,363	799	88.4
	38	122,453	48,110	92.7	9,053	2,363	54.0	5,626	2,077	93.4	828	723	85.5
	41	114,478	46,799	92.8	8,957	2,239	55.5	5,282	1,987	93.3	767	663	86.4
	47	146,902	50,110	92.5	11,471	2,329	54.5	6,984	2,269	94.2	1,151	811	88.9
	Average	**149,874**	**48,636**	**93.4**	**14,091**	**2,209**	**59.0**	**6,918**	**2,192**	**94.1**	**1,123**	**769**	**87.5**
Altered (MBQ	20	220,462	48,306	93.9	43,119	2,427	61.4	5,830	2,052	94.4	1,017	692	88.4

10, stand_emit_conf 10)	23	253,853	50,640	93.6	45,554	2,415	60.5	7,736	2,221	95.1	1,418	784	89.9
	25	298,625	52,642	92.6	36,861	2,739	54.3	8,783	2,377	95.3	1,608	859	89.5
	26	269,242	44,126	97.2	36,674	1,113	81.4	9,303	2,435	94.7	1,740	895	89.4
	27	245,668	47,323	97.5	41,005	1,585	80.8	8,109	2,315	94.7	1,442	812	89.0
	30	206,528	46,976	93.9	33,848	2,274	59.9	5,363	1,932	94.0	897	649	87.1
	37	250,756	51,326	92.6	34,624	2,604	56.3	8,469	2,252	95.4	1,606	799	90.0
	38	217,709	48,110	93.8	33,145	2,363	58.7	5,928	2,077	94.1	967	723	86.9
	41	201,095	46,799	93.8	28,616	2,239	60.4	5,568	1,987	94.5	915	663	89.0
	47	265,024	50,110	93.5	47,155	2,329	59.2	7,373	2,269	94.8	1,348	811	90.0
	Average	**242,896**	**48,636**	**94.3**	**38,060**	**2,209**	**63.3**	**7,246**	**2,192**	**94.7**	**1,296**	**769**	**88.9**

3.4.4 Genotype concordance

47/50 (94%) of the aCRC patients had previously been genotyped for 153 coding region SNPs with MAFs >5%, spanning a total of 97 genes, using the Illumina Goldengate platform (San Diego, CA) (our unpublished data). We compared the genotypes to the calls made from our exome re-sequencing pipeline (after excluding low confidence calls or those with <4x coverage). Using our default parameters, 6,739 genotypes were called, of which, 6,723 (99.76%) were concordant. Using the second set of parameters (minimum base quality = 10 and variant quality = 10), 6,739 genotypes were called and from these, 6,724 (99.78%) calls were concordant. The final set of parameters tested (minimum base quality = *10*, variant quality = 10 and enabling a BAQ calculation) produced 6,736 genotypes, of which 6,720 (99.76%) were concordant (Table 3.3).

Across all 3 parameter settings, there was a total of 17 discrepancies between previous genotyping and our sequencing data. The majority (n = 10 inconsistencies, 59%) of these had a low sequencing depth (≤8x). A further 4 inconsistencies (23%) were due to variants being incorrectly called by our pipeline as heterozygous due to a fraction of reads exhibiting mutant alleles; mutant:wild-type allele ratios ranged from 1:7 to 1:11. The final 3 discrepancies (18%) had a low quality score (quality by depth <5).

3.4.5 *In vitro* validation of variants predicted *in silico*

We studied 1,138 genes that were likely to be involved in colorectal tumourigenesis to determine the accuracy of our pipeline's variant calls to identify potentially disease-causing mutations, which we defined as rare (minor allele frequency [MAF] ≤1%) or novel variants that were predicted to truncate the encoded protein. Genes included were from the base excision repair (BER) pathway (141 genes), the mismatch repair (MMR) pathway (28 genes), the Wnt signalling pathway (82 genes), the bone morphogenetic protein (BMP) signalling pathway (79 genes), the Notch signalling pathway (73 genes), and other pathways implicated through genome-wide association studies (GWAS, 220 genes) (Figure 3.2; Smith et al. 2013). From transposon-based screens in mice (March et al. 2011) a further 545 genes were identified. Note that some genes were found in multiple studies or pathways, so overall 1,138 genes in total were included in our analyses.

Table 3.3. The concordances between genotyping carried out using Illumina's GoldenGate assay and variant calls from our pipeline. When calling variants, we used 3 different sets of parameters.

Pipeline parameters	**Default**		**MBQ = 10, emitting threshold = 10**		**MBQ = 10, emitting threshold = 10, BAQ calculation on**	
Sample ID	**Concordant calls**	**%**	**Concordant calls**	**%**	**Concordant calls**	**%**
1	143/143	100.0	143/143	100.0	143/143	100.0
2	143/143	100.0	143/143	100.0	142/142	100.0
3	145/146	99.3	145/146	99.3	145/146	99.3
4	142/142	100.0	142/142	100.0	142/142	100.0
5	146/147	99.0	146/147	99.3	146/147	99.3
6	139/139	100.0	139/139	100.0	139/139	100.0
7	146/146	100.0	146/146	100.0	146/146	100.0
8	138/138	100.0	138/138	100.0	138/138	100.0
9	143/143	100.0	143/143	100.0	143/143	100.0
10	143/144	99.3	143/144	99.3	142/143	99.3
11	143/144	99.3	143/144	99.3	143/145	98.6
12	138/139	99.0	138/139	99.3	138/139	99.3
13	148/148	100.0	148/148	100.0	147/147	100.0
14	146/148	98.7	145/147	98.6	145/147	98.6
15	145/145	100.0	145/145	100.0	145/145	100.0
16	147/147	100.0	147/147	100.0	147/147	100.0
17	142/144	99.0	142/144	98.6	142/144	98.6
18	145/145	100.0	145/145	100.0	145/145	100.0
19	145/145	100.0	145/145	100.0	144/144	100.0
20	0/0	-	0/0	-	0/0	-
21	141/141	100.0	141/141	100.0	141/141	100.0
22	130/130	100.0	131/131	100.0	130/130	100.0

23	152/152	100.0	152/152	100.0	152/152	100.0
24	143/143	100.0	143/143	100.0	143/143	100.0
25	151/151	100.0	151/151	100.0	151/151	100.0
26	152/152	100.0	152/152	100.0	152/152	100.0
27	152/152	100.0	152/152	100.0	152/152	100.0
28	147/147	100.0	147/147	100.0	147/147	100.0
29	144/144	100.0	144/144	100.0	144/144	100.0
30	150/151	99.3	150/151	99.3	150/151	99.3
31	145/146	99.3	145/146	99.3	145/146	99.3
32	136/136	100.0	136/136	100.0	136/136	100.0
33	139/140	99.3	139/140	99.3	139/140	99.3
34	143/143	100.0	143/143	100.0	143/143	100.0
35	145/145	100.0	145/145	100.0	145/145	100.0
36	0/0	-	0/0	-	0/0	-
37	152/152	100.0	152/152	100.0	152/152	100.0
38	149/149	100.0	149/149	100.0	149/149	100.0
39	142/142	100.0	142/142	100.0	142/142	100.0
40	136/137	99.0	136/136	100.0	135/135	100.0
41	150/150	100.0	150/150	100.0	150/150	100.0
42	136/136	100.0	136/136	100.0	136/136	100.0
43	135/135	100.0	135/135	100.0	135/135	100.0
44	133/133	100.0	133/133	100.0	134/134	100.0
45	136/136	100.0	136/136	100.0	136/136	100.0
46	131/131	100.0	131/131	100.0	131/131	100.0
47	152/152	100.0	152/152	100.0	152/152	100.0
48	131/134	97.8	131/134	97.8	132/135	97.8
49	143/143	100.0	144/144	100.0	144/144	100.0
50	0/0	-	0/0	-	0/0	-
Total	**6723/6739**	**99.8**	**6724/6739**	**99.8**	**6720/6736**	**99.8**

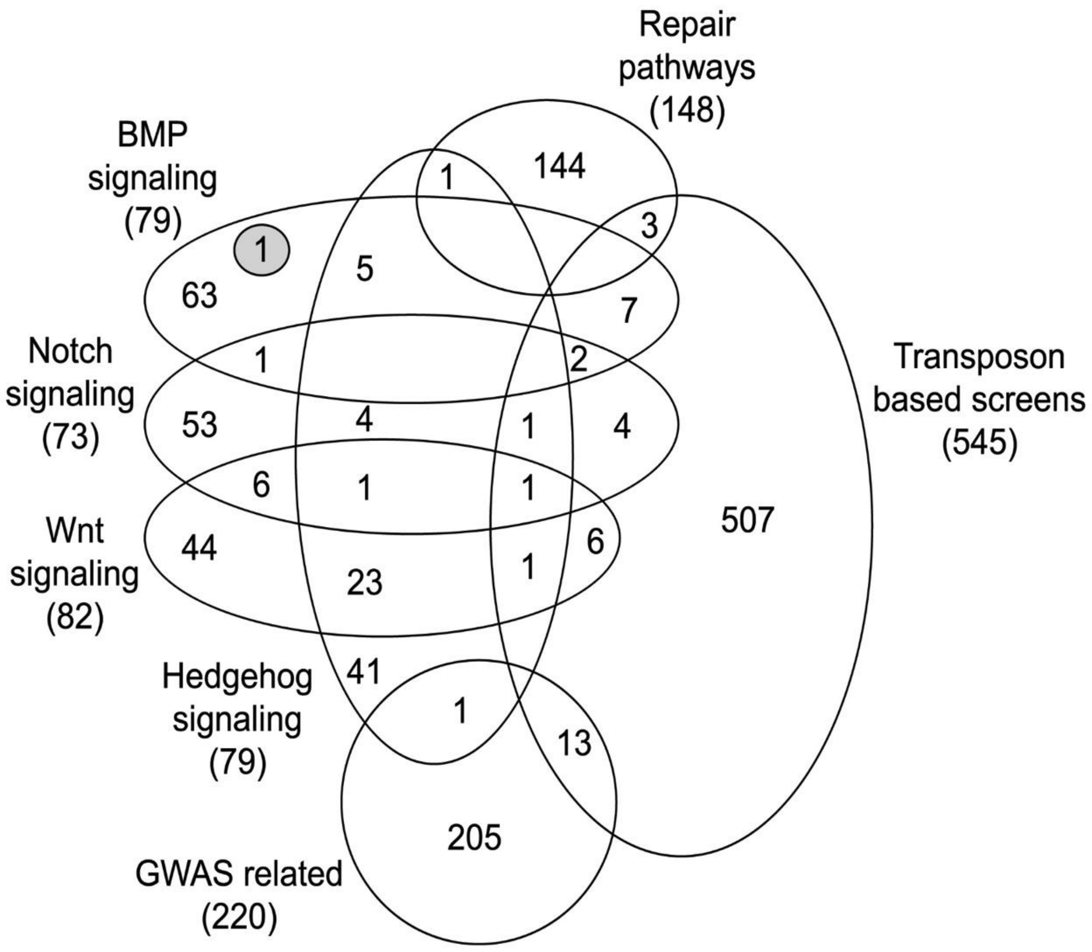


Figure 3.2. The 1,138 genes likely to play a role in colorectal tumourigenesis (Smith, Naven, et al. 2013), distributed by pathway or study. The shaded circle indicates a gene common to both the BMP signalling and GWAS related groups. Reproduced with permission; data were gathered and the figure was produced by Dr Christopher Smith.

The average depth of coverage across the 1,138 colorectal tumourigenesis genes was 35x. The percent of coding bases in the ORFs covered to ≥10x was

83.3%, and at ≥20x was 67.7%. Figure 3.3 (Smith, Naven, et al. 2013) graphically represents coverage statistics at ≥20x depth for the entire subset, with genes grouped by pathway or study. Average coverage of the ORFs at ≥20x depth was similar between groups.

From the 1,138 genes, and using our optimised parameters, 23 novel or rare nonsense mutations, 16 splice site mutations and 30 indels were predicted (Table 3.4). Sanger sequencing confirmed that 12 (52.2%) of the nonsense variants were present in the germline DNA. Examples of confirmed variants are shown in Figure 3.4. The remaining 11 (47.8%) nonsense variants could not be validated, and 8 of these were of low quality with quality by depth scores < 10. Figure 3.5 shows examples of variants that could not be validated.

Of the splice site mutations, 5 (31.3%) of these variants were true positives (3 were at splice acceptor sites and the other 2 were at splice donor sites), and all were likely to affect function as they were predicted to cause truncation of or skipping of substantial portions of the ORF. From the remaining splice site variants, eight (50%) could not be validated, 7 of which had low quality by depth scores <10 (one had a quality by depth score of 11.2). The final 3 (18.8%) splice site variants failed amplification and so could not be validated experimentally. Of the 30 rare or novel indels, 20 were validated. The remaining 10 (33.4%) were situated at loci with low sequence complexity; these loci often contained short repeats of 2-3 bases or homopolymer runs (>3 bases).

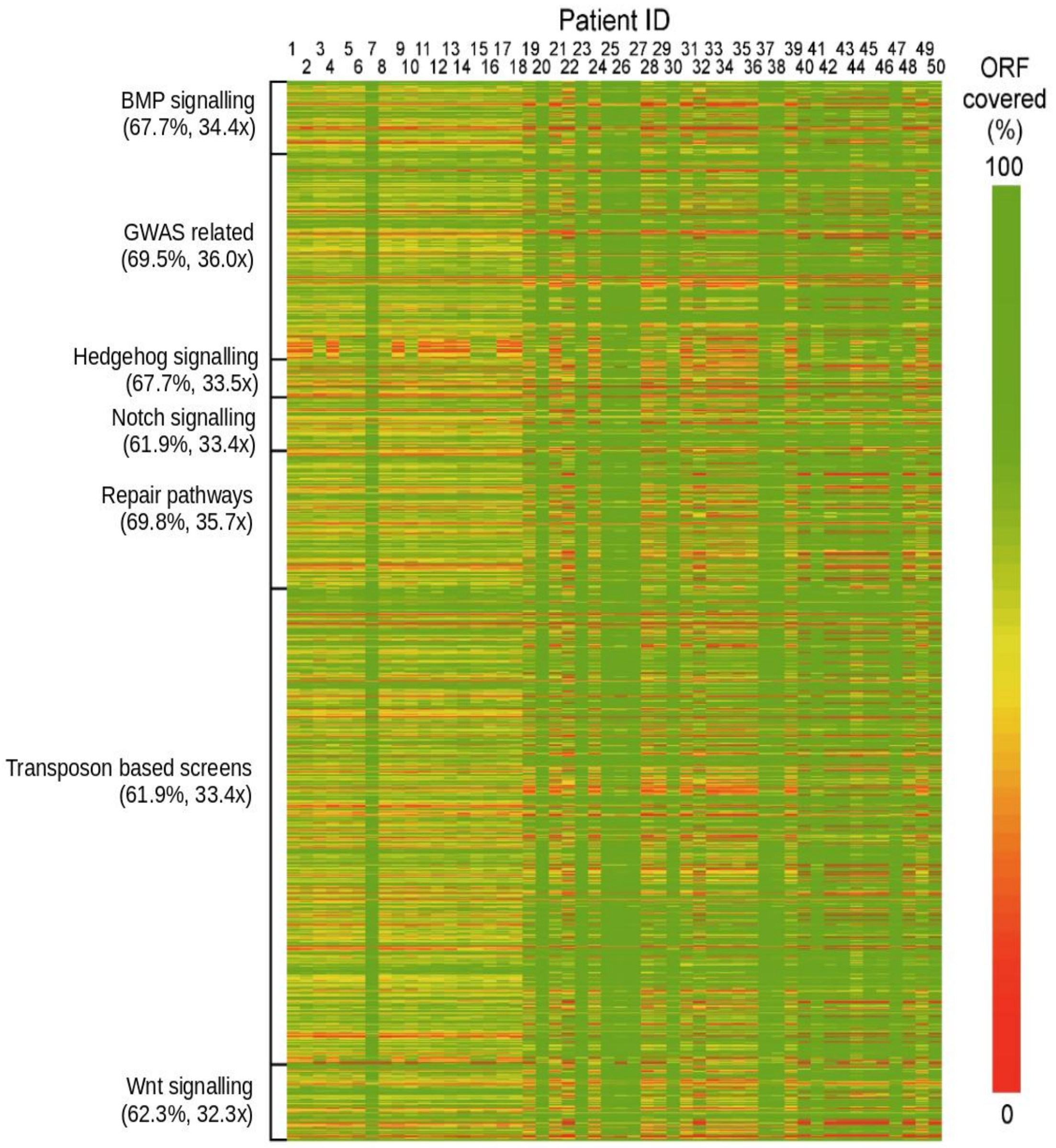


Figure 3.3. Percentages of the open reading frame covered to a depth of ≥20x of 1,138 genes likely to play a role in CRC tumourigenesis (Smith, Naven, et al. 2013). Genes are grouped by pathway or type of study. The average percent of the ORFs sequenced at ≥20x coverage, along with the average coverage, is shown for each group of genes. Reproduced with permission, with minor modification; data were generated by myself and the figure was produced by myself.

Table 3.4. Rare or novel truncating variants identified in 1,138 genes likely to play a role in colorectal tumourigenesis were predicted by our pipeline. The original DNA samples were then assayed via Sanger sequencing in order to validate the predictions.

	Variant type			**Total**
	Nonsense	**Splice site**	**Indel**	
Predicted	23	16	30	69
Validated	12	5	20	37
Validated (%)	52.2%	31.3%	66.7%	53.6%

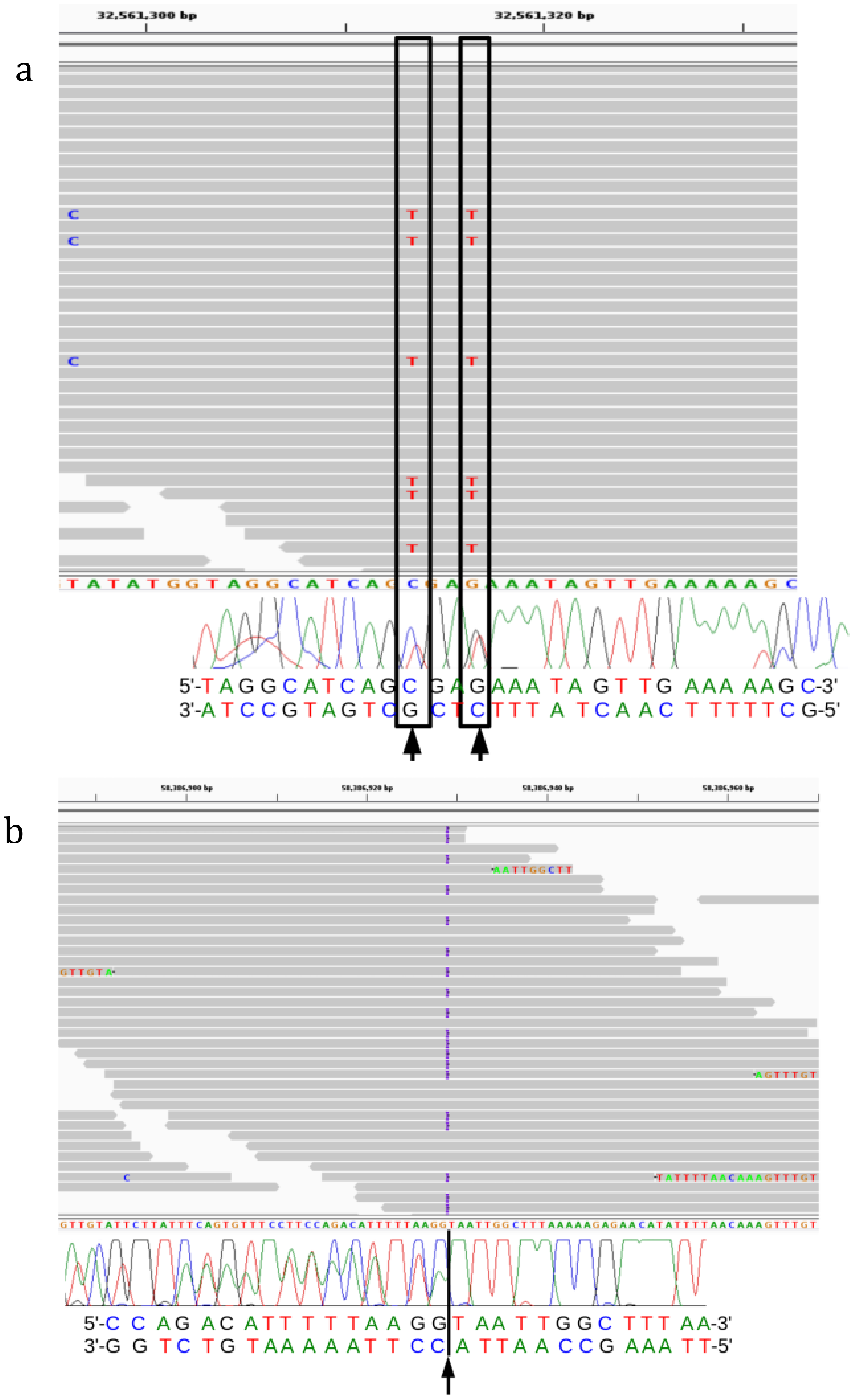

a
32,561,300 bp
32,561,320 bp
TATATGGTAGGCATCAGCGAGAAATAGTTGAAAAAGC
5'-TAGGCATCAGC GAGAAA TA GT TG AAA AAGC-3'
3'-ATCCGTAGTCG CTCTTT A TCAAC T TTTTCG-5'
b
58,386,900 bp
58,386,920 bp
58,386,940 bp
58,386,960 bp
GTTGTATTCTTATTTCAGTGTTTCCTTCCAGACATTTTTAAGGTAATTGGCTTTAAAAAGAGAACATATTTTAACAAAGTTTGT
5'-CC AG AC ATT T T TAA GGT AA T TGGC TTT AA-3'
3'-GG TC TG TAA A A ATT CCA TTAA CCG AAA TT-5'

Figure 3.4. Sanger sequencing confirmed the presence of a g.32561313C>T and a g.32561316G>T nonsense mutation carried by the same patient (a). The presence of a g.58386929_58386932dupTAAT insertion was also confirmed in a separate patient (b). The insertion site is marked in the chromatogram by a black arrow and vertical line, and in the NGS reads as a series of purple indicators.

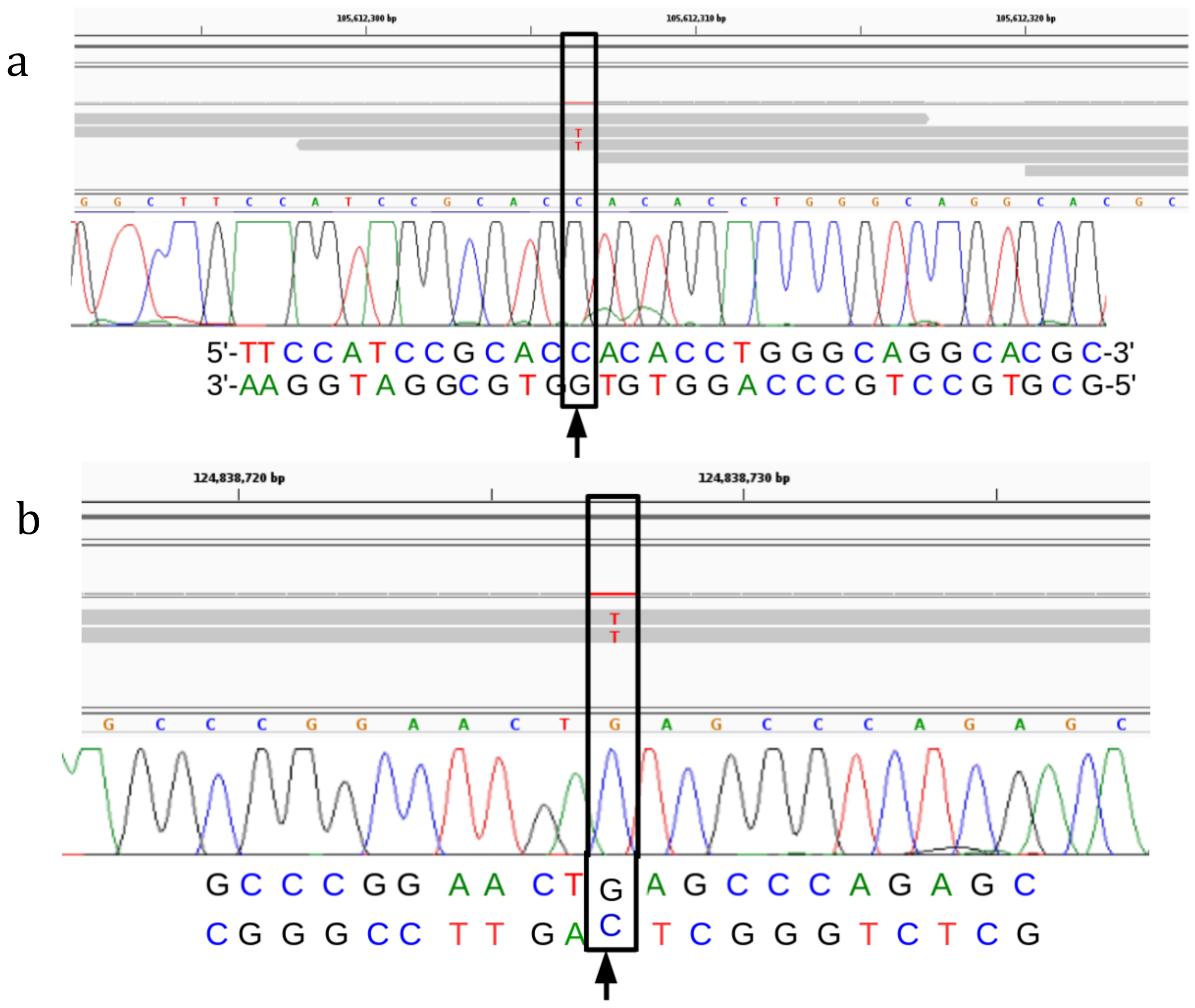


Figure 3.5. An NC_000014.8:g.105612306C>T mutation could not be validated by Sanger sequencing (a). Visual inspection of the NGS reads revealed the locus had low sequencing depth. A separate NC_000012.11:g.124838727G>T variant (b) could also not be detected via Sanger sequencing, and again the NGS data revealed low depth.

3.5 Discussion

To obtain a highly specific, reliable variant call set a bioinformatics pipeline was adopted and sets of parameters rigourously tested. Our pipeline utilised the publicly available BWA, SAMtools, GATK and SnpEff software, along with several in-house Perl scripts used for data parsing and custom analysis. For sensitive detection of germline variants, we recommend mapping reads using BWA's backtrack algorithm (Li & Durbin 2009). Mapping of the sequencing reads with BWA worked well, with 97.3% of all reads mapped to release GRCh37 of the human genome (96.2% of reads were mapped in the correct mate pair).

Due to the wide array of tools available, major differences in variant calling pipelines are usually found at the alignment phase. Each tool generally works in a unique way; SOAP, as used for BGI for example, can trim the 3-‘ end of reads in order to avoid the problem of decreasing quality towards the end of the read (Fonseca et al. 2012). We aligned reads with BWA, which doesn't trim reads, and have shown that calling variants from bases with lower quality scores results in >3 fold more variants than called by BGI. Mapping tools used are therefore a good candidate for the observed differences in number of variants predicted.

For exome sequencing data, removing PCR duplicates using SAMtools is advised prior to realignment of reads around known indels. Removal of duplicates will help to lower false positive variant calls that arise due to PCR artefacts. Local realignment of reads around know indels is advantageous as is base quality score recalibration, performed with GATK, and so these steps are also recommended (Li 2014; DePristo et al. 2011).

For germline variant calling, we found that lowering the minimum base quality required to include a base in variant calling allowed for much more overlap with BGI's pipeline. However, many more variants were called that were not in BGI's call set. Using the lower base quality score parameter we only increased our concordance with known genotypes by 1 call out of >6,000. The effort required to filter false positives therefore outweighs the benefit of obtaining a very minimal increase in true positive rate.

The combined average depth of coverage across our exomes averaged 30.2x, which provided enough data for reliable variant calling. This was proven by our comparisons with previously obtained genotyping data, which showed a 99.8% concordance rate and was consistent across parameter settings. The majority of the differences were primarily due to a sequencing depth of ≤8x. These factors suggest our pipeline has excellent capabilities of calling variants accurately across samples sequenced on multiple sequencers, and processed using different enrichment kits.

The comparisons between data from our pipeline and BGI's pipeline show that we were able to call the majority of variants called by BGI. We successfully found 93.4% and 94.1% of BGI's SNPs and indels, respectively, using the default parameters. However, when looking at the rare, potentially pathogenic variants, we only found 59.0% and 87.5% of their SNPs and indels respectively. We therefore modified parameters of our variant calling step and increased the overlap with BGI's novel SNPs and novel indels to 63.3% and 88.9%, respectively. While this also increased the average number of variants we called per exome by around 93,000, we anticipated many of these extra calls being false positives and so allowed variants with low quality scores to be marked for later filtering. The false positive rates were considered when attempting to validate predicted variants *in vitro*.

From the analysis of 1,138 genes likely to be involved in colorectal tumourigenesis, we identified 69 variants which were novel or had a MAF ≤1% and we were able to validate 37 (53.6%) of these by experimentally re-assaying the original gDNA used for exome sequencing. Interestingly, we were able to validate 66.7% of predicted indels, which are notoriously difficult to call accurately. Other studies have attempted to validate variants called from NGS data using Sanger sequencing and shown that multiple factors affect validation rates. Park et al. (2014) conclude that around 66% of variants called by GATK could be validated, and variants are more likely to be validated if they have both high read depth and high quality scores. Other factors, such as higher mapping quality and performing read realignment positively impacted upon validation rates (Park et al. 2014). Other work (Zhang et al. 2015) has shown that sequencing platform and analysis pipeline effect validation rates; calling SNPs using GATK from paired-end exome data after mapping with BWA's paired-end or single-end mode gave respective validation rates of 90% and 86.6%. Exome capture methods and sequencer have also been considered; exomes sequenced using Agilent's SureSelect Human All Exon kit and Illumina's HiSeq 2000 gave an 88.3% SNP validation rate, whereas capturing exomes using the Ion TargetSeq and sequencing on the Ion Proton sequencer gave a 60% validation rate. 89.6% and 15.8% of indels called from HiSeq and Ion Proton data were validated, respectively (Zhang et al. 2015).

Together, our work shows that our approach using NGS and bioinformatic analyses are acceptable for detecting common polymorphisms as well as rare or novel potentially disease causing alleles.

Chapter 4 - Development of next generation sequencing protocols to identify somatic mutations in formalin-fixed paraffin-embedded tumours

4.1 Details of work performed

Sectioning of FFPE blocks, slide staining and DNA extraction was performed by myself and Dr Christopher Smith. Library preparation and sequencing of all samples was performed by myself, Dr Christopher Smith and Shelley Rundle. Bioinformatics and downstream analyses was performed by myself.

4.2 Introduction

Comprehensive and accurate profiling of a patient’s tumour is essential for stratification of targeted therapies. For example, patients carrying *KRAS* or *NRAS* mutations fail to respond to cetuximab (an anti-EGFR antibody) used in the treatment of advanced CRC (aCRC) (Karapetis et al. 2008; De Roock, Claes, et al. 2010). Similarly, in non-small-cell lung cancer, response to the tyrosine kinase inhibitor gefitinib (which targets EGFR's tyrosine kinase domain) is increased in patients harbouring particular somatic EGFR mutations (Lynch et al. 2004). Many of these mutations cause single amino acid substitutions or small in-frame deletions close to the ATP-binding region of EGFR's tyrosine kinase domain; this leads to not only increased sensitivity to gefitinib but also increased EGFR signalling (Lynch et al. 2004).

The majority of tumour samples available for molecular analyses are from FFPE tissues. Cellular DNA is normally damaged during processing causing it to be fragmented (Wong et al. 2013) and contain formaldehyde induced cross-links (Fraenkel-Conrat & Olcott 1948), abasic sites (Zsikla et al, 2004) and deaminated cytosines (Chen 2014) (Chapter 1, Section 1.5.2). These lesions reduce the quality and quantity of DNA available for subsequent amplification. Furthermore, contamination with normal stroma is often unavoidable (Gerlinger et al. 2012), resulting in low levels of mutant alleles. Therefore, molecular profiling of DNA from FFPE tumours is technically challenging.

Mutation analysis of FFPE derived DNA has, until recently, largely depended upon the use of technologies that interrogate only a small genomic region or number of ‘hotspot’ mutations (e.g. Sanger sequencing and pyrosequencing). However, the advent of high-throughput NGS has seen a paradigm shift with researchers taking advantage of the increased genomic coverage and higher sensitivity (Voelkerding et al. 2009). This means that NGS work flows and variant calling pipelines have to deliver high quality results for routine use in patient stratification.

Here, we tested NGS of DNA from FFPE material to identify previously characterised somatic mutations in patients with aCRC and to develop robust strategies for the identification of novel somatic mutations amongst the background of FFPE-associated artefacts.

4.3 Materials and methods

4.3.1 Patients and samples

Patients were from the MRC trials COIN and COIN-B for the treatment of aCRC (Chapter 2, Section 2.2). The somatic mutation status of *KRAS* (codons 12, 13 and 61), *BRAF* (codons 594 and 600), *NRAS* (codons 12 and 61) and *PIK3CA* (codons 542, 545, 546 and 1047) was previously determined in tumour samples from these patients using pyrosequencing (QIAGEN) and MALDI-TOF MS (Sequenom) technologies (Maughan et al. 2011; Smith et al. 2013) (Chapter 2, Section 2.6). Here, we analysed germline blood DNA (gDNA), and DNA from nFFPE colon and tFFPE CRCs from 19 patients that carried (Table 4.1) and 19 that did not carry such mutations.

Table 4.1 Previously determined somatic mutations in *KRAS*, *NRAS*, *BRAF* and *PIK3CA* from 19 CRC patients (Smith et al. 2013).

Patient ID	**Previously determined mutations**			
	KRAS	***BRAF***	***NRAS***	***PIK3CA***
19	WT	WT	WT	p.E545K
51	p.G12D	WT	WT	WT
52	p.G12V	WT	WT	WT
53	WT	p.V600E	WT	WT
54	p.G13D	WT	WT	WT
55	p.G12V	WT	WT	p.H1047R
56	p.G12D	WT	WT	WT
57	p.G12D	WT	WT	p.E545K,
58	WT	WT	p.Q61K	p.H1047R
59	p.G12D	WT	WT	p.E545K
60	p.G12D	WT	WT	p.E545K
61	p.G13D	WT	WT	WT
62	p.G13D	WT	WT	WT
63	p.G13D	WT	WT	WT
64	p.G13D	WT	WT	WT
65	p.G12V	WT	WT	p.E545K
66	WT	p.D594G	WT	WT
67	WT	WT	WT	p.H1047R
68	WT	WT	WT	p.E545K

gDNA was extracted from blood samples as previously described (Maughan et al. 2011). DNA extraction from tFFPE and nFFPE specimens was carried out as described in Chapter 2, Section 2.4.

4.3.2 UDG treatment

A minimum of 250 ng of tFFPE derived DNA was treated with 10U of UDG as described (Chapter 2, Section 2.7).

4.3.3 TruSeq Custom Amplicon (TSCA) design, library preparation and sequencing

Custom amplicons spanning the open reading frames of *KRAS*, *BRAF*, *NRAS* and *PIK3CA* were designed using Illumina's DesignStudio software (Chapter 2, Section 2.8.2.1). Where possible, oligonucleotide probes were designed to regions free of known common polymorphisms. Library preparation and NGS was carried out as described (Chapter 2, Sections 2.8.2.2 – 2.8.2.4).

4.3.4 Bioinformatics and variant calling

Sequencing data were processed through our variant calling pipeline (Chapter 2, Section 2.9, Smith et al. 2013). Briefly, reads were aligned to the reference human genome (GRCh37) using BWA's (Li & Durbin 2009) backtrack algorithm, SAMtools (H. Li et al. 2009) was used to convert files to BAM format, to which paired end information was added (Chapter 2, Section 2.9.2). SAMtools was also used to sort reads by genomic coordinates. GATK's RealignerTargetCreator (McKenna et al. 2010) identified reads mapping to regions containing known indels and IndelRealigner realigned those reads. GATK's BaseRecalibrator recalibrated base quality scores across all reads (Chapter 2, Section 2.9.2). Variant calling was performed for each sample using GATK's HaplotypeCaller. The MiSeq Somatic Variant Caller software (on board

MiSeq 1) was also used to call variants for all samples sequenced on MiSeq 1. SnpEff (Cingolani et al. 2012) and GATK's VariantAnnotator were used to annotate the variants in the resulting VCF files (Chapter 2, Section 2.4). All software up to this point were used with their default parameters. Custom Perl scripts were written to facilitate all bespoke analyses and data parsing (Chapter 2, Section 2.9).

4.4 Results

We initially analysed blood gDNA from 19 patients with aCRC together with DNA from their normal healthy colon and tumour tissue that had undergone FFPE treatment (nFFPE and tFFPE, respectively). These patients' tumours carried a total of 25 somatic mutations in *KRAS, BRAF, NRAS* and *PIK3CA*, as previously determined by Pyrosequencing or Sequenom (Smith et al. 2013). Thirteen CRCs h*ad KRA*S mutations (5x p.G12D, 3x p.G12V, 5x p.G13D), two had *BRAF* mutations (1x p.D594G, 1x p.V600E), one had an *NRA*S mutation (p.Q61K) and nine had *PIK3C*A mutations (6x p.E545K, 3x p.H1047R). We carried out NGS of these samples on a custom TSCA panel targeting the ORF of *KRAS, BRAF, NRAS* and *PIK3CA* and analysed the libraries on two MiSeq sequencers (with 150 bp [MiSeq 2] and 175 bp [MiSeq 1] read length settings). Output metrics for MiSeq 1 and MiSeq 2, respectively, were as follows; 7.5 Mb vs. 6.2 Mb of sequencing data produced, average base quality 32.9 vs. 32.1 for gDNA, 33.5 vs. 33.1 for nFFPE, and 33.7 vs. 32.8 for tFFPE, and mean sequencing depth 4288x vs. 3666.3x for gDNA, 2913.5x vs. 2419.2x for nFFPE, and 3174x vs. 2769.8x for tFFPE (Tables 4.2 and 4.3). There was a higher mean coverage for gDNA as compared to DNA from FFPE processed samples.

4.4.1 Mapping reads

As expected, average read lengths from MiSeq 1 and MiSeq 2 were different (176 bp and 151 bp, respectively; Table 4.2) due to the different cycle settings used initially. Although both sequencers produced reads with mapping scores >30 (the threshold at which 1 in 1000 reads is likely to be mapped incorrectly; Table 4.4), reads mapping to *PIK3CA* or *BRAF* had considerably lower mapping scores from MiSeq 1 than from MiSeq 2 (Figure 4.1a). In addition, more reads from MiSeq 2 mapped in the correct paired-end style than those from MiSeq 1 (Figure 4.1b, Table 4.5). There were no notable differences between tissues, suggesting that the problem was not FFPE related. Using Integrative Genomics Viewer (IGV) software (Thorvaldsdóttir et al. 2013; Robinson et al. 2011) we found numerous reads sequenced on MiSeq 1 were labelled as incorrectly mapped and contained 3' overhangs that did not match the reference sequence (Figure 4.2a).

Table 4.2. Sequencing metrics for runs on two MiSeqs using the same libraries prepared from 19 patients with known somatic mutations.

* We used 500 (MiSeq1) or 300 (MiSeq2) cycle reagent cartridges.

Sequencing run	MiSeq 1			MiSeq 2		
Tissue type	gDNA	nFFPE	tFFPE	gDNA	nFFPE	tFFPE
Mb sequenced	3.3	2.2	2.0	2.6	1.7	1.9
No. reads	18724	12745	11560	17002	11545	12479
Read length*	176	176	176	151	151	151
Base quality	32.9	33.5	33.7	32.1	33.1	32.8
Average depth of targets	4288	2913.5	3174	3666.3	2419.2	2769.8
% ORF at 1000x coverage of						
KRAS	94.0	70.0	80.9	92.6	69.7	68.2
BRAF	86.5	70.4	80.9	90.5	66.6	78.0
NRAS	76.2	79.8	79.0	83.5	72.1	79.6
PIK3CA	95.0	88.2	91.8	94.2	82.2	87.7

Table 4.3. Average depth of coverage (by individual sample on MiSeq1). FFPE tissues typically had lower depths of coverage than their gDNA counterparts. The five samples with high levels of C:G>T:A artefacts (shaded) did not show higher than average sequencing depths.

Sample	gDNA average	nFFPE average	tFFPE average
67	4,395.2	1,371.6	4,087.5
51	3,998.5	3,364.0	2,731.1
68	4,969.7	2,884.9	3,864.5
52	4,527.0	3,491.5	4,060.9
53	4,010.2	4,099.9	4,471.9
54	4,110.2	3,596.0	2,981.0
55	4,314.3	1,244.6	1,195.7
56	4,222.3	3,529.6	3,651.7
19	3,492.4	3,031.7	3,816.7
57	4,034.5	1,229.8	4,067.4
58	4,358.5	2,095.9	3,688.0
59	4,022.7	4,535.6	4,491.9
60	4,608.8	4,697.0	2,536.9
61	3,700.3	4,115.3	3,381.8
62	4,515.4	4,912.6	4,395.2
63	4,787.1	1,372.6	3,429.4
64	4,931.9	1,758.3	4,251.9
65	4,738.1	2,981.9	1,598.6
66	4,847.3	4,174.5	2,968.3
Average	**4,288.0**	**2,913.5**	**3,174.0**

Table 4.4. Mapping qualities across each of the target genes from MiSeq 1 and MiSeq 2.

MiSeq (Read length)	**Tissue**	**Average mapping quality (by gene)**			
		KRAS	***NRAS***	***BRAF***	***PIK3CA***
1 (176bp)	Blood	58.1	52.9	48.3	41.4
	nFFPE	57.7	52.0	47.0	39.4
	tFFPE	58.3	53.1	47.3	40.0
	Average	**58.1**	**52.7**	**47.5**	**40.3**
2 (151bp)	Blood	59.5	59.6	59.5	57.6
	nFFPE	59.5	59.6	59.5	56.7
	tFFPE	59.5	59.6	59.6	57.5
	Average	**59.5**	**59.6**	**59.6**	**57.3**

Table 4.5. Correct mapping of paired-end reads from MiSeq 1 and MiSeq 2.

MiSeq (Read length)	**Tissue**	**Reads correctly mapped as a pair (%)**			
		KRAS	***NRAS***	***BRAF***	***PIK3CA***
1 (176bp)	Blood	93.0	55.6	55.4	26.8
	nFFPE	92.2	54.3	54.7	21.3
	tFFPE	93.8	59.3	51.6	20.3
	Average	**93.0**	**56.4**	**53.9**	**22.8**
2 (151bp)	Blood	99.8	99.7	99.6	99.3
	nFFPE	99.8	99.7	99.6	99.4
	tFFPE	99.8	99.7	99.7	99.5
	Average	**99.8**	**99.7**	**99.6**	**99.4**

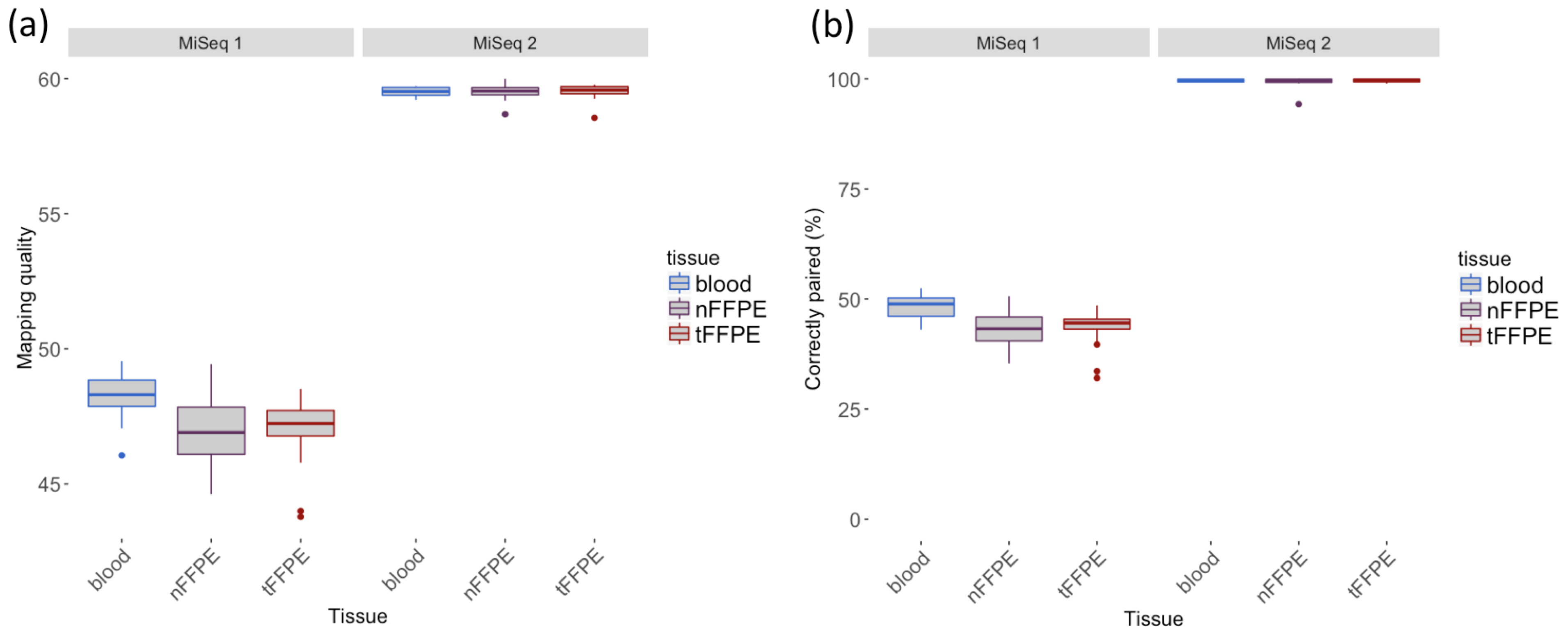


Figure 4.1. (a) Mapping qualities for all tissues on MiSeq 1 compared to MiSeq 2. (b) The amount of reads mapping as a correct pair (%) on MiSeq 1 in comparison to MiSeq 2.

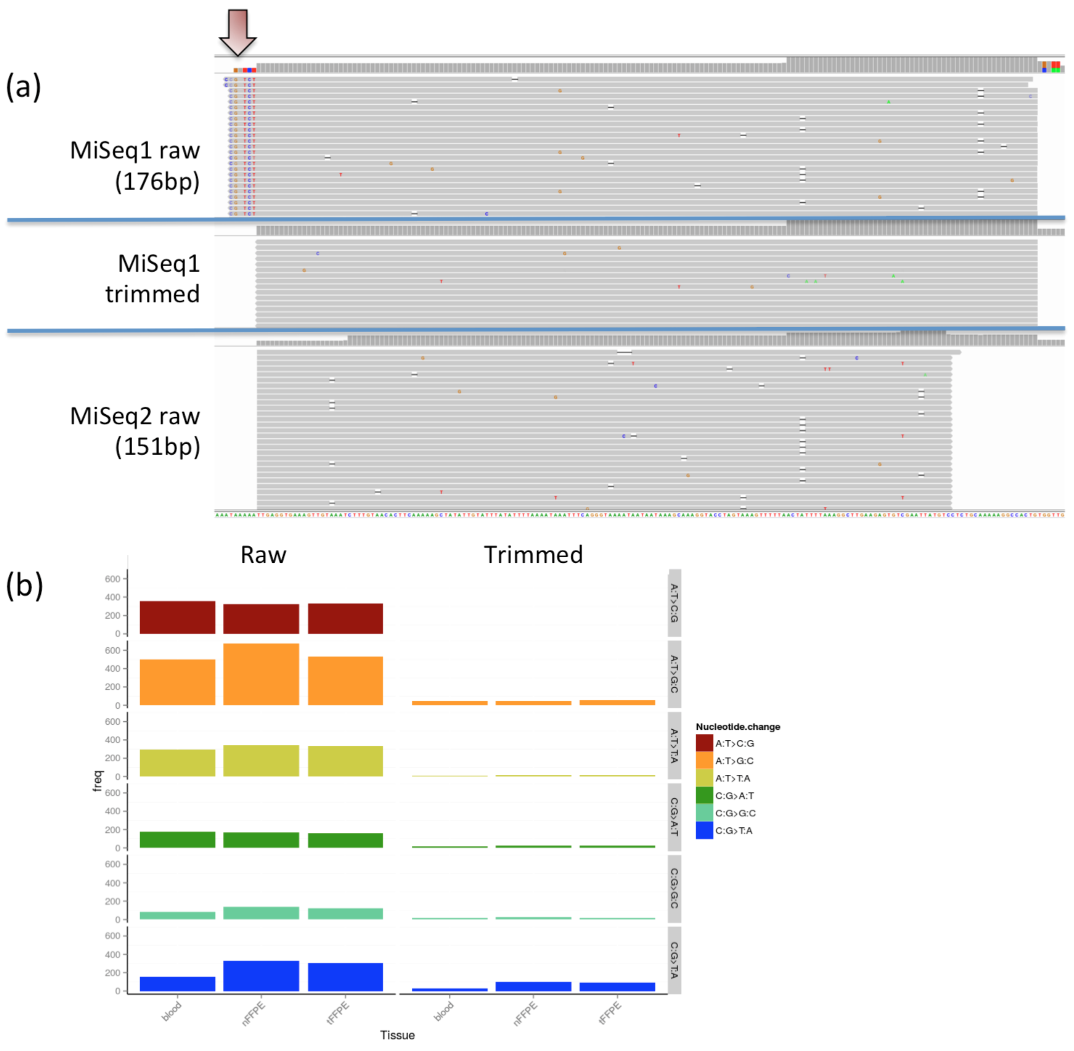


Figure 4.2. (a) Visual inspection using IGV revealed reads from MiSeq 1 contained adapter sequence at read ends (top panel, red arrow). Trimming the adapters was performed in MiSeq 1 data (middle panel). MiSeq 2 data used fewer sequencing cycles and no adapters were seen. (b) The amount of variants called from MiSeq 1 data pre- (left) and post- (right) trimming.

These mismatching sequences were common to several amplicons and were found to match partial adaptor sequences. Using Trimmomatic (Bolger et al. 2014), these sequencing adapters were accurately removed (Figure 4.2a) prior to further bioinformatic analyses. Removal of adaptors resulted in fewer reads with soft-clipped (unaligned) bases and decreased variants (Figures 4.2b).

4.4.2 Identifying known mutations

We assessed the MiSeq Somatic Variant Caller and GATK's HaplotypeCaller to identify the known somatic mutations. MiSeq's Somatic Variant Caller successfully identified all 25 (100%) somatic mutations whereas HaplotypeCaller identified 20 (80%) of known variants using data from MiSeq 1 and 23 (92%) from MiSeq 2 (Table 4.6). The variants consistently missed by HaplotypeCaller were p.E545K (both mutations missed on MiSeq 2 and 2/5 missed on MiSeq 1) and p.H1047R in *PIK3CA* (3/5 mutations missed on MiSeq 1). In all of these cases, there was >1000 fold depth of coverage. However, visual inspection of sequence data using IGV suggested that the frequency of the mutant alleles might be too low to be recognised above the background noise; the p.E545K mutations were only seen in 2-3% of mapped reads from both MiSeqs. We also noted that for p.H1047R, the mutation mapped to a probe region, potentially resulting in a strand bias towards templates containing the wild type allele (Guo et al. 2012).

Table 4.6. Identification of somatic mutations (n = 25) using MiSeq Somatic Variant Caller and GATK's HaplotypeCaller using NGS data from 19 patients with aCRC.

Platform	MiSeq 1			MiSeq 1			MiSeq 2		
Caller	MiSeq Somatic Variant Caller			HaplotypeCaller			HaplotypeCaller		
Tissue	gDNA	nFFPE	tFFPE	gDNA	nFFPE	tFFPE	gDNA	nFFPE	tFFPE
Known Mutations	0/25	0/25	25/25	0/25	0/25	20/25	0/25	0/25	23/25
Sensitivity	-	-	100%	-	-	80%	-	-	92%
Other Hot-Spot?	0	10	18	0	0	0	0	0	0
Specificity	-	95.2%	91.4%	-	-	-	-	-	-

Twenty-eight false positives were predicted at the hotspot sites (*KRAS* codons 12, 13 and 61, *BRAF* codons 594 and 600, *NRAS* codons 12 and 61 and *PIK3CA* codons 542, 545, 546 and 1047) by MiSeq's Somatic Variant Caller (18 in tFFPE and 10 in nFFPE; 93.3% specificity) (Table 4.6) whereas no false positives were found by HaplotypeCaller in data from both MiSeqs (100% specificity).

4.4.3 Investigating the background mutation profile

We investigated the background mutation profile associated with the use of DNA from FFPE material by comparing variants called by HaplotypeCaller across the entire targeted *KRAS*, *BRAF*, *NRAS* and *PIK3CA* loci using gDNA and matched nFFPE DNA samples (data from MiSeq 1 & 2 combined). A three-fold increase in SNVs was predicted in nFFPE DNA compared to gDNA (735 vs. 216), whereas only a small number of indels were predicted (14 vs. 7). In terms of type of SNVs, we observed the following for nFFPE DNA vs. gDNA (Figure 4.3) respectively (expressed as a percent of all nFFPE DNA and gDNA SNV calls); A:T>T:A (6.9 vs. 1.5), C:G>A:T (4.2 vs. 2.3), C:G>G:C (9.5 vs. 3.0), and A:T>C:G (1.7 vs. 0.0). However, the most notable difference was a significant increase in C:G>T:A substitutions in FFPE tissue (27.1 vs. 5.3) and in A:T>G:C (27.9 vs. 10.6), with most variants being predicted with high confidence. The increase in C:G>T:A’s was primarily noted in five samples (Figure 4.4) and these samples had similar sequencing coverage compared to the others (Table 4.3).

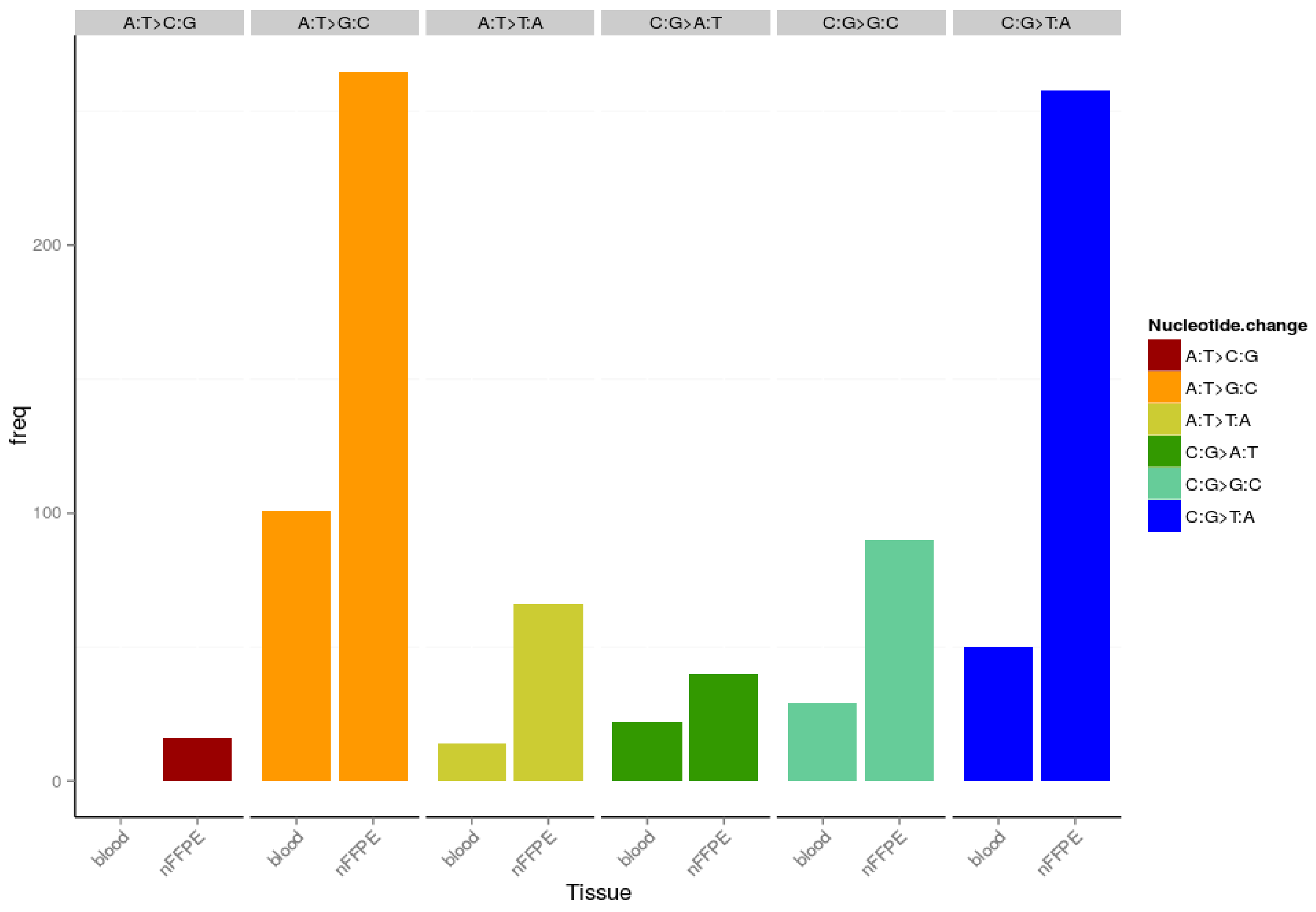


Figure 4.3. Background mutation profile associated with the use of DNA from FFPE material. A large increase in A:T>G:C and C:G>T:A transitions was observed in nFFPE.

We studied the effect of pre-treating DNA samples with UDG by analysing tFFPE DNA from a further 19 patients whose CRCs were wild type at the mutation hotspots in *KRAS*, *BRAF*, *NRAS* and *PIK3CA*. UDG treatment reduced the total number of SNVs (treated vs. Untreated; 181 vs. 210) predicted across the entire *KRAS*, *BRAF*, *NRAS* and *PIK3CA* loci in these samples.

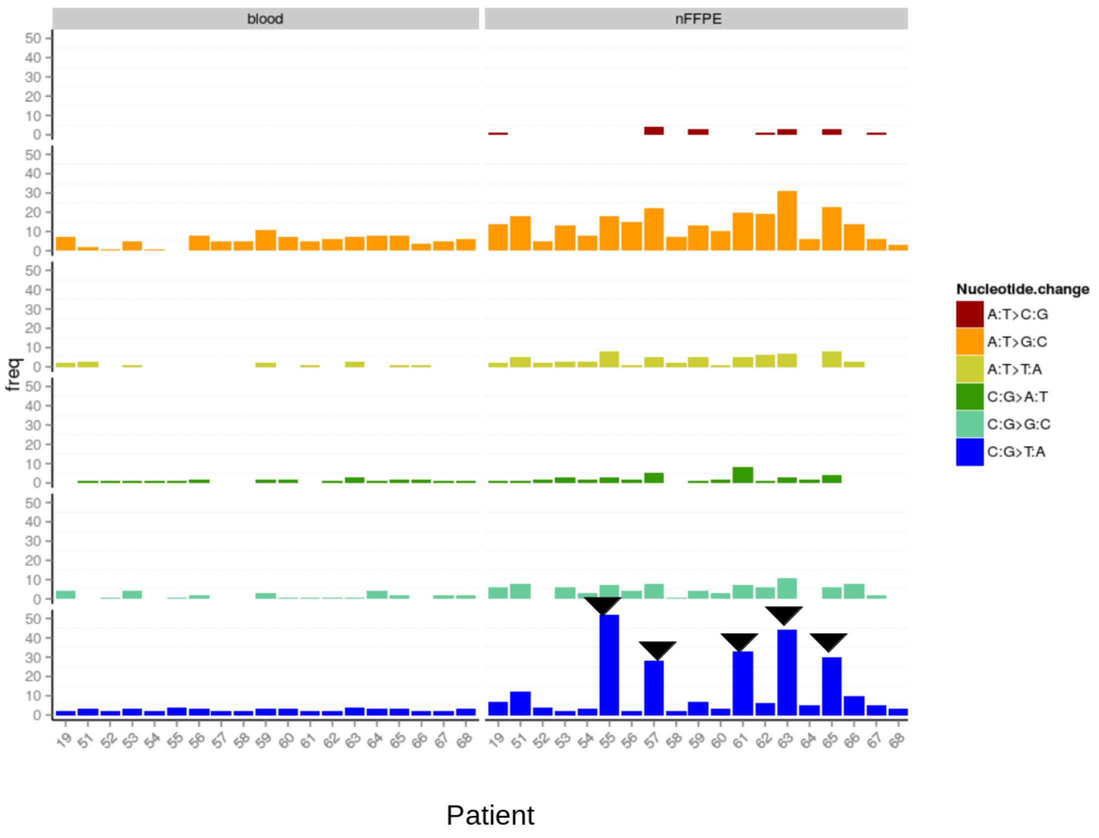


Figure 4.4. The increase in C:G>T:A's artefacts was particularly noted in five samples (arrowheads).

It had little effect on the predicted numbers of (percent of SNVs called with and without treatment) A:T>C:G (0.5 vs. 0.5), A:T>G:C (18.9 vs. 19.9), A:T>T:A (7.7 vs. 7.4), C:G>A:T (3.3 vs. 2.0) and C:G>G:C (0.5 vs. 0.5) substitutions. In contrast, UDG treatment significantly reduced the predicted number of C:G>T:A substitutions (15.3 vs. 23.0) (Figure 4.5, Table 4.7). It was again noted that seven untreated DNA samples were responsible for the excess of C:G>T:A substitutions; these were dramatically reduced after treatment (Figure 4.6).

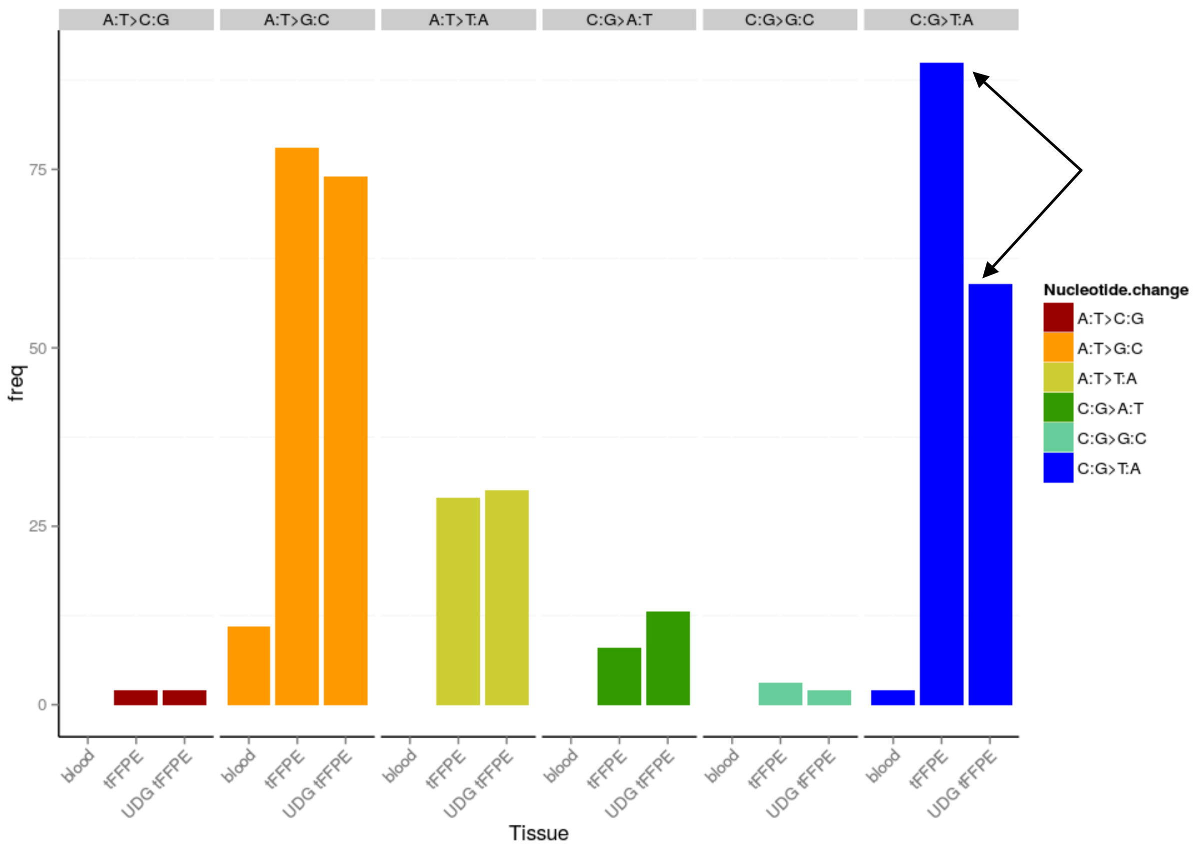


Figure 4.5. UDG treatment of DNA from tFFPE samples (UDG tFFPE), reduced the frequency of C:G>T:A substitutions (black arrows).

Table 4.7. Breakdown of predicted SNVs and indels across the entire KRAS, BRAF, NRAS and PIK3CA loci in 19 patients without hotspot mutations.

Tissue	gDNA	tFFPE – UDG (%)	tFFPE +UDG (%)	In Both +/- UDG	Somatic
Total SNVs	13	210	181	22	3
A:T>C:G	0	2 (0.5)	2 (0.5)	0	0
A:T>G:C	11	78 (19.9)	74 (18.9)	10	0
A:T>T:A	0	29 (7.4)	30 (7.7)	1	0
C:G>A:T	0	8 (2.0)	13 (3.3)	0	0
C:G>G:C	0	3 (0.5)	2 (0.5)	0	0

C:G>T:A	2	90 (23.0)	60 (15.3)	11	3
Total indels	0	18	10	0	0
Insertions	0	1	2	0	0
Deletions	0	17	8	0	0

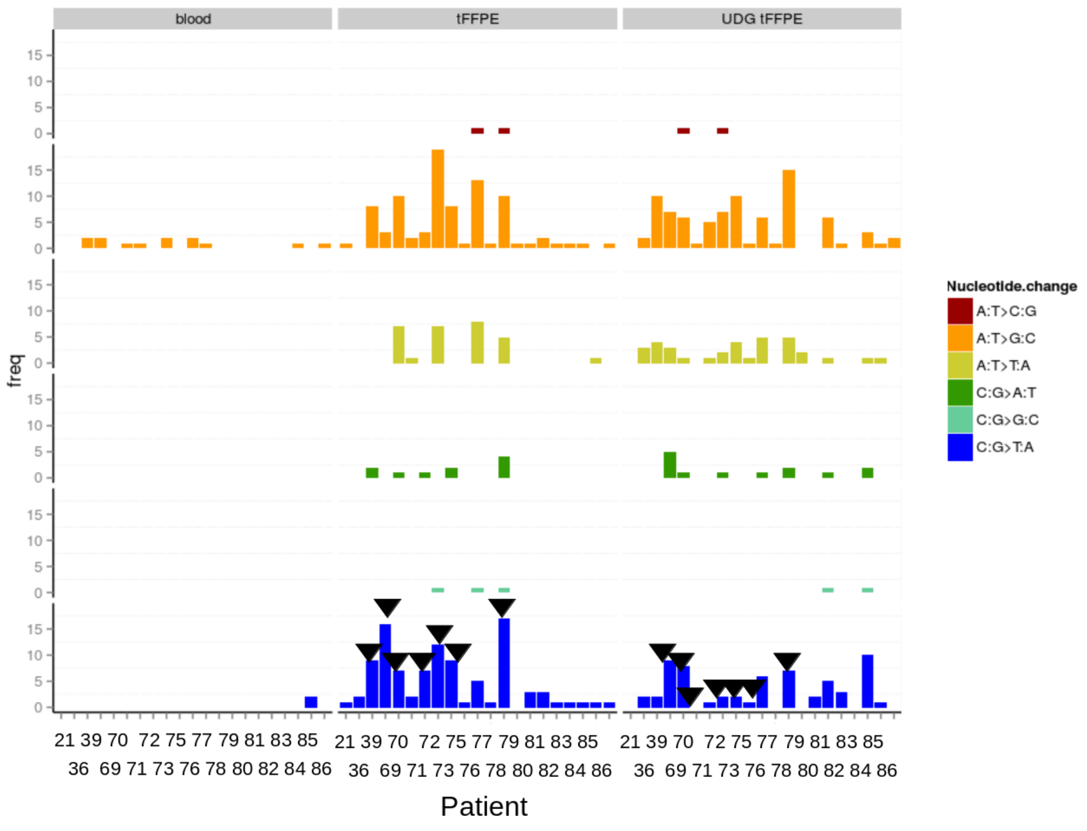


Figure 4.6. Analysis of an independent set of tFFPE DNA samples showed that seven had high proportions of C:G>T:A mutations prior to UDG treatment (arrowheads). These artefacts were removed or reduced (arrowheads) after UDG treatment (UDG tFFPE).

Given the large number of variants predicted across all four loci studied, we considered strategies to facilitate the identification of genuine somatic variants over the background of FFPE-associated artefacts. UDG treatment significantly reduced the predicted number of C:G>T:A substitutions. We considered variants more likely to be genuine if they were consistently found in both the UDG treated and untreated runs (column 'In Both +/- UDG'). Only 22 such SNVs were consistently predicted and, of these, 16 were present in the germline. Three SNVs had little or no sequencing coverage in the gDNA and could not be confirmed as somatic. The remaining 3 SNVs were proven to be somatic. These variants included a p.R357Q and p.E545K in *PIK3CA*, as well as a p.S430L in *BRAF*. For tFFPE +/- UDG columns, the percent of SNVs in each group are shown out of all SNVs detected are shown (Table 4.7).

4.5 Discussion

4.5.1 Presence of adapter sequences

We found that 176 bp reads showed lower mapping qualities compared to 151 bp reads, particularly for the larger genes on our panel (e.g. *PIK3CA*). We attributed this to the introduction of partial adapter sequences in the longer reads. The amplicons in our project were designed to be on average 175 bp in length; however, for shorter amplicons, when the number of sequencing cycles exceeded the amplicon length, partial adapter sequences were introduced at the 3' ends of the reads. This explains why only particular amplicons exhibited large numbers of incorrectly paired reads. This problem was efficiently resolved using Trimmomatic, which successfully removed the adaptors; this resulted in fewer unaligned (soft-clipped) bases at the ends of reads, along with fewer variant calls (Figure 4.2b). Using read lengths that are shorter than the intended amplicon length bypasses the problem of sequencing into adapters.

4.5.2 Somatic variant calling in FFPE DNA

Schweiger et al. (2009) were amongst the first to apply NGS to the analysis of FFPE DNA, and demonstrated a correlation between fresh frozen and FFPE DNA using whole genome and exome-sequencing. They found that longer fixation times (>72 hours) or time spent between surgical resection and fixation do not negatively affect sequencing efforts; however, this contrasts with other researchers findings (Chapter 1 Section 1.5.3). FFPE samples gave sequencing yields comparable to fresh-frozen samples. Increased storage time of samples (sometimes over 18 years) did not automatically exclude FFPE samples from use in NGS experiments. FFPE induced DNA damage may, however, result in genomic mismatches, but concordance rates of variant calls from fresh-frozen and FFPE samples can still reach 95% (Schweiger et al. 2009). Numerous studies have since utilised NGS, and suggest an ability to detect mutations down to 1-4% mutant allele levels (Bourgon et al. 2014) with >95% sensitivity and specificity (Harismendy et al. 2011; Hadd et al. 2013; Altimari et al. 2013; Ihle et al. 2014). Here we carried out NGS using Illumina’s MiSeq platform and tested two variant callers for detecting known hotspot mutations: MiSeq's Somatic Variant Caller had 100% sensitivity but lower specificity whereas HaplotypeCaller had 80-92% sensitivity and 100% specificity.

HaplotypeCaller is designed for use with germline samples and may not detect variants at low frequency in a given sample (McKenna et al. 2010). This may explain why the sensitivity to detect known variants was lower than for the MiSeq's Somatic Variant Caller. The Somatic Variant Caller detected all of the expected known variants, plus a further eighteen and ten variants at hotspots in tumour and normal FFPE samples respectively. While the frequency of these additional variants may be too low for detection with germline callers or earlier sequencing technologies, we did not expect any to be seen in the normal FFPE. It is possible that Somatic Variant Caller is calling sequencing artefacts as somatic variants (Qiu et al. 2016), or that there is contamination between normal stroma and tumour tissue (Elloumi et al. 2011). A further possibility is that erroneous mapping could result in false positive variants, especially if the mapper aligns to just the targeted region of the genome (Qiu et al. 2016; Park et al. 2016) as is possibly performed by MiSeq software.

4.5.3 Performance of other somatic variant calling pipelines

The performance of several somatic variant callers have been evaluated (Krøigård et al. 2016), including MuTect (Cibulskis et al. 2013), Shimmer (Hansen et al. 2013), Somatic Sniper (Larson et al. 2012), VarScan 2 (Koboldt et al. 2012) and Strelka (Saunders et al. 2012).

For sequencing regions at great depth, MuTect and Strelka appear more agreeable between callers (Krøigård et al. 2016), generating a larger pool of common calls. Somatic Sniper appears to generate variant many calls that are not detected by other callers, while Shimmer fails to detect variants detected by multiple other callers (Krøigård et al. 2016). A middle group may therefore be optimal, seeking to maximise the number of true positives without generating too many false negatives. Our study used a somatic variant caller as part of a software package, for which we had no control over parameters; using a separate somatic caller would allow for control over parameters and therefore finetuning.

Other studies have found differences between numbers of mutated genes reported by SomaticSniper and MuTect, with MuTect reporting more mutant genes and a higher frequency of single nucleotide variants (Bodini et al. 2015) including those that are implicated in tumours (Futreal et al. 2004).

Studies looking at somatic variation in CRC have also focused on hotspots in genes such as *KRAS*, *NRAS* and *BRAF* using commercially available variant callers (Sakai et al. 2015). Other groups have used larger panels of around 50 genes tot sequence CRCs and reported rare somatic variants in genes such as *CDH1*, *APC, MLH1* and *NRAS* (Chang et al. 2016). Sakai et al. (2015) report C:G>A:T false positive calls which if genuine would have lead to a mutation a *KRAS* codon 37 and 175, and *NRAS* codon 351. In their study, they filtered out those particular sites in downstream analyses. They also report detection of the p.G12S mutation in *KRAS* is detectable down to a frequency 2.4% and p.G13D is detectable at 3.92% . In *NRAS*, they detect p.Q61K at 0.58% and *BRAF* p.V600E at 1.85%. Detection of such low frequency variants provides confidence in high depth, targeted sequencing approaches such as ours. However, it is also possible that artefacts at these frequencies may also be erroneously identified as somatic variants.

4.5.4 FFPE background mutation profile

In addition to oxidative DNA damage generated during sample preparation (Costello et al. 2013) and errors introduced by the polymerase used (Chen 2014), DNA derived from FFPE contains pre-existing damage that contributes to a background mutation profile. It has previously been suggested that there is an FFPE-associated C:G>T:A substitution profile (Do & Dobrovic 2012; Do et al. 2013; Bourgon et al. 2014) and we also found a significant increase in C:G>T:A's in FFPE tissue as compared to matched blood DNA. We found this excess in 5/19 samples from our first cohort and 7/19 samples from our second cohort, suggesting that the induced damage was specific to a limited number of the FFPE samples (perhaps those treated too long or too harshly). This profile is thought to be caused by deamination of cytosine to uracil; adenine is then misincorporated opposite and, upon DNA replication, results in the replacement of the original cytosine with a thymine (Do & Dobrovic 2012; Do et al. 2013; Bourgon et al. 2014). Such artefacts could easily be mistaken for low-level clinically important mutations (Lamy et al. 2011).

A:T>G:C transitions also appeared frequently in our FFPE-processed samples. Oxidation of thymine's methyl group results in 5-hydroxymethyl uracil (5-hmU) and 5-formyluracil (5-foU) (Klungland et al. 2001; Schormann et al. 2014). Whilst 5-hmU exhibits normal base pairing (i.e. base pairs with adenine, as expected of uracil), 5-foU is capable of pairing with adenine and guanine (Klungland et al. 2001; Masaoka et al. 2001; Krokan et al. 2002). When 5-foU is present in template DNA, it has been shown that efficiency of opposite incorporation of adenine drops to 25-50% of that when pairing with thymine. Incorporation of guanine opposite 5-foU however, increases by as much as 5.6-fold; this accelerated rate may be promoted by increasing pH (7.2 – 8.6) (Masaoka et al. 2001). Masaoka et al. (2001) also showed that template extension is possible using DNA polymerase, even when the mismatch appears at the terminus of a primer. During replication, 5-foU can induce A:T>G:C transitions (Krokan et al. 1997; Kasai et al. 1990; Bjelland et al. 1995). Whether formalin-fixation or the extraction process itself is capable of driving the oxidation of thymine to 5-foU, and hence A:T>G:C mutations is not clear.

It has been suggested that treating FFPE derived DNA with UDG prior to use in NGS reduces the frequency of C:G>T:A transitions (Do & Dobrovic 2012; Do et al. 2013) and our data support this. It has been proposed that the abasic sites created by UDG's excision of uracil in FFPE DNA halt the progression of certain polymerases, including the polymerase used in Illumina's TSCA protocol (Sikorsky et al. 2007). However, it is not clear why the level of C:G>T:A's in the UDG-treated nFFPE-DNA samples observed *herein* does not reach that observed in gDNA, but this could be explained by the methylation of cytosine at CpG islands (Do et al. 2013). At such sites, methylated cytosine is deaminated to thymine and is not targeted by UDG.

When using NGS as a screening tool to find novel somatic mutations in FFPE DNA, we observed large numbers of SNVs, most of which are likely to be artefacts. It has been suggested that the use of replicates sequenced using independent NGS runs helps mitigate against the errors encountered (Robasky et al. 2014). We sequenced 19 samples in duplicate to determine the number of somatic variants shared between runs. We found that only 5.3% of variants were consistently predicted across both runs, and the majority of these were present in the germline. Multiple runs of the same samples may therefore help to remove artefacts in downstream analyses.

4.5.5 Recommendations for somatic variant calling from FFPE samples

We therefore recommend taking care to avoid cross contamination of normal and tumour tissue during sample preparation, pre-treating FFPE-associated DNA with UDG, mapping data to the whole reference genome, performing local realignment and base quality score recalibration, and using a somatic variant caller such as MuTect (Cibulskis et al. 2013) to accurately detect somatic variants.

Chapter 5 – Investigation of potential novel genetic biomarkers of cetuximab-resistance

5.1 Details of work performed

In this chapter, library preparation and sequencing was performed by myself and Dr Christopher Smith. Bioinformatics analyses was performed by myself, and assessment of on-target read metrics was cross-checked by Dr Kevin Ashelford.

5.2 Introduction

NGS is increasingly used on a routine basis to interrogate the somatic mutation profiles of a range of cancers including CRCs. Precise and thorough mutation profiles are essential to help inform the prognostic outcome of the patient (Drmanac et al. 1998; Tian et al. 2013). Furthermore, since therapy is increasingly stratified upon the tumours mutation profile, optimal efficacy is dependent on accurate high-throughput sequence analysis. For example, somatic mutations in *KRAS*, present in 35-45% of CRCs (Maughan et al. 2011; Smith et al. 2013), rendered cetuximab (a monoclonal antibody against EGFR) ineffective, by virtue of the tumour cell's ability to constitutively activate EGFR signalling independent of ligand binding (Segal et al. 2015). For those aCRC patients with *KRAS* wild-type tumours, cetuximab was shown to increase median survival by 4.7 months compared with best supportive care alone, although tumour location can significantly influence patient outcome (Lu et al. 2016). It is likely that mutations in other EGFR pathway genes confer resistance to cetuximab; for example, mutations in *NRAS* (mutated in 2-4% of CRCs) cause resistance (De Roock, Claes, et al. 2010). *BRAF* lies immediately downstream of *KRAS* and is mutated in ~8 - 10% of all CRCs (Barras 2015). Generally, mutations of *BRAF* are mutually exclusive of those in *KRAS* (Smith et al. 2013) and it is likely that *BRAF* mutations also confer cetuximab resistance (Barras 2015; De Roock, Claes, et al. 2010; Di Nicolantonio et al. 2008).

Here, we initially sought novel genes causing resistance to cetuximab by carrying out NGS of EGFR pathway genes in CRCs from patients that did not respond to treatment. To exclude known mechanisms of resistance, we focused on patients that were wild-type for common hot-spot mutations in *KRAS, BRAF* and *NRAS*. We discovered hospot mutations in a number of patients that were missed by traditional methods, as well as other non-synonymous *KRAS* and EGFR pathway mutations that may lead to cetuximab resistant. The mutation profile observed in our samples hints at a possible role of DNA methylation in cetuximab response.

5.3 Materials and Methods

5.3.1 Patient selection criteria and samples

Patients were enrolled in the MRC trials COIN or COIN-B (Maughan et al. 2011; Chapter 2, Section 2.2). Patients were selected for their lack of response to cetuximab (from Arm B of COIN and both arms of COIN-B). Response was defined according to Response Evaluation Criteria in Solid Tumours (RECIST) guidelines. We selected patients that showed either ‘progressive’ or ‘stable’ disease at 12 weeks, who’s ‘best response’ over the course of the trial was either ‘progressive’ or ‘stable’ disease and that were wild type for *KRAS*, *BRAF* and *NRAS* hotspot mutations as determined by Sequenom and Pyrosequencing (Maughan et al. 2011; Smith et al. 2013; Chapter 2, Section 2.6).

5.3.2 Germline and FFPE DNA extraction and UDG treatment

We obtained gDNA from these patients, as well as matched tFFPE DNA from their CRCs as previously described (Chapter 2, Section 2.4). Treatment with UDG was carried out as described (Chapter 2, Section 2.7).

5.3.3 TruSeq Custom Amplicon Design

Custom amplicons were designed using Illumina’s DesignStudio software targeting the ORFs of 49 genes (including *KRAS*, *BRAF* and *NRAS*) (Chapter 2, Section 2.8.2.1) with a known or suspected role in EGFR signalling or that had been implicated in altering response to cetuximab (Chapter 1, Section 1.6.5, Figure 1.8). Optimum amplicon length was 175 bp.

TSCA Library Preparation was carried out as described in Chapter 2, Section 2.8.2.2. Two DNA samples from each FFPE CRC were processed; one was treated with UDG (+UDG) to reduce FFPE-associated C:G>T:A artefacts (Do et al. 2013; Do & Dobrovic 2012; Hofreiter et al. 2001; Chapter 4, Section 4.3.3), and the other remained untreated (-UDG). Pre- and post-PCR phases of work were carried out in dedicated laboratories.

5.3.4 MiSeq sequencing

All sequencing was carried out using an Illumina MiSeq (Chapter 2, Section 2.8.2.4). To ensure that a minimum sequencing depth of 400x was reached, batches of ≤14 samples were sequenced at any one time. For each run, 150 cycles of paired-end sequencing was performed.

5.3.5 Bioinformatics pipeline and somatic variant calling

Alignment of sequencing reads was performed using BWA (Li & Durbin 2009) as described in Chapter 2, Section 2.9.1. Base quality score recalibration and indel realignment were also performed using SAMtools and GATK (Chapter 2, Section 2.9.2). Reads not mapping to our target panel were removed using Custom Script 6. Dr Kevin Ashelford (Wales Gene Park Bioinformatics Facility and co-supervisor) validated the removal of off-target reads.

Somatic variants were called using MuTect (Cibulskis et al. 2013) using the UDG-treated tumour and the matched germline sample. Parameters used for MuTect are described in Chapter 2, Section 2.9.3.

Variant annotation was performed using snpEff (Cingolani et al. 2012) and Custom Script 2 (Chapter 2, Section 2.9.4). Nucleotides adjacent to the variant site were taken from human reference GRCh37 and included in the the final annotation for somatic variants.

5.3.6 Coverage calculations

Depth of coverage across each sequenced sample was calculated using Custom Script 3 (Chapter 2, Section 2.9.5). Average coverage was calculated for all 49 genes on our target panel, along with the percent of each ORF covered to a depth of ≥500x.

5.3.7 Somatic variant filtering and profiling

Somatic variants were included for downstream analysis if they were assigned a 'PASS' in the filter field by MuTect. Only variants likely to change the encoded protein sequence were included, as determined by an 'EXON', 'NON_SYNONYMOUS_CODING', 'SPLICE_SITE_ACCEPTOR', 'SPLICE_SITE_DONOR', 'START_GAINED', 'START_LOST', 'STOP_GAINED', or 'STOP_LOST' tag issued by snpEff (Cingolani et al. 2012). The number of specific nucleotide alterations across our gene panel was recorded and the reference genome (GRCh37) was consulted to determine whether C:G>T:A mutations were in a CpG island or not.

5.4 Results

5.4.1 Patients

Twenty one aCRC patients were selected for whom we had matching germline and tumour DNA, who showed no response to cetuximab, and that were believed to be wild-type for the common somatic hotspots in *KRAS*, *BRAF* and *NRAS*.

5.4.2 Gene panel

A total of 1,536 amplicons were generated targeting the ORFs of the 49 selected genes (including *KRAS*, *BRAF* and *NRAS*) in the EGFR pathway. Illumina's DesignStudio in-built scoring system (that estimates the percentage of attempted probes that will successfully enrich the target sequences) predicted an average score of 79% (ranging from 60-98%). Average amplicon length was 176 bp (ranging from 170-190 bp). The cumulative target size was 112,427 bp. In total there were 37 gaps in the target sequence covering 1,664 bp of sequence. However, total ORF coverage was 99%.

5.4.3 Sequencing of prepared libraries

Four TruSeq Custom Amplicon libraries were prepared and sequenced across separate runs on an Illumina MiSeq. Libraries 1 and 3 were each sequenced on a single MiSeq run. Libraries 2 and 4 were pooled into two aliquots each and each aliquot sequenced on a separate MiSeq run. One hundred and fifty cycles of paired end sequencing was performed for each run.

5.4.4 Read mapping and removal of off-target reads

Mapped reads that did not map to our target region were removed. This left a total of 108,557,316 mapped reads (An average of 18,092,886 per run; Table 5.1) and 47,707,903 unmapped reads (average 7,951,317 per run; Table 5.1). We had 16.39 Gb mapped data across all six runs for downstream analyses. Average mapping quality was 57.81, and an average 97.4% mapped correctly as a pair (Table 5.1).

5.4.5 Depth of coverage

Germline samples consistently achieved better coverage across all 49 ORFs than their matched tumour counterparts (Figure 5.1). At ≥500x coverage, average ORF coverage was 76.62%, 50.69% and 40.85% in germline, tumour and UDG-treated tumour samples respectively (Table 5.2). Average sequencing depth over the 49 genes in our targeted panel was 1,159.19x (range 227.02 – 2,626.57x; Table 5.2).

Table 5.1 Sequencing and mapping metrics for samples sequenced on the MiSeq.

Run ID (library)	**Tissue**	**Mapping status**	**Read count**	**Mb obtained**	**Correctly mapped (%)**	**Mapping quality**
1 (1)	blood	mapped	35,657,382	5,384.26	97.69	57.72
1 (1)	blood	unmapped	1,859,426	280.77	0.00	0.00
2 (2)	tFFPE	mapped	12,769,865	1,928.25	97.31	57.90
2 (2)	tFFPE	unmapped	10,089,369	1,523.49	0.00	0.00
3 (2)	tFFPE	mapped	20,257,137	3,058.83	98.22	58.06
3 (2)	tFFPE	unmapped	3,075,240	464.36	0.00	0.00
3 (2)	UDG tFFPE	mapped	957,526	144.59	98.59	58.12
3 (2)	UDG	unmapped	806,882	121.84	0.00	0.00

	tFFPE					
4 (3)	blood	mapped	6,857,241	1,035.44	97.92	57.99
4 (3)	blood	unmapped	284,023	42.89	0.00	0.00
4 (3)	UDG tFFPE	mapped	10,246,556	1,547.23	98.16	58.12
4 (3)	UDG tFFPE	unmapped	7,862,654	1,187.26	0.00	0.00
5 (4)	UDG tFFPE	mapped	9,954,674	1,503.16	96.81	57.79
5 (4)	UDG tFFPE	unmapped	9,395,083	1,418.66	0.00	0.00
6 (4)	blood	mapped	5,965,826	900.84	97.86	57.87
6 (4)	blood	unmapped	1,266,559	191.25	0.00	0.00
6 (4)	tFFPE	mapped	2,538,989	383.39	95.66	57.17
6 (4)	tFFPE	unmapped	3,875,582	585.21	0.00	0.00
6 (4)	UDG tFFPE	mapped	3,352,120	506.17	95.75	57.35
6 (4)	UDG tFFPE	unmapped	9,193,085	1,388.16	0.00	0.00

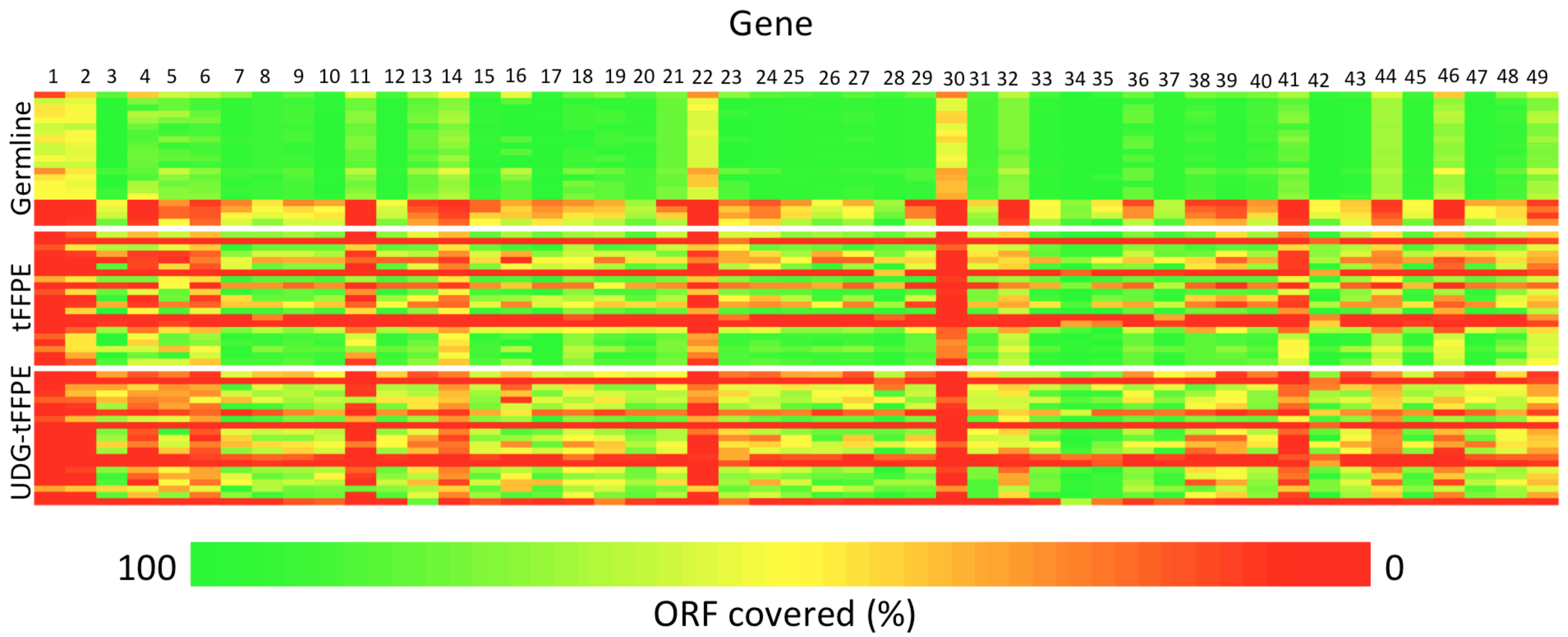


Figure 5.1 ORF coverage at ≥500x for genes in our target panel. Coverage is shown for germline, tumour FFPE and UDG-treated tumour FFPE samples.

Table 5.2 Average ORF depth and coverage at ≥500x across our gene panel. Coverage calculations for each tissue are shown.

Gene number	Gene name	Average depth			% ORF covered at ≥500x		
		gDNA	tumour	UDG-tumour	gDNA	tumour	UDG-tumour
1	AKT1	452.17	135.74	91.55	40.76	6.66	3.91
2	AKT2	715.54	267.02	161.05	44.11	20.65	10.07
3	AKT3	1,985.98	1,115.48	663.64	90.37	56.16	45.66
4	ARAF	826.78	481.31	296.51	61.52	31.74	20.07
5	AREG	1,122.16	741.14	471.80	70.58	45.65	33.70
6	AXL	1,169.76	525.77	356.74	69.84	30.99	21.27
7	BCL2L11	1,307.65	1,099.07	732.81	84.17	62.04	48.46
8	BRAF	1,886.95	1,207.76	731.52	85.63	61.26	51.03
9	CBL	1,973.15	1,238.06	824.44	81.19	58.84	49.48
10	CDH1	2,013.22	1,187.93	804.59	89.09	59.87	49.59
11	DOK2	891.00	376.58	264.64	62.96	23.72	13.98
12	EGF	2,182.05	1,263.60	801.75	90.94	60.63	49.97
13	EGFR	1,337.83	1,234.09	776.27	69.39	50.57	40.72
14	ERBB2	1,075.70	510.52	356.21	59.52	29.09	20.40
15	ERBB3	2,390.57	1,296.38	911.00	83.50	61.54	52.71
16	EREG	1,478.31	980.82	668.00	79.85	54.24	38.24
17	FCGR2A	2,288.46	1,366.12	808.89	86.09	63.77	52.57
18	FCGR3A	2,279.05	1,261.42	878.01	80.61	52.17	42.99
19	FLT1	1,727.07	1,274.54	686.17	80.72	57.37	45.50
20	GAB1	2,418.94	1,651.93	995.58	88.03	62.15	55.83
21	GRB2	1,506.57	892.22	629.59	71.41	55.14	46.21
22	HRAS	432.78	235.48	176.51	42.05	15.30	9.25
23	IGF1	2,130.14	1,393.14	1,022.35	82.96	54.29	43.60
24	IGF1R	1,716.61	954.72	608.75	82.95	52.99	40.97
25	JAK1	2,130.36	1,291.00	812.55	86.32	60.35	48.85
26	JAK2	2,085.23	1,264.44	806.60	88.98	60.16	51.56
27	KDR	2,080.99	1,098.57	632.05	85.08	57.24	44.54
28	KRAS	2,018.14	1,297.71	885.97	91.46	70.50	58.92
29	MAP2K1	1,804.25	1,004.58	660.48	80.12	53.74	47.38
30	MAP2K2	495.62	135.97	104.02	36.61	8.68	5.36
31	MAPK1	1,998.77	1,125.84	772.81	83.81	53.89	49.67
32	MAPK3	1,409.02	656.25	511.38	61.73	41.16	36.14
33	MET	1,959.49	1,571.36	944.34	88.38	65.53	56.93
34	NCK1	3,308.38	2,551.07	1,510.19	95.92	78.70	73.07
35	NRAS	2,229.87	1,449.62	1,015.41	92.41	69.03	64.92
36	PDK1	1,755.20	1,179.56	769.24	76.23	55.89	48.88
37	PIK3CA	2,454.70	1,764.31	1,098.53	91.71	71.96	61.13
38	PLCG1	1,657.75	1,066.96	646.54	76.95	54.08	38.86

39	PLCG2	1,463.91	773.15	471.42	80.92	48.04	33.32
40	PRKCA	1,895.87	1,132.03	755.25	80.68	57.34	47.73
41	PRKCD	1,022.79	395.57	237.26	65.39	26.84	14.62
42	PTEN	2,086.68	2,061.92	1,310.66	92.05	76.10	61.37
43	RAF1	1,887.09	968.52	618.96	87.37	54.60	42.24
44	SHC1	1,220.03	571.12	381.33	58.55	35.99	24.66
45	SOS1	2,010.36	1,422.81	905.89	88.47	64.47	56.23
46	SRC	981.79	615.84	428.24	55.39	34.85	27.29
47	STAT1	1,685.86	1,126.15	675.99	89.22	61.15	46.28
48	STAT3	1,812.76	1,159.05	769.54	81.17	55.48	45.35
49	TP53	1,354.86	690.20	446.63	61.42	41.39	30.01
Average		**1,675.88**	**1,042.13**	**671.22**	**76.62**	**50.69**	**40.85**

5.4.6 Somatic *KRAS* analyses

KRAS hotspot mutations were detected in seven of twenty one (33.3%) UDG-treated CRCs that were believed to be wild type prior to NGS. Five (23.8%) CRCs had multiple hotspot mutations; one (4.8%) CRC had a p.G12D (g.25,398,284C>T) and a p.G12C (g.25,398,285C>A) mutation, three (14.3%) CRCs had a p.G12V (g.25,398,284C>A) and a p.G12C (g.25,398,285C>A) mutation and one CRC had a p.G12V (g.25,398,284C>A) and a p.Q61R (g.25,380,276T>C) mutation. Of the remaining two (9.5%) CRCs, one (4.8%) CRC had a p.G12V (g.25,398,284C>A) mutation and one (4.8%) had a p.Q61H (g.25,380,275T>G) mutation.

Twenty five potential protein-altering mutations were predicted outside of known hotspot across ten of the twenty one (47.6%) CRCs whose DNA was sequenced (Table 5.3). Five (20%) of these mutations were likely to either truncate or lengthen the protein. One CRC carried a g.25,398,324G>A start gain mutation upstream of codon 1, potentially altering the reading frame of the gene. One CRC carried a truncating p.K5X (g.25,398,306T>A) mutation and a g.25,378,709T>C splice site acceptor mutation. A separate CRC carried a p.K101X (g.25,378,697T>A) truncating mutation. One CRC carried a p.X190Q (g.25,368,377A>G) mutation, resulting in the loss of a stop codon.

Table 5.3 Somatic *KRAS* mutations detected in CRCs that showed lack of response to cetuximab.

Patient ID	**Variant type**	
	ORF altering	**Other non-synonymous**
36		p.K16M (g.25398272T>A)
71		p.T87S (g.25380198G>C)
72	p.X190Q (g.25368377A>G)	p.S172R (g.25368429G>C) p.E31K (g.25398228C>T)
75	codon unspecified (g.25398324G>A)	
77	p.K5X (g.25398306T>A) codon unspecified (g.25378709T>C)	p.T127I (g.25378618G>A), p.M67I (g.25380257C>T) p.A66V (g.25380261G>A) p.Y64C (g.25380267T>C)
79		p.K104R (g.25378687T>C) p.L6H (g.25398302A>T)
85	p.K101X (g.25378697T>A)	p.Q129R (g.25378612T>C) p.V103A (g.25378690A>G)
86		p.K104R (g.25378687T>C) p.S89L (g.25380192G>A) p.E49G (g.25380312T>C)
87		p.K117I (g.25378648T>A) p.Y96C (g.25380171T>C)
82		p.P178R (g.25368412G>C) p.Q150R (g.25378549T>C) p.R68K (g.25380255C>T)

5.4.7 Somatic hotspot profiles across *BRAF*, *NRAS*, *PIK3CA* and *PTEN*

No CRCs carried any hotspot mutations at codons 600 and 594 in *BRAF*. One CRC, however, carried a *BRAF* p.F595S (g.140,453,151A>G) mutation. In *NRAS*, one CRC posessed a G13D (g.115,258,744C>T) hotspot mutation. For *PIK3CA*, one CRC carried both a p.E542K (g. 178,936,082G>A) and a p.E545K (g.178,936,091G>A) mutation. Three other CRCs also carried a single p.E545K mutation.

In *PTEN*, eight CRCs carried mutations that were likely to alter the ORF. Several nonsense mutations were called, including p.C71X, p.R130X, p.Y155X, p.Y178X, and p.R189X, each carried by a separate CRC. One CRC carried p.K260X, p.K330X and p.R335X. One CRC carried a g.89,725,043G>A splice site acceptor, one carried a g.896,53,780A>T splice site acceptor and one CRC carried a g.89,690,801A>T splice site acceptor. One CRC possessed a p.K289R mutation; mutations at this codon are thought to affect localisation of the protein (Song et al. 2012).

5.4.8 Background somatic profile

A total of 3,054 variants were included for analysis after filtering, equating to a mean 63.33 variants (range 7 – 144) per gene across all twenty one UDG-treated CRCs (Table 5.4). Of these, 2.59% were A:T>C:G, 18.11% were A:T>G:C, 17.87% were A:T>T:A, 12.85% were C:G>A:T, 4.49% were C:G>G:C and 44.09% were C:G>T:A variants. A mean 49.41% of the C:G>T:A mutations were at CpG islands (range 0 – 91.67%, Figure 5.2).

Table 5.4 Variant profile in DNA from UDG-treated CRCs across a 49 gene panel.

Gene	**Variant count**	**A:T>C:G (%)**	**A:T>G:C (%)**	**A:T>T:A (%)**	**C:G>A:T (%)**	**C:G>G:C (%)**	**C:G>T:A (%)**	**C:G>T:A in CpG (%)**
AKT1	23	0.00	34.78	8.70	4.35	0.00	52.17	91.67
AKT2	36	2.78	33.33	13.89	11.11	2.78	36.11	61.54
AKT3	40	0.00	10.00	22.50	12.50	7.50	47.50	26.32
ARAF	52	0.00	21.15	17.31	17.31	3.85	40.38	52.38
AREG	14	0.00	28.57	28.57	7.14	7.14	28.57	0.00
AXL	67	1.49	10.45	13.43	14.93	2.99	56.72	50.00
BCL2L11	21	0.00	23.81	9.52	9.52	4.76	52.38	63.64
BRAF	69	5.80	21.74	26.09	11.59	4.35	30.43	38.10
CBL	70	4.29	12.86	18.57	12.86	8.57	42.86	53.33
CDH1	83	2.41	12.05	16.87	9.64	6.02	53.01	59.09
DOK2	39	0.00	15.38	10.26	2.56	5.13	66.67	69.23
EGF	111	1.80	20.72	24.32	11.71	1.80	39.64	40.91
EGFR	144	4.86	23.61	20.14	7.64	5.56	38.19	63.64
ERBB2	88	1.14	14.77	15.91	10.23	4.55	53.41	48.94
ERBB3	144	3.47	5.56	18.75	18.06	4.17	50.00	48.61
EREG	17	0.00	5.88	29.41	17.65	0.00	47.06	37.50
FCGR2A	23	0.00	17.39	21.74	13.04	13.04	34.78	50.00
FCGR3A	22	9.09	13.64	22.73	13.64	4.55	36.36	37.50
FLT1	115	0.87	16.52	20.87	15.65	2.61	43.48	38.00
GAB1	63	1.59	11.11	11.11	17.46	1.59	57.14	38.89
GRB2	16	0.00	18.75	18.75	6.25	0.00	56.25	55.56
HRAS	7	0.00	0.00	0.00	28.57	14.29	57.14	75.00
IGF1	14	0.00	21.43	14.29	7.14	21.43	35.71	60.00
IGF1R	117	3.42	22.22	17.09	17.95	5.13	34.19	67.50
JAK1	90	1.11	17.78	24.44	11.11	3.33	42.22	57.89
JAK2	90	5.56	18.89	15.56	7.78	3.33	48.89	43.18
KDR	127	1.57	14.96	19.69	14.96	2.36	46.46	30.51
KRAS	37	2.70	29.73	13.51	24.32	8.11	21.62	25.00
MAP2K1	26	0.00	11.54	23.08	15.38	0.00	50.00	53.85
MAP2K2	24	0.00	25.00	4.17	12.50	0.00	58.33	71.43
MAPK1	19	5.26	21.05	21.05	15.79	5.26	31.58	33.33
MAPK3	36	2.78	13.89	13.89	11.11	2.78	55.56	70.00
MET	113	3.54	10.62	22.12	17.70	4.42	41.59	46.81
NCK1	31	3.23	9.68	12.90	16.13	3.23	54.84	41.18

NRAS	40	12.50	22.50	22.50	7.50	2.50	32.50	38.46
PDK1	32	6.25	18.75	18.75	6.25	0.00	50.00	37.50
PIK3CA	113	4.42	18.58	22.12	12.39	2.65	39.82	22.22
PLCG1	110	3.64	12.73	18.18	13.64	6.36	45.45	54.00
PLCG2	100	4.00	16.00	16.00	21.00	2.00	41.00	60.98
PRKCA	43	2.33	11.63	25.58	23.26	2.33	34.88	46.67
PRKCD	37	0.00	29.73	16.22	2.70	2.70	48.65	88.89
PTEN	91	6.59	30.77	34.07	5.49	0.00	23.08	28.57
RAF1	65	0.00	12.31	18.46	23.08	6.15	40.00	53.85
SHC1	36	0.00	13.89	13.89	5.56	8.33	58.33	52.38
SOS1	105	4.76	14.29	18.10	17.14	3.81	41.90	29.55
SRC	47	6.38	21.28	6.38	6.38	2.13	57.45	81.48
STAT1	63	0.00	26.98	15.87	7.94	6.35	42.86	37.04
STAT3	68	1.47	26.47	22.06	14.71	1.47	33.82	56.52
TP53	116	6.03	22.41	16.38	17.24	8.62	29.31	32.35
Average	**62.327**	**2.59**	**18.11**	**17.87**	**12.85**	**4.49**	**44.09**	**49.41**

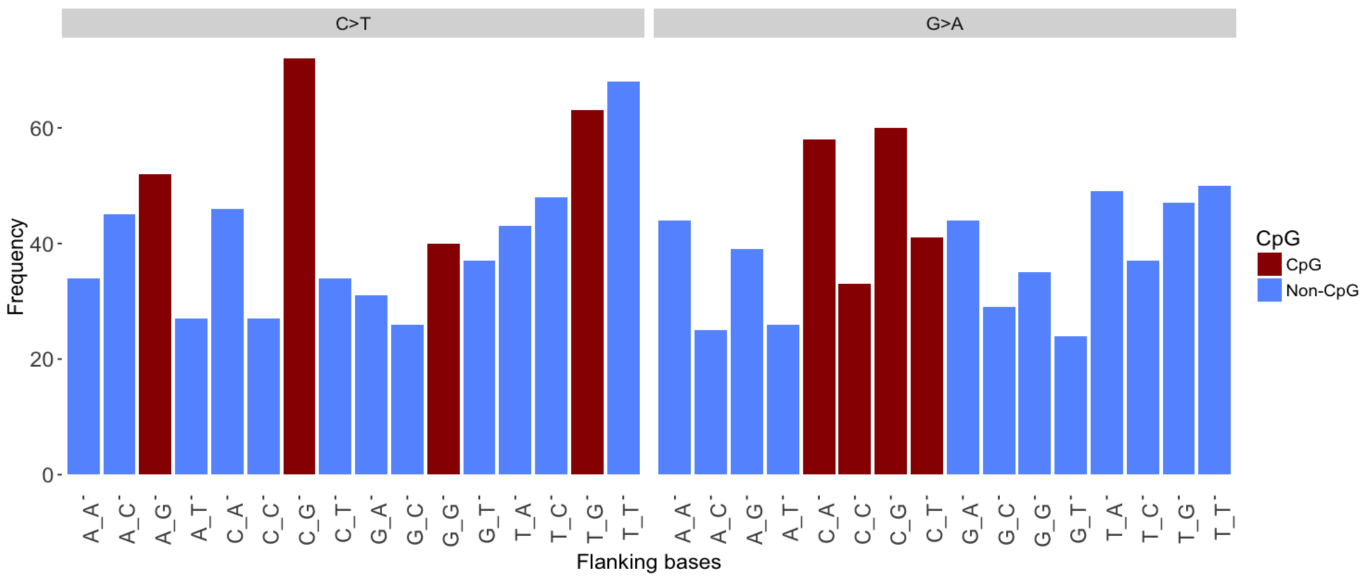


Figure 5.2 Sequence context of C:G>T:A mutations. Sequences involving CpG dinucleotides are indicated in red.

Mutations likely to affect the open reading frame, such as truncating nonsense mutations, proved less abundant than those that changed only the amino acid sequence (labelled by snpEff as ‘NON_SYNONYMOUS’, Figure 5.3). There was a marked increase in the frequency of C:G>T:A mutations.

5.5 Discussion

5.5.1 Read mapping

We observed that, on average, 18,092,886 reads were mapped on-target per MiSeq run across our 49 gene panel. An average of 7,951,317 reads per run were unmapped. Over 95% of our on-target mapped reads were mapped correctly with their paired read, and had consistently good mapping quality scores. This shows that BWA performed adequately across all our sequencing data.

Any mapped reads that did not match our target coordinates were discarded, whereas unmapped reads could not technically be removed using the same methodology. Given that our mapping quality metrics appear good, it is surprising to see the numbers of unmapped reads listed in Table 5.1. Interestingly, Run 1 (consisting of library 1) had the lowest ratio of on-target mapped reads to unmapped reads. This suggests later libraries had a higher portion of off-target reads that were discarded after mapping.

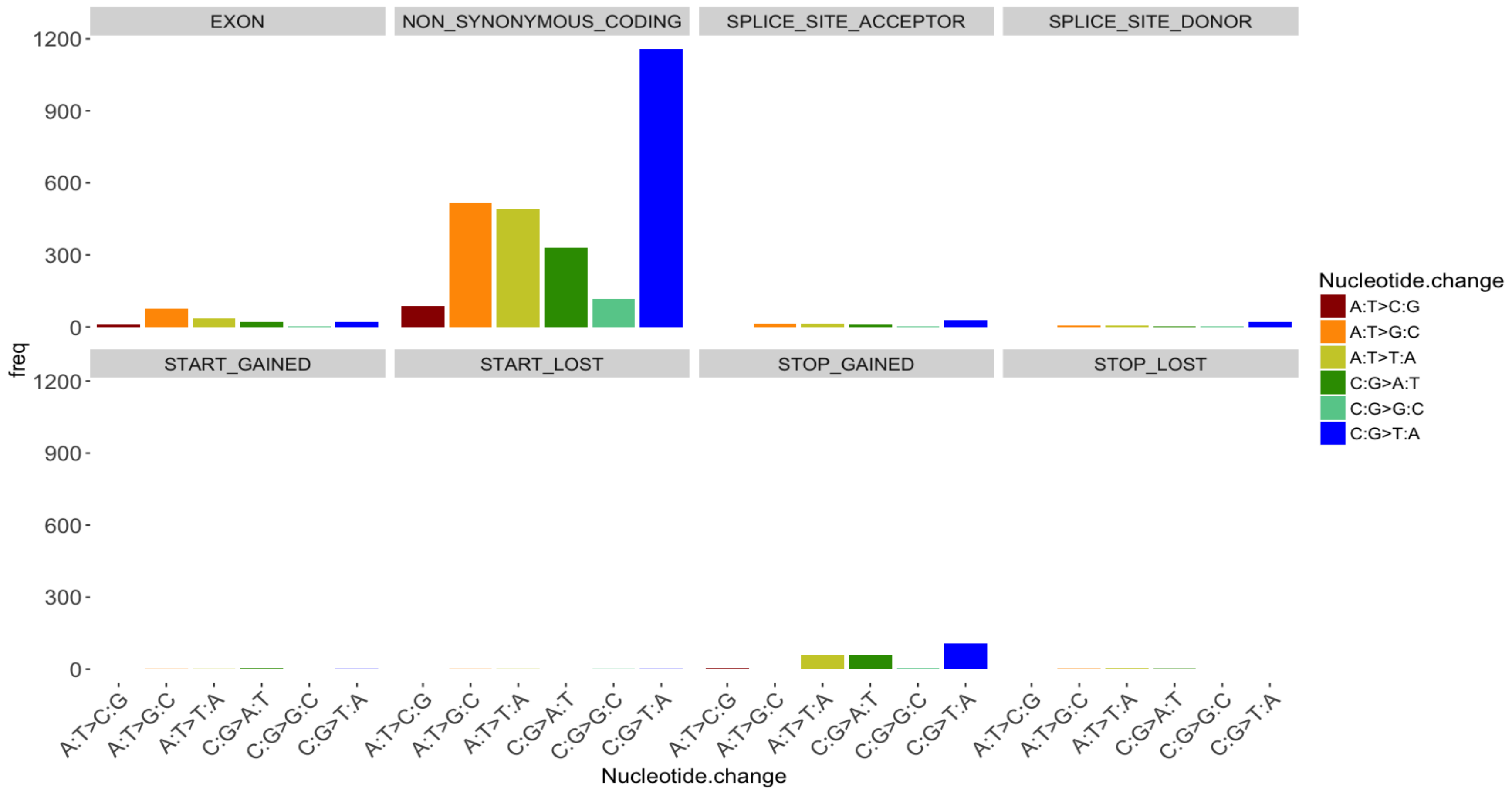


Figure 5.3 Mutation frequencies arranged by predicted affect on the encoded protein sequence.

5.5.2 Sequencing coverage

All of our samples were sequenced to a depth of >200x, with an average depth of 1,159.19x. Depth of coverage is an important consideration during NGS as increasing depth increases the sensitivity of somatic mutation detection; depths of around 1000x may be required to detect low frequency (1%) somatic mutations with a low false discovery rate (K. Chen et al. 2015). Lower depth may result in more sequencing artefacts being detected and perhaps called as somatic mutations if the matched germline sample does not have a similar sequencing depth (Alioto et al. 2015). We achieved a suitably high sequencing depth of 671.22x in our UDG-treated CRCs used for somatic variant calling. Matched germline samples yielded a depth of 1,675.88x, providing confidence in comparisons with our somatic samples.

We achieved ≥500x depth in 76.62%, 50.69% and 40.85% of our targeted region in germline, tumour and UDG-treated tumour samples, respectively. The differences observed between germline and tumour samples could be due the properties of FFPE-associated DNA (Carrick et al. 2015; Van Allen et al. 2014). The depth achieved across our targets is sufficient for calling of mutation with a variant allele frequency >1%, but lower frequency variants may be misinterpreted as sequencing artefacts. Higher depth is advised to ensure low false discovery rates (K. Chen et al. 2015)

5.5.3 Somatic analyses of *KRAS*, *BRAF*, *NRAS*, *PIK3CA* and *PTEN*

We detected multiple somatic hotspot mutations in CRCs that had not been detected via earlier methods. It is possible that the earlier methods used to determine the mutation status were not sensitive enough to detect low frequency mutations, whereas the increased sensitivity of NGS (Fujita et al. 2016; Chin et al. 2013) was sufficient for mutation detection.

In *KRAS*, a third (7/21) of CRCs had ≥1 hotspot mutation including p.G12C, p.G12D, p.G12V, p.Q61H and p.Q61R. None of these mutations were previously detected in the CRCs. Other *KRAS* mutations were detected that could have significantly altered the protein-coding region, including a p.K5X mutation. While this mutation is truncating, other mutations at codon 5 have been shown to enhance phosphorylation of downstream affectors (Gremer et al. 2011). A separate truncating mutation was found at codon 101 (p.K101X); it has been suggested that this is a potential ligand binding site (away from the active site) in HRAS and also probably in KRAS (Buhrman et al. 2011). A p.X190Q stop loss mutation is likely to change the structure and therefore overall function of the protein. In *NRAS*, a p.G13D mutation was found. The hotspot mutations in RAS genes were likely missed due to lower sensitivity of earlier technologies.

In *BRAF*, a p.F595S mutation was detected in one CRC; this CRC also harboured the *KRAS* p.K5X mutation along with codon 12 mutations. This mutation alters the functionally important motif spanning codons 594-595-596, and mutations specifically at codon 595 have previously been shown to be activating (Kordes et al. 2015); this mutation therefore represents a good candidate for consideration when treating CRC with EGFR pathway inhibitors.

PIK3CA had potentially activating mutations, including p.E542K, p.E545K, p.M1043I and p.A1046T (S. J. Chen et al. 2015; Tian et al. 2013). The CRC carrying p.M1043I also carried p.E542K and p.E545K, three truncating mutations in *PTEN*, but did not carry any *KRAS*, *NRAS* or *BRAF* hotspot mutations. Mutations at codons 1043 and 1046, along with codons 542, 545 and 1047 are good candidates for further investigation into their role in cetuximab-resistance.

Mutations in *PTEN* were detected that would have probably resulted in loss of function (Molinari & Frattini 2013). These included five truncating mutations at codons 71, 130, 155, 178 and 189, three predicted splice site variants and a p.K289R mutation that could affect protein localisation (Song et al. 2012). As loss of *PTEN* leads to potential activation of the PI3K pathway (Dillon & Miller 2014), cetuximab may not be effective at treating CRCs harbouring these mutations.

5.5.4 Background somatic profile

We observed high ratios of transition (A:T>G:C and C:G>T:A) mutations compared to transversions mutations, as expected when sequencing coding regions (Q. Liu et al. 2012). The high levels of transitions in our samples are generally consistent with colorectal mutation profiles (Alexandrov et al. 2013), although we observe higher levels of A:T>T:A transversion mutations.

Treatment of FFPE-associated CRC has been shown to reduce C:G>T:A artefacts in NGS. However, these mutations are still the most common in our data; they do however appear less likely to be labelled as ‘EXON’ by snpEff (Cingolani et al. 2012) or affect a start or stop codon. This is probably due to the fact that the start codon (ATG) and stop codons (TAA, TAG, TGA) have a limited GC content, hence less opportunity for C:G>T:A mutation to occur. EXON variants are described by snpEff documentation to affect potentially non-coding transcripts or untranslated regions, and therefore may not be subject to the same processes governing fully coding variants.

Given that almost half (>49%) of the C:G>T:A mutations are in a CpG sequence context (Table 5.4, Figure 5.2), the frequency of these variants could be explained by deamination of methylated cytosine. CpG islands are known hotspots for C>T transitions in the human genome (Gromenko et al. 2009), and have been reported at a rate observed in our somatic analyses (Cooper et al. 2010). Such cytosine modifications have been highlighted as potential biomarkers of cell proliferation in colorectal cancer (Uribe-Lewis et al. 2015).

One final observation is that genes with the highest fraction of C:G>T:A mutations in a CpG context, specifically *AKT1*, *HRAS*, *MAP2K2*, *MAPK3*, *PRKCD* and *SRC* (gene numbers 1, 22, 30, 32, 41 and 46 respectively), yielded poorer ORF coverage as shown in Figure 5.1. Genes with lower fractions of C:G>T:A mutations in a CpG context, namely AKT3, KRAS, PIK3CA, PTEN and SOS1 (numbers 3, 28, 37, 42, 45), showed better ORF coverage.

Chapter 6 - General Discussion

6.1 Development of custom Perl scripts

Throughout the course of this PhD, custom scripts were written in Perl to assist with the analyses of the NGS data. Custom script 1 was written to facilitate and manage our bioinformatics pipeline; data in FASTQ format was taken from the sequencers and Custom Script 1 parsed the data through each step in the pipeline. When starting the script, users were given the option to specify software parameters for each step (where applicable). Once the script was running, no more user input was required until the workflow was completed. This semi-automation gave the user free time to focus on other tasks.

Custom Script 2 compiled variants from the NGS data and added annotation to a filterable spreadsheet. Annotation fields added for each variant included presence in dbSNP, 1000 Genomes and COSMIC (Forbes et al. 2008), and their respective allele frequencies. The presence of a variant in each tissue (germline, nFFPE and tFFPE) was specified, allowing for filtering to identify potential somatic mutations.

Custom Script 3 calculated the depth of coverage across the ORFs. ORF regions were defined in CCDS (Pruitt et al. 2009) and RefSeq (Pruitt et al. 2007). This script gave numerical values for the coverage levels of each ORF as well as colour coding the results; this allowed for fast and effective visualisation of the data. Furthermore, it allowed for the identification of contamination or alignment issues by reporting coverage in genes not targeted by the designed panels. Identification of samples that failed sequencing could easily be identified, indicating problems with the sample DNA or library preparation.

Custom Script 4 took variant calls from our pipeline and compared them to calls from BGI's pipeline. The script compared the call sets to determine the variants detected by both pipelines, and which variants were unique. Considering data in this way allowed us to modify our pipeline parameters to maximise the number of variants that were found by both pipelines; doing so reduced the risk that we missed genuine variants.

Custom Script 5 compared genotypes inferred from our variant calls to prior genotyping data. Both datasets were first converted into the same data format, which proved technically challenging. As the cost of NGS falls, genotyping is more likely to be performed using this technology and using our custom script, we have shown that NGS can accurately predict genotypes with high concordance rates.

Custom Script 6 isolated reads mapping to target regions from reads mapping to off-target regions. We have shown that off-target reads represent contamination from other target panels used within our NGS facility. Custom Script 6 was able to effectively remove the contaminating reads. This has important applications in personalised medicine as patients may ultimately receive therapies based on NGS findings; isolation of on-target reads may therefore ensure more accurate variant calls. It should be noted that this script relied on differences between amplicons within different panels; where multiple panels targeted the same amplicons then the script was incapable of determining which panel amplicons originate from.

6.2 Key findings from this project

6.2.1 Key findings from Chapter 3

We used several pieces of bioinformatics software (BWA, SAMtools, GATK, SnpEff) in a bioinformatics pipeline for calling variants from NGS data. Custom Perl scripts were written to facilitate data analyses. By comparing against prior genotyping data and testing different parameter settings, we were able to show that NGS can predict variants with high concordance. Altering the variant calling parameters made only slight differences to the concordance rate, but did substantially increase the number of probable false positives we encountered. There are often discrepancies between variant callers (Krøigård et al. 2016) and choosing parameters which optimise the amount of common calls without increasing the burden of false positives represents a major challenge in bioinformatics. The few genotype discrepancies we observed were most likely due to poor sequencing coverage at the genotyped loci or poor quality score.

When validating SNVs *in vitro*, we showed that over half of the variants could be successfully validated. Variants were more likely to be validated if they were predicted with good quality scores. Our parameter settings that allowed calling from lower quality bases almost certainly contributed to this observation. We were able to validate ~67% of predicted indels, and validation was more successful at loci that had reasonable sequence complexity; validation was harder at loci that contained short dinucleotide repeats. Quality of indel calls has a significant impact upon their validation rate, as indels regarded as low quality have an error rate of >50%, whereas good quality indels have been shown to have a 7% error rate (Fang et al. 2014). It seems sensible then that filtering of variants based on quality scores is performed before effort is spent on further investigating interesting candidates.

Comparing our variant calls to those from BGI's pipeline, we found that we called over 3-fold as many variants as BGI. Our pipeline called the majority of BGI's variants, and relaxing our variant calling parameters allowed us to detect more of these. Relaxing the parameters, however, may have resulted in more potential false positive variants to be called.

We compared the ability of two variant callers (GATK's HaplotypeCaller and Illumina's Somatic Variant Caller) to detect previously determined hotspot mutations.

The options available for GATK users therefore make it a more attractive caller; for instance, variant calling parameters can be adjusted to fine-tune call sets. Furthermore, it is continuously being developed by its creators (McKenna et al. 2010; DePristo et al. 2011). The software can be readily integrated into bioinformatics pipelines whereas this is not as feasible with the MiSeq's Somatic Variant Caller.

6.2.2 Key findings from Chapter 4

We showed that adapter sequences are introduced into NGS reads when the read length extends beyond the amplicon length; this complicates downstream analyses. Use of Trimmomatic removed these adaptor sequences and reduced erroneous variant calls.

The two callers we used in operate very differently; GATK is capable of realigning sequencing reads around indels, recalibrating per-base quality scores and variant quality scores to ensure variants are represented as accurately as possible. GATK expects reads have been aligned against the full human genome sequence. Somatic Variant Caller on the other hand, takes reads that have been aligned against a set of target amplicon sequences. It does not offer realignment of reads or recalibration of quality scores. Somatic Variant Caller is designed to call variants in a tumour sample with no matched germline sample and is therefore very sensitive; as a result it may mistake signal noise for genuine variants. We found that Somatic Variant Caller correctly identified all of the known hotspot variants, but also erroneously called eighteen false positives. GATK’s HaplotypeCaller identified 80-92% of the known hotspot mutations and did not call any false positives at hotspots. Accurate somatic profiling therefore requires accurate mapping in combination with a caller sensitive enough to detect low frequency variants.

FFPE DNA showed a C:G>T:A mutator profile, as described in the literature (Do et al. 2013; Do & Dobrovic 2012; Bourgon et al. 2014). These mutations are not genuine somatic events and a proportion can be removed with UDG, which increases specificity when calling variants in FFPE. By performing duplicate runs of FFPE-associated samples, we showed that ~5% of variants were consistently called; these variants were more likely to be genuine (Robasky et al. 2014).

It is possible that fixation time for FFPE samples affects overall suitability for NGS (Schweiger et al. 2009) and could very well be a contributing factor in our study, as we saw the amount of FFPE-associated artefacts focused in a subset of our samples. Initimate knowledge of the formalin fixation process applied per sample could prove useul when investigating false positive rates in NGS of FFPE samples (Lamy et al. 2011).

We also encountered A:T>G:C artefacts in our FFPE-associated samples. These artefacts likely arose due to oxidation of thymine to 5-foU (Klungland et al. 2001; Schormann et al. 2014), which is capable of pairing with guanine. Removal of A:T>G:A artefacts with UDG was not as effective as it was for C:G>T:A artefacts. This may be due to UDG's substrate specificity; however, treatment of FFPE-associated DNA with 5-foU DNA glycosylase (FDG; Matsubara et al. 2003) may result in removal of A:T>G:C artefacts. It may be beneficial for future NGS work to treat samples with FDG.

6.2.3 Key findings from Chapter 5

We noticed a high level of off-target reads in our somatic analyses. Mapping scores were generally good, and reads were mapping as paired-end reads as expected. It is possible that such off-target reads therefore represent contamination that occurred during library preparation. As contamination can occur during library preparation (Tosar et al. 2014) and sequencing itself (Dickins et al. 2014), extreme care must be taken throughout NGS protocols to ensure that accurate data are obtained. High levels of contamination (64% of the data) are not unheard of in NGS (Schmieder & Edwards 2011). A literature documenting non-human contamination is rapidly building, with evidence that projects such as the 1000 Genomes can be affected (Langdon 2014). The literature, however, provides limited bioinformatic solutions to the problem (Dickins et al. 2014; Cibulskis et al. 2011; Leggett et al. 2013).

We detected multiple variants in genes that could affect response to cetuximab. A third of the CRCs analysed in our study, that were previously believed to be wild type for *KRAS*, were found to carry *KRAS* mutations at somatic hotspots (codons 12 and 61). Furthermore, potentially novel biomarkers were discovered including a p.K5X mutation. This truncating mutation would most likely lead to loss of function as it occurs early in the ORF, although mutations at this codon have previously been associated with enhanced phosphorylation of downstream signaling molecules (Gremer et al. 2011). A p.K101X mutation in *KRAS* was found that could affect the ligand binding site and therefore downstream signalling (Buhrman et al. 2011), and a p.X190Q *KRAS* mutation is also likely to change the protein's the structure and function. In *NRAS*, that was also previously believed to be wild type, a p.G13D mutation was discovered. The earlier technologies used to profile the CRCs may not have had the sensitivity to accurately call these mutations.

A *BRAF* p.F595S mutation was discovered in the same CRC harbouring the p.K5X *KRAS* mutation. If the *KRAS* mutation leads to loss of function and hence decrease in signalling, then mutations at *BRAF* codon 595, previously believed to be activating (Kordes et al. 2015), may restore signalling and lead to lack of response in cetuximab-treatment.

Further mutations in *PIK3CA* may also be good candidates as cetuximab-resistance biomarkers. Activating mutations including p.E542K, p.E545K, p.M1043I and p.A1046T were detected in our samples that have also been indicated in the literature (S. J. Chen et al. 2015; Tian et al. 2013). The p.M1043I mutation appears in combination with p.E542K, p.E545K and truncating mutations in *PTEN*. *KRAS*, *NRAS* and *BRAF* hotspots were all wild type in this CRC, increasing the likelihood that the *PIK3CA* mutants have a role in cetuximab response.

Several candidate mutations were detected in *PTEN* that could lead to loss of function, and hence cetuximab-resistance (Dillon & Miller 2014; Molinari & Frattini 2013). Mutations include nonsense mutations at codon 72, 130, 155, 178 and 189 and a missense mutation (p.K289R) that may affect localisation of the protein (Song et al. 2012).

We also noted numerous mutations in other EGFR pathway genes that warrant further investigation for a potential role in causing resistance to cetuximab. These variants need to be screened for in CRCs of patients that did respond to cetuximab – if they are genuine biomarkers of non-response then one would expect their absence or statistical under-representation in a responsive group of patients.

We observed high levels of C:G>T:A mutations in our UDG-treated CRCs. Around half of these DNA lesions appeared in a CpG context, suggested that the affected cytosine may be methylated (Gromenko et al. 2009) and their frequency is consistent with the literature (Cooper et al. 2010). These mutations may be indicators of methylation-associated gene expression or cell proliferation in CRC (Uribe-Lewis et al. 2015) and so further investigation may yield insights into cetuximab-response.

As well as SNVs, alterations in expression levels of *EGFR* and other genes have been associated with response to cetuximab (Shigeta et al. 2013). However, the role of EGFR expression in modulating response remains controversial. Beyond the classical Ras-Raf-Mek-Erk signalling cascade, genes encoding members of seemingly unrelated pathways (such as E-cadherin) have also been suggested to alter response (Nikolova et al. 2009). It has also been suggested that mutations in other genes, such as *FCGR3A* (Pander et al. 2010), are associated with reduced PFS in patients treated with cetuximab. Furthermore, microRNAs have also been suggested to alter response: Saridaki et al. (2014) have recently associated a polymorphism in the let-7 microRNA binding site in *KRAS* with improved response to cetuximab. However, many of these alternate mechanisms require independent validation before they can be used in patient stratification.

Publications resulting from this PhD

Exome Resequencing identifies potential tumor-supressor genes that predispose to colorectal cancer

Smith CG*, Naven M*, Harris R, Colley J, West H, Li N, Liu Y, Adams R, Maughan TS, Nichols L, Kaplan R, McLeod HL & Cheadle JP.

*joint first authors.

Human Mutation (2013) 34:1026-1034

References

1000 Genomes Project Consortium et al., 2015. A global reference for human genetic variation. *Nature*, 526(7571), pp.68–74. Available at: http://www.nature.com/doifinder/10.1038/nature15393\nhttp://www.ncbi.nlm.nih.gov/pubmed/26432245.

Adams et al., 2011. Intermittent versus continuous oxaliplatin and fluoropyrimidine combination chemotherapy for first-line treatment of advanced colorectal cancer: Results of the randomised phase 3 MRC COIN trial. *The Lancet Oncology*, 12(7), pp.642–653.

von Ahlfen et al., 2007. Determinants of RNA quality from FFPE samples. *PLoS ONE*, 2(12), p.e1261.

Alexandrov et al., 2013. Signatures of mutational processes in human cancer. *Nature*, 500, pp.415–21. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3776390&tool=pmcentrez&rendertype=abstract.

Alioto et al., 2015. A comprehensive assessment of somatic mutation detection in cancer using whole-genome sequencing. *Nature Communications*, 6, p.10001. Available at: http://www.ncbi.nlm.nih.gov/pubmed/26647970.

Van Allen et al., 2014. Whole-exome sequencing and clinical interpretation of formalin-fixed, paraffin-embedded tumor samples to guide precision cancer medicine. *Nature medicine*, (April 2013). Available at: http://www.ncbi.nlm.nih.gov/pubmed/24836576 [Accessed May 23, 2014].

Aloy et al., 2006. Structural systems biology: modelling protein interactions. *Nature reviews. Molecular cell biology*, 7(3), pp.188–97. Available at: http://www.ncbi.nlm.nih.gov/pubmed/16496021.

Altimari et al., 2013. 454 next generation-sequencing outperforms allele-specific PCR, sanger sequencing, and pyrosequencing for routine KRAS mutation analysis of formalin-fixed, paraffin-embedded samples. *OncoTargets and Therapy*, 6, pp.1057–1064.

Altschul et al., 1990. Basic local alignment search tool. *Journal of molecular biology*, 215(3), pp.403–410.

Amaro, Chiara & Pfeffer, 2016. Molecular evolution of colorectal cancer: from multistep carcinogenesis to the big bang. *Cancer and Metastasis Reviews*, pp.63–74. Available at: http://link.springer.com/10.1007/s10555-016-9606-4.

Aslam et al., 2015. Identification of high-risk Dukes B colorectal cancer by microRNA expression profiling: A preliminary study. *Colorectal Disease*, 17(7), pp.578–588. Available at: http://www.embase.com/search/results?subaction=viewrecord&from=export&id=L604939178\nhttp://dx.doi.org/10.1111/codi.12886\nhttp://sfx.library.uu.nl/utrecht?sid=EMBASE&issn=14631318&id=doi:10.1111/codi.12886&atitle=Identification+of+high-risk+Dukes+B+color.

Avraham & Yarden, 2011. Feedback regulation of EGFR signalling: decision making by early and delayed loops. *Nature reviews. Molecular cell biology*, 12(2), pp.104–117.

Barras, 2015. BRAF Mutation in Colorectal Cancer: An Update. *Biomarkers in cancer*, 7(Suppl 1), pp.9–12. Available at: http://www.la-press.com/braf-mutation-in-colorectal-cancer-an-update-article-a5051.

Di Bartolomeo et al., 2014. Lack of KRAS, NRAS, BRAF and TP53 mutations improves outcome of elderly metastatic colorectal cancer patients treated with cetuximab, oxaliplatin and UFT. *Targeted Oncology*, 9(2), pp.155–162.

Bazan et al., 2005. Specific TP53 and/or Ki-ras mutations as independent predictors of clinical outcome in sporadic colorectal adenocarcinomas: Results of a 5-year Gruppo Oncologico dell'Italia Meridionale (GOIM) prospective study. *Annals of Oncology*, 16(SUPPL. 4).

Bennewith & Dedhar, 2011. Targeting hypoxic tumour cells to overcome metastasis. *BMC Cancer*, 11(1), p.504.

Bentley et al., 2008. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456(7218), pp.53–9. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2581791&tool=pmcentrez&rendertype=abstract [Accessed February 28, 2013].

Berkman et al., 2011. Sequencing and assembly of low copy and genic regions of isolated Triticum aestivum chromosome arm 7DS. *Plant Biotechnology Journal*, 9(7), pp.768–775.

Bjelland et al., 1995. Oxidation of thymine to 5-formyluracil in DNA: mechanisms of formation, structural implications, and base excision by human cell free extracts. *Biochemistry*, 34, pp.14758–14764.

Blazej, Kumaresan & Mathies, 2006. Microfabricated bioprocessor for integrated nanoliter-scale Sanger DNA sequencing. *Proceedings of the National Academy of Sciences of the United States of America*, 103(19), pp.7240–7245.

Bodini et al., 2015. The hidden genomic landscape of acute myeloid leukemia: Subclonal structure revealed by undetected mutations. *Blood*, 125(4), pp.600–605.

Boland & Goel, 2010. Microsatellite Instability in Colorectal Cancer. *Gastroenterology*, 138(6).

Bolger, Lohse & Usadel, 2014. Trimmomatic: A flexible trimmer for Illumina sequence data. *Bioinformatics*, 30(15), pp.2114–2120.

Bos et al., 1987. Prevalence of ras gene mutations in human colorectal cancers. *Nature*, 327(6120), pp.293–7. Available at: http://www.ncbi.nlm.nih.gov/pubmed/3587348.

Bourgon et al., 2014. High-throughput detection of clinically relevant mutations in archived tumor samples by multiplexed PCR and next-generation sequencing. *Clinical Cancer Research*, 20(8), pp.2080–2091.

Bowers et al., 2009. Virtual terminator nucleotides for next-generation DNA sequencing. *Nature methods*, 6(8), pp.593–595.

Brenner, Kloor & Pox, 2014. Colorectal cancer. *Lancet*, 383(9927), pp.1490–502. Available at: http://www.ncbi.nlm.nih.gov/pubmed/24225001.

Buhrman et al., 2011. Analysis of binding site hot spots on the surface of Ras GTPase. *Journal of Molecular Biology*, 413(4), pp.773–789.

Cannon-Albright et al., 2011. High quality and quantity Genome-wide germline genotypes from FFPE normal tissue. *BMC research notes*, 4(1), p.159.

Carrick et al., 2015. Robustness of next generation sequencing on older formalin-fixed paraffin-embedded tissue. *PLoS ONE*, 10(7).

Chang et al., 2016. NRAS germline variant G138R and multiple rare somatic mutations on APC in colorectal cancer patients in Taiwan by next generation sequencing.

Chen, K. et al., 2015. Clinical actionability enhanced through deep targeted sequencing of solid tumors. *Clinical Chemistry*, 61(3), pp.544–553.

Chen, 2014. DNA polymerases drive DNA sequencing-by-synthesis technologies: Both past and present. *Frontiers in Microbiology*, 5(JUN).

Chen, S.-J. et al., 2015. Ultra-deep targeted sequencing of advanced oral squamous cell carcinoma identifies a mutation-based prognostic gene signature. *Oncotarget 18066 Oncotarget*, 6(20). Available at: www.impactjournals.com/oncotarget\nwww.impactjournals.com/oncotarget/.

Chin, da Silva & Hegde, 2013. Assessment of clinical analytical sensitivity and specificity of next-generation sequencing for detection of simple and complex mutations. *BMC genetics*, 14(1), p.6. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3599218&tool=pmcentrez&rendertype=abstract.

Choi et al., 2013. Revisiting the ventral medial nucleus of the hypothalamus: The roles of SF-1 neurons in energy homeostasis. *Frontiers in Neuroscience*, 7, pp.1–9.

Choudhary et al., 2012. Proliferation rate but not mismatch repair affects the long-term response of colon carcinoma cells to 5FU treatment. *Cancer Letters*, 320(1), pp.56–64.

Cibulskis et al., 2011. ContEst: Estimating cross-contamination of human samples in next-generation sequencing data. *Bioinformatics*, 27(18), pp.2601–2602.

Cibulskis et al., 2013. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nature biotechnology*, 31(3), pp.213–9. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3833702&tool=pmcentrez&rendertype=abstract [Accessed February 28, 2013].

Cingolani et al., 2012. A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain w 1118; iso-2; iso-3. *Fly*, 6(2), pp.80–92. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3679285&tool=pmcentrez&rendertype=abstract.

Cisyk et al., 2015. Characterizing the prevalence of chromosome instability in interval colorectal cancer. *Neoplasia (New York, N.Y.)*, 17(3), pp.306–316.

Clarke et al., 2009. Continuous base identification for single-molecule nanopore DNA sequencing. *Nature nanotechnology*, 4(4), pp.265–270.

Collins, Morgan & Patrinos, 2003. The Human Genome Project: lessons from large-scale biology. *Science (New York, N.Y.)*, 300(5617), pp.286–290.

Cooper et al., 2010. Methylation-mediated deamination of 5-methylcytosine appears to give rise to mutations causing human inherited disease in CpNpG trinucleotides, as well as in CpG dinucleotides. *Human Genomics*, 4(6), pp.406–10. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3525222&tool=pmcentrez&rendertype=abstract.

Costello et al., 2013. Discovery and characterization of artifactual mutations in deep coverage targeted capture sequencing data due to oxidative DNA damage during sample preparation. *Nucleic acids research*, pp.1–12. Available at: http://www.ncbi.nlm.nih.gov/pubmed/23303777 [Accessed March 1, 2013].

Cross, Start & Smith, 1990. Does delay in fixation affect the number of mitotic figures in processed tissue? *Journal of clinical pathology*, 43(7), pp.597–599.

Van Cutsen et al., 2001. Oral capecitabine compared with intravenous fluorouracil plus leucovorin in patients with metastatic colorectal cancer: Results of a large phase III study. *Journal of Clinical Oncology*, 19(21), pp.4097–4106.

David et al., 2011. SHRiMP2: Sensitive yet practical short read mapping. *Bioinformatics*, 27(7), pp.1011–1012.

Delattre et al., 1989. Multiple genetic alterations in distal and proximal colorectal cancer. *Lancet*, 2(8659), pp.353–356.

DePristo et al., 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics*, 43(5), pp.491–8. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3083463&tool=pmcentrez&rendertype=abstract [Accessed February 28, 2013].

Dickins et al., 2014. Controlling for contamination in re-sequencing studies with a reproducible web-based phylogenetic approach. *BioTechniques*, 56(3), pp.134–141.

Dillon & Miller, 2014. Therapeutic targeting of cancers with loss of PTEN function. *Current drug targets*, 15(1), pp.65–79. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=4310752&tool=pmcentrez&rendertype=abstract.

Ding et al., 2015. Meta-Analysis of the association between APC promoter methylation and colorectal cancer. *OncoTargets and Therapy*, 8, pp.211–222.

Do et al., 2013. Reducing sequence artifacts in amplicon-based massively parallel sequencing of formalin-fixed paraffin-embedded DNA by enzymatic depletion of uracil-containing templates. *Clinical Chemistry*, 59(9), pp.1376–1383.

Do & Dobrovic, 2012. Dramatic reduction of sequence artefacts from DNA isolated from formalin-fixed cancer biopsies by treatment with uracil-DNA glycosylase. *Oncotarget*, 3(5), pp.546–558. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3388184&tool=pmcentrez&rendertype=abstract.

Dohm et al., 2007. SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Research*, 17(11), pp.1697–1706.

Dohm et al., 2008. Substantial biases in ultra-short read data sets from high-throughput DNA sequencing. *Nucleic Acids Research*, 36(16).

Douglas & Rogers, 1998. DNA damage caused by common cytological fixatives. *Mutation Research - Fundamental and Molecular Mechanisms of Mutagenesis*, 401(1-2), pp.77–88.

Douillard et al., 2000. *Irinotecan combined with fluorouracil compared with* *fluorouracil alone as first-line treatment for metastatic colorectal cancer: a multicentre randomised trial.*,

Dow et al., 2015. Apc Restoration Promotes Cellular Differentiation and Reestablishes Crypt Homeostasis in Colorectal Cancer. *Cell*, 161(7), pp.1539–1552. Available at: http://www.sciencedirect.com/science/article/pii/S0092867415006236.

Drmanac et al., 1998. Accurate sequencing by hybridization for DNA diagnostics and individual genomics. *Nature biotechnology*, 16(1), pp.54–58.

Drmanac et al., 1989. Sequencing of megabase plus DNA by hybridization: theory of the method. *Genomics*, 4(2), pp.114–128.

Elloumi et al., 2011. Systematic bias in genomic classification due to contaminating non-neoplastic tissue in breast tumor samples. *BMC Med Genomics*, 4, p.54. Available at: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3151208/pdf/1755-8794-4-54.pdf.

Emrich et al., 2002. Microfabricated 384-lane capillary array electrophoresis bioanalyzer for ultrahigh-throughput genetic analysis. *Analytical Chemistry*, 74(19), pp.5076–5083.

Erlich et al., 2008. Alta-Cyclic: a self-optimizing base caller for next-generation sequencing. *Nature methods*, 5(8), pp.679–682.

Esposito et al., 2013. The S492R EGFR ectodomain mutation is never detected in KRAS wild-type colorectal carcinoma before exposure to EGFR monoclonal antibodies. *Cancer Biology and Therapy*, 14(12), pp.1143–1146.

EVS, 2014. Exome Variant Server. *NHLBI GO Exome Sequencing Project (ESP)*. Available at: http://evs.gs.washington.edu/EVS/.

Ewing et al., 1998. Base-calling of automated sequencer traces usingPhred. I. Accuracy assessment. *Genome research*, pp.175–185. Available at: http://genome.cshlp.org/content/8/3/175.short.

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

Fang et al., 2014. Reducing INDEL calling errors in whole-genome and exome sequencing data. *Genome Medicine*, 6(10), p.89. Available at: http://genomemedicine.com/content/6/10/89/abstract\nhttp://genomemedicine.com/content/pdf/s13073-014-0089-z.pdf.

Fearnhead, Britton & Bodmer, 2001. The ABC of APC. *Human molecular genetics*, 10(7), pp.721–733.

Fearon & Vogelstein, 1990. A genetic model for colorectal tumorigenesis. *Cell*, 61(5), pp.759–767.

Fedurco et al., 2006. BTA, a novel reagent for DNA attachment on glass and efficient generation of solid-phase amplified DNA colonies. *Nucleic Acids Research*, 34(3).

Fernandez & Torres-Alemán, 2012. The many faces of insulin-like peptide signalling in the brain. *Nature Reviews Neuroscience*, 13(4), pp.225–239.

Flicek et al., 2013. Ensembl 2013. *Nucleic Acids Research*, 41(D1).

Fodde, Smits & Clevers, 2001. APC, signal transduction and genetic instability in colorectal cancer. *Nature reviews. Cancer*, 1(1), pp.55–67. Available at: http://www.ncbi.nlm.nih.gov/pubmed/11900252.

Fonseca et al., 2012. Tools for mapping high-throughput sequencing data. *Bioinformatics*, 28(24), pp.3169–3177.

Forbes et al., 2008. The catalogue of somatic mutations in cancer (COSMIC). *Current Protocols in Human Genetics*, (SUPPL. 57).

Fraenkel-Conrat & Olcott, 1948. The reaction of formaldehyde with proteins; cross-linking between amino and primary amide or guanidyl groups. *Journal of the American Chemical Society*, 70(8), pp.2673–2684.

Frattini et al., 2007. PTEN loss of expression predicts cetuximab efficacy in metastatic colorectal cancer patients. *British journal of cancer*, 97(8), pp.1139–1145.

Fruman & Rommel, 2014. PI3K and cancer: lessons, challenges and opportunities. *Nature reviews. Drug discovery*, 13(2), pp.140–56. Available at: http://www.ncbi.nlm.nih.gov/pubmed/24481312.

Fujita et al., 2016. Detection of low-prevalence somatic TSC2 mutations in sporadic pulmonary lymphangioleiomyomatosis tissues by deep sequencing. *Human Genetics*, 135(1), pp.61–68.

Futreal et al., 2004. A census of human cancer genes. *Nature reviews. Cancer*, 4(3), pp.177–83. Available at: http://dx.doi.org/10.1038/nrc1299.

George et al., 1997. Capillary electrophoresis methodology for identification of cancer related gene expression patterns of fluorescent differential display polymerase chain reaction. *J.Chromatogr.B Biomed.Sci.Appl.*, 695(1387-2273 SB - IM), pp.93–102.

Gerlinger et al., 2012. Intratumor Heterogeneity and Branched Evolution Revealed by Multiregion Sequencing. *New England Journal of Medicine*, 366(10), pp.883–892.

Giacchetti et al., 2000. Phase III multicenter randomized trial of oxaliplatin added to chronomodulated fluorouracil-leucovorin as first-line treatment of metastatic colorectal cancer. *Journal of Clinical Oncology*, 18(1), pp.136–147.

Giardine et al., 2005. Galaxy: A platform for interactive large-scale genome analysis. *Genome Research*, 15(10), pp.1451–1455.

Goecks, Nekrutenko & Taylor, 2010. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 11(8), p.R86.

Goldstein & Armin, 2001. Epidermal growth factor receptor immunohistochemical reactivity in patients with American Joint Committee on Cancer Stage IV colon adenocarcinoma: implications for a standardized scoring system. *Cancer*, 92(5), pp.1331–1346.

Green et al., 2008. A Complete Neandertal Mitochondrial Genome Sequence Determined by High-Throughput Sequencing. *Cell*, 134(3), pp.416–426.

Green et al., 2006. Analysis of one million base pairs of Neanderthal DNA. *Nature*, 444(7117), pp.330–336.

Gremer et al., 2011. Germline KRAS mutations cause aberrant biochemical and physical properties leading to developmental disorders. *Human Mutation*, 32(1), pp.33–43.

Griffioen & Molema, 2000. Angiogenesis: potentials for pharmacologic intervention in the treatment of cancer, cardiovascular diseases, and chronic inflammation. *Pharmacological reviews*, 52(2), pp.237–268.

Gromenko et al., 2009. Deamination of 5-Methylcytosine Residues in Mammalian Cells. *Acta Naturae*, 3, pp.121–124.

Guo et al., 2012. The effect of strand bias in Illumina short-read sequencing data. *BMC genomics*, 13, p.666. Available at: http://www.biomedcentral.com/1471-2164/13/666.

Hadd et al., 2013. Targeted, high-depth, next-generation sequencing of cancer genes in formalin-fixed, paraffin-embedded and fine-needle aspiration tumor specimens. *The Journal of molecular diagnostics : JMD*, 15(2), pp.234–47. Available at: http://www.ncbi.nlm.nih.gov/pubmed/23321017 [Accessed March 7, 2013].

Haggar & Boushey, 2009. Colorectal cancer epidemiology: Incidence, mortality, survival, and risk factors. *Clinics in Colon and Rectal Surgery*, 22(4), pp.191–197.

Hansen et al., 2013. Shimmer: Detection of genetic alterations in tumors using next generation sequence data. *Bioinformatics (Oxford, England)*, pp.1–6.

Available at: http://www.ncbi.nlm.nih.gov/pubmed/23620360.

Harismendy et al., 2011. Detection of low prevalence somatic mutations in solid tumors with ultra-deep targeted sequencing. *Genome biology*, 12(12), p.R124. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3334619&tool=pmcentrez&rendertype=abstract [Accessed March 11, 2013].

Harris et al., 2008. Single-molecule DNA sequencing of a viral genome. *Science (New York, N.Y.)*, 320(5872), pp.106–109.

Healey, Matson & Walt, 1997. Fiberoptic DNA sensor array capable of detecting point mutations. *Analytical biochemistry*, 251(2), pp.270–279.

Hedegaard et al., 2014. Next-generation sequencing of RNA and DNA isolated from paired fresh-frozen and formalin-fixed paraffin-embedded samples of human cancer and normal tissue. *PLoS ONE*, 9(5).

Hert, Fredlake & Barron, 2008. Advantages and limitations of next-generation sequencing technologies: A Comparison of electrophoresis and non-electrophoresis methods. *Electrophoresis*, 29(23), pp.4618–4626.

Hoff et al., 2001. *Comparison of oral capecitabine versus intravenous fluorouracil plus leucovorin as first-line treatment in 605 patients with metastatic colorectal cancer: results of a randomized phase III study.*,

Hofreiter, Serre, et al., 2001. Ancient DNA. *Nature reviews. Genetics*, 2(5), pp.353–359.

Hofreiter, Jaenicke, et al., 2001. DNA sequences from multiple amplifications reveal artifacts induced by cytosine deamination in ancient DNA. *Nucleic acids research*, 29(23), pp.4793–4799.

Holohan et al., 2013. Cancer drug resistance: an evolving paradigm. *Nature reviews. Cancer*, 13(10), pp.714–26. Available at: http://www.ncbi.nlm.nih.gov/pubmed/24060863.

Homer, Merriman & Nelson, 2009. BFAST: An alignment tool for large scale genome resequencing. *PLoS ONE*, 4(11).

Horvat & Stabuc, 2011. Microsatellite instability in colorectal cancer. *Radiology and Oncology*, 45(2), pp.75–81.

Huang, Quesada & Mathies, 1992. DNA sequencing using capillary array electrophoresis. *Analytical chemistry*, 64(18), pp.2149–2154.

Hynes & Lane, 2005. ERBB receptors and cancer: the complexity of targeted inhibitors. *Nature reviews. Cancer*, 5(5), pp.341–354.

Ihle et al., 2014. Comparison of high resolution melting analysis, pyrosequencing, next generation sequencing and immunohistochemistry to conventional Sanger sequencing for the detection of p.V600E and non-p.V600E BRAF mutations. *BMC cancer*, 14, p.13. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3893431&tool=pmcentrez&rendertype=abstract.

Imelfort & Edwards, 2009. De novo sequencing of plant genomes using second-generation technologies. *Briefings in Bioinformatics*, 10(6), pp.609–618.

Jeck et al., 2007. Extending assembly of short DNA sequences to handle error. *Bioinformatics*, 23(21), pp.2942–2944.

Jhawer et al., 2008. PIK3CA mutation/PTEN expression status predicts response of colon cancer cells to the epidermal growth factor receptor inhibitor cetuximab. *Cancer Research*, 68(6), pp.1953–1961.

Johnston & Kaye, 2001. Capecitabine: a novel agent for the treatment of solid tumors. *Anti-cancer drugs*, 12(8), pp.639–646. Available at: http://ovidsp.ovid.com/ovidweb.cgi?T=JS&PAGE=reference&D=emed5&NEWS=N&AN=2001320341.

Kanno et al., 2000. Roles of two VEGF receptors, Flt-1 and KDR, in the signal transduction of VEGF effects in human vascular endothelial cells. *Oncogene*, 19(17), pp.2138–2146.

Karapetis et al., 2008. *K-ras mutations and benefit from cetuximab in advanced colorectal cancer.*,

Karolchik et al., 2003. The UCSC Genome Browser Database. *Nucleic Acids Research*, 31(1), pp.51–54.

Kasai et al., 1990. 5-Formyldeoxyuridine: a new type of DNA damage induced by ionizing radiation and its mutagenicity to salmonella strain TA102. *Mutation research*, 243, pp.249–253.

Kelly & Cassidy, 2007. Chemotherapy in metastatic colorectal cancer. *Surgical Oncology*, 16(1), pp.65–70.

Kent, 2002. BLAT - The BLAST-like alignment tool. *Genome Research*, 12(4), pp.656–664.

Kerick et al., 2011. Targeted high throughput sequencing in clinical cancer settings: formaldehyde fixed-paraffin embedded (FFPE) tumor tissues, input amount and tumor heterogeneity. *BMC medical genomics*, 4(1), p.68. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3192667&tool=pmcentrez&rendertype=abstract [Accessed March 11, 2013].

Kim, Yoo & Hahn, 1996. Postelectrophoresis capillary scanning method for DNA sequencing. *Analytical chemistry*, 68(5), pp.936–939.

Kircher & Kelso, 2010. High-throughput DNA sequencing--concepts and limitations. *BioEssays : news and reviews in molecular, cellular and developmental biology*, 32(6), pp.524–536.

Kircher, Stenzel & Kelso, 2009. Improved base calling for the Illumina Genome Analyzer using machine learning strategies. *Genome biology*, 10(8), p.R83.

Klungland et al., 2001. 5-Formyluracil and its nucleoside derivatives confer toxicity and mutagenicity to mammalian cells by interfering with normal RNA and DNA metabolism. *Toxicology Letters*, 119, pp.71–78.

Knudson, 1971. Mutation and cancer: statistical study of retinoblastoma. *Proceedings of the National Academy of Sciences of the United States of America*, 68(4), pp.820–823.

Kobayashi et al., 2005. *EGFR mutation and resistance of non-small-cell lung cancer to gefitinib.*,

Koboldt et al., 2012. VarScan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing. *Genome Research*, 22(3), pp.568–576.

Koboldt et al., 2009. VarScan: Variant detection in massively parallel sequencing of individual and pooled samples. *Bioinformatics*, 25(17), pp.2283–2285.

Kokkat et al., 2013. Archived formalin-fixed paraffin-embedded (FFPE) blocks: A valuable underexploited resource for extraction of DNA, RNA, and protein. *Biopreservation and biobanking*, 11(2), pp.101–6. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=4077003&tool=pmcentrez&rendertype=abstract.

Kołos, Wasążnik-Jędras & Nasierowska-Guttmejer, 2015. Can the histological type of colorectal cancer determine the carcinogenesis pathway? *Polish journal of pathology : official journal of the Polish Society of Pathologists*, 66(2), pp.109–20. Available at: http://www.ncbi.nlm.nih.gov/pubmed/26247523.

Kordes et al., 2015. Cooperation of BRAF(F595L) and mutant HRAS in histiocytic sarcoma provides new insights into oncogenic BRAF signaling. *Leukemia*, (October 2015), pp.937–946. Available at: http://www.ncbi.nlm.nih.gov/pubmed/26582644.

Korlach et al., 2008. Selective aluminum passivation for targeted immobilization of single DNA polymerase molecules in zero-mode waveguide nanostructures. *Proceedings of the National Academy of Sciences of the United States of America*, 105(4), pp.1176–1181.

Krasinskas, 2011. EGFR Signaling in Colorectal Carcinoma. *Pathology research international*, 2011, p.932932. Available at:

http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3042643&tool=pmcentrez&rendertype=abstract [Accessed March 11, 2013].

Van Krieken et al., 2016. RAS testing in metastatic colorectal cancer: advances in Europe. *Virchows Archiv*, 468(4), pp.383–396.

Krøigård et al., 2016. Evaluation of Nine Somatic Variant Callers for Detection of Somatic Mutations in Exome and Targeted Deep Sequencing Data. *PLOS ONE*, 11(3), p.e0151664. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=4803342&tool=pmcentrez&rendertype=abstract.

Krokan, Drabløs & Slupphaug, 2002. Uracil in DNA--occurrence, consequences and repair. *Oncogene*, 21, pp.8935–8948.

Krokan, Standal & Slupphaug, 1997. DNA glycosylases in the base excision repair of DNA. *Biochemistry Journal*, 325, pp.1–16. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1218522&tool=pmcentrez&rendertype=abstract.

Kwak et al., 2005. Irreversible inhibitors of the EGF receptor may circumvent acquired resistance to gefitinib. *Proceedings of the National Academy of Sciences of the United States of America*, 102(21), pp.7665–7670.

de la Chapelle, 2004. Genetic predisposition to colorectal cancer. *Nature reviews. Cancer*, 4(10), pp.769–80. Available at: http://www.ncbi.nlm.nih.gov/pubmed/15510158 [Accessed March 7, 2013].

Lam et al., 2012. Detecting and annotating genetic variations using the HugeSeq pipeline. *Nature biotechnology*, 30(3), pp.226–9. Available at: http://www.ncbi.nlm.nih.gov/pubmed/22398614 [Accessed March 1, 2013].

Lamy et al., 2011. Metastatic colorectal cancer KRAS genotyping in routine practice: results and pitfalls. *Modern pathology : an official journal of the United States and Canadian Academy of Pathology, Inc*, 24(8), pp.1090–1100.

Lander et al., 2001. Initial sequencing and analysis of the human genome. *Nature*, 409(6822), pp.860–921.

Langdon, 2014. Mycoplasma contamination in the 1000 Genomes Project. *BioData mining*, 7, p.3. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=4022254&tool=pmcentrez&rendertype=abstract.

Langmead et al., 2009. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome biology*, 10(3), p.R25. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2690996&tool=pmcentrez&rendertype=abstract [Accessed February 28, 2013].

Larson et al., 2012. Somaticsniper: Identification of somatic point mutations in whole genome sequencing data. *Bioinformatics*, 28(3), pp.311–317.

Lee et al., 2013. A new assay for measuring chromosome instability (CIN) and identification of drugs that elevate CIN in cancer cells. *BMC cancer*, 13, p.252. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3671967&tool=pmcentrez&rendertype=abstract.

Lee et al., 2012. Bioinformatics tools and databases for analysis of next-generation sequence data. *Briefings in Functional Genomics*, 11(1), pp.12–24.

Leggett et al., 2013. Sequencing quality assessment tools to enable data-driven informatics for high throughput genomics. *Frontiers in Genetics*, 4(DEC).

Li et al., 2013. Perturbation of the mutated EGFR interactome identifies vulnerabilities and resistance mechanisms. *Molecular systems biology*, 9(705), p.705. Available at: http://www.ncbi.nlm.nih.gov/pubmed/24189400.

Li, R. et al., 2008. SOAP: Short oligonucleotide alignment program. *Bioinformatics*, 24(5), pp.713–714.

Li, R. et al., 2009. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics (Oxford, England)*, 25(15), pp.1966–7. Available at: http://www.ncbi.nlm.nih.gov/pubmed/19497933 [Accessed February 27, 2013].

Li, H. et al., 2009. The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, 25(16), pp.2078–9. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2723002&tool=pmcentrez&rendertype=abstract [Accessed February 27, 2013].

Li, 2014. Toward better understanding of artifacts in variant calling from high-coverage samples. *Bioinformatics (Oxford, England)*, 30(20), pp.2843–2851.

Li & Durbin, 2009. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics (Oxford, England)*, 25(14), pp.1754–60. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2705234&tool=pmcentrez&rendertype=abstract [Accessed February 27, 2013].

Li, H., Ruan & Durbin, 2008. Mapping short DNA sequencing reads and variants calling using mapping quality scores (Supplementary Text). *Genome research*, 18(11), pp.1–8. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2577856&tool=pmcentrez&rendertype=abstract.

Lièvre et al., 2006. KRAS mutation status is predictive of response to cetuximab therapy in colorectal cancer. *Cancer research*, 66(8), pp.3992–5. Available at: http://www.ncbi.nlm.nih.gov/pubmed/16618717 [Accessed March 1, 2013].

Lin et al., 2011. Comparative studies of de novo assembly tools for next-generation sequencing technologies. *Bioinformatics (Oxford, England)*, 27(15), pp.2031–2037.

Liu, C.-M. et al., 2012. SOAP3: ultra-fast GPU-based parallel alignment tool for short reads. *Bioinformatics (Oxford, England)*, 28(6), pp.878–9. Available at: http://www.ncbi.nlm.nih.gov/pubmed/22285832 [Accessed March 11, 2013].

Liu, Q. et al., 2012. Steps to ensure accuracy in genotype and SNP calling from Illumina sequencing data. *BMC genomics*, 13 Suppl 8(Suppl 8), p.S8. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3535703&tool=pmcentrez&rendertype=abstract.

Longley, Harkin & Johnston, 2003. 5-fluorouracil: mechanisms of action and clinical strategies. *Nature reviews. Cancer*, 3(5), pp.330–338.

Loupakis et al., 2009. KRAS codon 61, 146 and BRAF mutations predict resistance to cetuximab plus irinotecan in KRAS codon 12 and 13 wild-type metastatic colorectal cancer. *British journal of cancer*, 101(4), pp.715–21. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2736831&tool=pmcentrez&rendertype=abstract [Accessed March 3, 2013].

Lu et al., 2016. Primary tumor location is an important predictive factor for wild-type KRAS metastatic colon cancer treated with cetuximab as front-line bio-therapy. *Asia-Pacific Journal of Clinical Oncology*.

Luo et al., 2012. RET is a potential tumor suppressor gene in colorectal cancer. *Oncogene*, (January 2011), pp.1–11. Available at: http://www.ncbi.nlm.nih.gov/pubmed/22751117 [Accessed March 11, 2013].

Lynch 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, pp.2129–2139. Available at: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=15118073.

March et al., 2011. Insertional mutagenesis identifies multiple networks of co-operating genes driving intestinal tumorigenesis. *Nature Genetics*, 43(12), pp.1202–1209.

Margulies et al., 2005. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057), pp.376–380.

Mariella, 2008. Sample preparation: The weak link in microfluidics-based biodetection. *Biomedical Microdevices*, 10(6), pp.777–784.

Masaoka et al., 2001. Oxidation of Thymine to 5-Formyluracil in DNA Promotes Misincorporation of dGMP and Subsequent Elongation of a Mismatched Primer Terminus by DNA Polymerase. *Journal of Biological Chemistry*, 276, pp.16501–16510.

Matsubara et al., 2003. Mammalian 5-formyluracil-DNA glycosylase. 1. Identification and characterization of a novel activity that releases 5-formyluracil from DNA. *Biochemistry*, 42(17), pp.4993–5002.

Maughan et al., 2011. Addition of cetuximab to oxaliplatin-based first-line combination chemotherapy for treatment of advanced colorectal cancer: results of the randomised phase 3 MRC COIN trial. *Lancet*, 377(9783), pp.2103–14. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3159415&tool=pmcentrez&rendertype=abstract [Accessed March 6, 2013].

McKay et al., 2002. Evaluation of the epidermal growth factor receptor (EGFR) in colorectal tumours and lymph node metastases. *European Journal of Cancer*, 38(17), pp.2258–2264.

McKenna et al., 2010. The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, 20(9), pp.1297–1303.

Meriggi et al., 2014. The Emerging Role of NRAS Mutations in Colorectal Cancer Patients Selected for Anti-EGFR Therapies. *Reviews on recent clinical trials*, pp.35–39. Available at: http://www.ncbi.nlm.nih.gov/pubmed/24758538.

Metzker, 2010. Sequencing technologies - the next generation. *Nature reviews. Genetics*, 11(1), pp.31–46. Available at: http://www.ncbi.nlm.nih.gov/pubmed/19997069 [Accessed May 21, 2013].

Mitsudomi & Yatabe, 2010. Epidermal growth factor receptor in relation to tumor development: EGFR gene and cancer. *The FEBS journal*, 277(2), pp.301–308.

Molinari & Frattini, 2013. Functions and Regulation of the PTEN Gene in Colorectal Cancer. *Frontiers in oncology*, 3(January), p.326. Available at: http://journal.frontiersin.org/article/10.3389/fonc.2013.00326/abstract.

Montagut et al., 2012. Identification of a mutation in the extracellular domain of the Epidermal Growth Factor Receptor conferring cetuximab resistance in colorectal cancer. *Nature Medicine*, 18(9), pp.1445–1445.

Moroni et al., 2005. *Gene copy number for epidermal growth factor receptor (EGFR) and clinical response to antiEGFR treatment in colorectal cancer: a* *cohort study.*,

Nakabeppu, 2014. Cellular levels of 8-oxoguanine in either DNA or the nucleotide pool play pivotal roles in carcinogenesis and survival of cancer cells. *International Journal of Molecular Sciences*, 15(7), pp.12543–12557.

Ng et al., 2009. Targeted capture and massively parallel sequencing of 12 human exomes. *Nature*, 461(7261), pp.272–6. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2844771&tool=pmcentrez&rendertype=abstract [Accessed February 28, 2013].

Di Nicolantonio et al., 2008. Wild-type BRAF is required for response to panitumumab or cetuximab in metastatic colorectal cancer. *Journal of clinical oncology : official journal of the American Society of Clinical Oncology*, 26(35), pp.5705–12. Available at: http://www.ncbi.nlm.nih.gov/pubmed/19001320 [Accessed March 3, 2013].

Nigro et al., 1989. Mutations in the p53 gene occur in diverse human tumour types. *Nature*, 342(6250), pp.705–708.

Nikolova et al., 2009. Cetuximab attenuates metastasis and u-PAR expression in non-small cell lung cancer: U-PAR and E-cadherin are novel biomarkers of cetuximab sensitivity. *Cancer Research*, 69(6), pp.2461–2470.

Obrand & Gordon, 1997. Incidence and patterns of recurrence following curative resection for colorectal carcinoma. *Diseases of the colon and rectum*, 40(1), pp.15–24.

Oda et al., 2005. A comprehensive pathway map of epidermal growth factor receptor signaling. *Molecular systems biology*, 1, p.2005.0010. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1681468&tool=pmcentrez&rendertype=abstract [Accessed March 4, 2013].

Okello et al., 2010. Comparison of methods in the recovery of nucleic acids from archival formalin-fixed paraffin-embedded autopsy tissues. *Analytical biochemistry*, 400(1), pp.110–7. Available at: http://www.ncbi.nlm.nih.gov/pubmed/20079706 [Accessed March 11, 2013].

Olivier, Hollstein & Hainaut, 2010. TP53 mutations in human cancers: origins, consequences, and clinical use. *Cold Spring Harbor perspectives in biology*, 2(1).

Organ & Tsao, 2011. An overview of the c-MET signaling pathway. *Therapeutic Advances in Medical Oncology*, 3(1 Suppl), pp.S7–S19.

Pander et al., 2010. Correlation of FCGR3A and EGFR germline polymorphisms with the efficacy of cetuximab in KRAS wild-type metastatic colorectal cancer. *European Journal of Cancer*, 46(10), pp.1829–1834.

Pannett & Thakker, 2001. Somatic mutations in MEN type 1 tumors, consistent with the Knudson “two-hit” hypothesis. *Journal of Clinical Endocrinology and Metabolism*, 86(9), pp.4371–4374.

Pao et al., 2005. Acquired resistance of lung adenocarcinomas to gefitinib or erlotinib is associated with a second mutation in the EGFR kinase domain. *PLoS Medicine*, 2, pp.0225–0235.

Park et al., 2014. Comprehensive analysis to improve the validation rate for single nucleotide variants detected by next-generation sequencing. *PLoS ONE*, 9(1).

Park et al., 2016. UNDR ROVER - a fast and accurate variant caller for targeted DNA sequencing. *BMC bioinformatics*, 17(1), p.165. Available at: http://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-016-1014-9.

Parsons et al., 2005. Colorectal cancer: mutations in a signalling pathway. *Nature*, 436(7052), p.792. Available at: http://www.ncbi.nlm.nih.gov/pubmed/16094359 [Accessed March 11, 2013].

Phelps et al., 2009. A Two-Step Model for Colon Adenoma Initiation and Progression Caused by APC Loss. *Cell*, 137(4), pp.623–634.

Pihan, 2013. Centrosome dysfunction contributes to chromosome instability, chromoanagenesis, and genome reprograming in cancer. *Frontiers in oncology*, 3(November), p.277. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3824400&tool=pmcentrez&rendertype=abstract.

Pino & Chung, 2010. The Chromosomal Instability Pathway in Colon Cancer. *Gastroenterology*, 138(6), pp.2059–2072.

Postma et al., 2005. Chromosomal instability in flat adenomas and carcinomas of the colon. *Journal of Pathology*, 205(4), pp.514–521.

Powell et al., 1992. APC mutations occur early during colorectal tumorigenesis. *Nature*, 359(6392), pp.235–237.

Prenen et al., 2009. PIK3CA mutations are not a major determinant of resistance to the epidermal growth factor receptor inhibitor cetuximab in metastatic colorectal cancer. *Clinical Cancer Research*, 15(9), pp.3184–3188.

Pruitt et al., 2009. The consensus coding sequence (CCDS) project: Identifying a common protein-coding gene set for the human and mouse genomes. *Genome Research*, 19(7), pp.1316–1323.

Pruitt, Tatusova & Maglott, 2007. NCBI reference sequences (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic acids research*, 35(Database issue), pp.D61–5. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1716718&tool=pmcentrez&rendertype=abstract [Accessed February 28, 2013].

Pushkarev, Neff & Quake, 2009. Single-molecule sequencing of an individual human genome. *Nature biotechnology*, 27(9), pp.847–850.

Qiu et al., 2016. Data Interoperability of Whole Exome Sequencing (WES) Based Mutational Burden Estimates from Different Laboratories. *International Journal of Molecular Sciences*, 17(5), p.651. Available at: http://www.mdpi.com/1422-0067/17/5/651.

Quinlan et al., 2008. Pyrobayes: an improved base caller for SNP discovery in pyrosequences. *Nature methods*, 5(2), pp.179–181.

Resnick et al., 2004. Epidermal growth factor receptor, c-MET, beta-catenin, and p53 expression as prognostic indicators in stage II colon cancer: a tissue microarray study. *Clinical cancer research : an official journal of the American Association for Cancer Research*, 10(9), pp.3069–3075.

Robasky, Lewis & Church, 2014. The role of replicates for error mitigation in next-generation sequencing. *Nature reviews. Genetics*, 15(1), pp.56–62. Available at: http://www.ncbi.nlm.nih.gov/pubmed/24322726.

Robinson et al., 2011. Integrative genomics viewer. *Nature biotechnology*, 29(1), pp.24–26.

Ronaghi et al., 1996. Real-time DNA sequencing using detection of pyrophosphate release. *Analytical biochemistry*, 242(1), pp.84–89.

De Roock, Jonker, et al., 2010. Association of KRAS p.G13D mutation with outcome in patients with chemotherapy-refractory metastatic colorectal cancer treated with cetuximab. *JAMA : the journal of the American Medical Association*, 304(16), pp.1812–1820.

De Roock, Claes, et al., 2010. Effects of KRAS, BRAF, NRAS, and PIK3CA mutations on the efficacy of cetuximab plus chemotherapy in chemotherapy-refractory metastatic colorectal cancer: a retrospective consortium analysis. *The lancet oncology*, 11(8), pp.753–62. Available at: http://www.ncbi.nlm.nih.gov/pubmed/20619739 [Accessed March 4, 2013].

Roper et al., 2007. Infrared temperature control system for a completely noncontact polymerase chain reaction in microfluidic chips. *Analytical Chemistry*, 79(4), pp.1294–1300.

Roth et al., 2012. JointSNVMix: A probabilistic model for accurate detection of somatic mutations in normal/tumour paired next-generation sequencing data. *Bioinformatics*, 28(7), pp.907–913.

Rothenberg et al., 1999. *A multicenter, phase II trial of weekly irinotecan (CPT-*

11) in patients with previously treated colorectal carcinoma.,

Rougemont et al., 2008. Probabilistic base calling of Solexa sequencing data. *BMC bioinformatics*, 9, p.431.

Ruffalo, LaFramboise & Koyutürk, 2011. Comparative analysis of algorithms for next-generation sequencing read alignment. *Bioinformatics (Oxford, England)*, 27(20), pp.2790–6. Available at: http://www.ncbi.nlm.nih.gov/pubmed/21856737 [Accessed March 6, 2013].

Rumble et al., 2009. SHRiMP: Accurate mapping of short color-space reads. *PLoS Computational Biology*, 5(5).

Sakai et al., 2015. Extended RAS and BRAF mutation analysis using next-generation sequencing. *PLoS ONE*, 10(5).

Samowitz et al., 2007. APC mutations and other genetic and epigenetic changes in colon cancer. *Molecular cancer research : MCR*, 5(2), pp.165–170.

Sampson et al., 2005. MutYH (MYH) and colorectal cancer. *Biochemical Society transactions*, 33(Pt 4), pp.679–683.

Sanger, Air, et al., 1977. Nucleotide sequence of bacteriophage phi X174 DNA. *Nature*, 265(5596), pp.687–695.

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

Saridaki et al., 2014. A let-7 microRNA-binding site polymorphism in KRAS predicts improved outcome in metastatic colorectal cancer (mCRC) patients treated with salvage cetuximab/panitumumab monotherapy. *Clinical Cancer Research*, 20(17), pp.4499–4510.

Sartore-Bianchi et al., 2009. PIK3CA mutations in colorectal cancer are associated with clinical resistance to EGFR-targeted monoclonal antibodies. *Cancer Research*, 69(5), pp.1851–1857.

Sassolas, Leca-Bouvier & Blum, 2008. DNA biosensors and microarrays. *Chemical Reviews*, 108(1), pp.109–139.

Saunders et al., 2012. Strelka: Accurate somatic small-variant calling from sequenced tumor-normal sample pairs. *Bioinformatics*, 28(14), pp.1811–1817.

Schmidt et al., 2014. Stat5 regulates the phosphatidylinositol 3-kinase/Akt1 pathway during mammary gland development and tumorigenesis. *Molecular and cellular biology*, 34(7), pp.1363–77. Available at: http://www.ncbi.nlm.nih.gov/pubmed/24469394.

Schmieder & Edwards, 2011. Fast identification and removal of sequence contamination from genomic and metagenomic datasets. *PLoS ONE*, 6(3).

Schormann, Ricciardi & Chattopadhyay, 2014. Uracil-DNA glycosylases-Structural and functional perspectives on an essential family of DNA repair enzymes. *Protein science : a publication of the Protein Society*, 23(12), pp.1667–85. Available at: http://www.ncbi.nlm.nih.gov/pubmed/25252105 [Accessed September 10, 2015].

Schweiger et al., 2009. Genome-wide massively parallel sequencing of formaldehyde fixed-paraffin embedded (FFPE) tumor tissues for copy-number-and mutation-analysis. *PLoS ONE*, 4(5).

Segal et al., 2015. A Phase II Efficacy and Safety, Open-Label, Multicenter Study of Imprime PGG Injection in Combination With Cetuximab in Patients With Stage IV KRAS-Mutant Colorectal Cancer. *Clinical Colorectal Cancer*.

Serth et al., 2000. Quantitation of DNA extracted after micropreparation of cells from frozen and formalin-fixed tissue sections. *The American journal of pathology*, 156(4), pp.1189–1196.

Shendure et al., 2005. Accurate multiplex polony sequencing of an evolved bacterial genome. *Science (New York, N.Y.)*, 309(5741), pp.1728–1732.

Shendure et al., 2004. Advanced sequencing technologies: methods and goals. *Nature reviews. Genetics*, 5(5), pp.335–344.

Shendure et al., 2011. Overview of DNA sequencing strategies. *Current Protocols in Molecular Biology*, (SUPPL.96).

Shibata et al., 2000. RIKEN Integrated Sequence Analysis (RISA) system-384-format sequencing pipeline with 384 multicapillary sequencer. *Genome Research*, 10(11), pp.1757–1771.

Shigeta et al., 2013. Expression of Epidermal Growth Factor Receptor Detected by Cetuximab Indicates Its Efficacy to Inhibit In Vitro and In Vivo Proliferation of Colorectal Cancer Cells. *PLoS ONE*, 8(6).

Sikorsky et al., 2007. DNA damage reduces Taq DNA polymerase fidelity and PCR amplification efficiency. *Biochemical and Biophysical Research Communications*, 355(2), pp.431–437.

Slupska et al., 1999. Functional expression of hMYH, a human homolog of the Escherichia coli MutY protein. *Journal of Bacteriology*, 181(19), pp.6210–6213.

Smith, Naven, et al., 2013. Exome resequencing identifies potential tumor-suppressor genes that predispose to colorectal cancer. *Human mutation*, 34(7), pp.1026–34. Available at: http://www.ncbi.nlm.nih.gov/pubmed/23585368 [Accessed May 27, 2015].

Smith et al., 1986. Fluorescence detection in automated DNA sequence analysis. *Nature*, 321(6071), pp.674–679. Available at: http://www.ncbi.nlm.nih.gov/pubmed/3713851.

Smith, Fisher, et al., 2013. Somatic profiling of the epidermal growth factor receptor pathway in tumors from patients with advanced colorectal cancer treated with chemotherapy ± cetuximab. *Clinical cancer research : an official journal of the American Association for Cancer Research*, 19(15), pp.4104–13. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3732482&tool=pmcentrez&rendertype=abstract.

Smith & Waterman, 1981. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1), pp.195–197.

Song, Salmena & Pandolfi, 2012. The functions and regulation of the PTEN tumour suppressor. *Nat Rev Mol Cell Biol*, 13(5), pp.283–296. Available at: http://www.ncbi.nlm.nih.gov/pubmed/22473468.

Spano et al., 2005. Epidermal growth factor receptor signaling in colorectal cancer: preclinical data and therapeutic perspectives. *Annals of oncology : official journal of the European Society for Medical Oncology / ESMO*, 16(2), pp.189–194.

Srinivasan, Sedmak & Jewell, 2002. Effect of fixatives and tissue processing on the content and integrity of nucleic acids. *The American journal of pathology*, 161(6), pp.1961–1971.

Swerdlow & Gesteland, 1990. Capillary gel electrophoresis for rapid, high resolution DNA sequencing. *Nucleic acids research*, 18(6), pp.1415–1419.

Tebbutt, Pedersen & Johns, 2013. Targeting the ERBB family in cancer: couples therapy. *Nature reviews. Cancer*, 13(9), pp.663–73. Available at: http://www.ncbi.nlm.nih.gov/pubmed/23949426.

Thibodeau, Bren & Schaid, 1993. Microsatellite instability in cancer of the proximal colon. *Science (New York, N.Y.)*, 260(5109), pp.816–819.

Thorvaldsdóttir, Robinson & Mesirov, 2013. Integrative Genomics Viewer (IGV): High-performance genomics data visualization and exploration. *Briefings in Bioinformatics*, 14(2), pp.178–192.

Tian et al., 2013. A combined oncogenic pathway signature of BRAF, KRAS and PI3KCA mutation improves colorectal cancer classification and cetuximab treatment prediction. *Gut*, 62(4), pp.540–9. Available at: http://www.ncbi.nlm.nih.gov/pubmed/22798500 [Accessed March 11, 2013].

Timmermann et al., 2010. Somatic mutation profiles of MSI and MSS colorectal cancer identified by whole exome next generation sequencing and bioinformatics analysis. *PLoS ONE*, 5.

Tokuda et al., 1990. Fundamental study on the mechanism of DNA degradation in tissues fixed in formaldehyde. *Journal of clinical pathology*, 43(9), pp.748–751.

Torre et al., 2015. Global Cancer Statistics, 2012. *CA: a cancer journal of clinicians.*, 65(2), pp.87–108. Available at: http://onlinelibrary.wiley.com/doi/10.3322/caac.21262/abstract.

Tosar et al., 2014. Mining of public sequencing databases supports a non-dietary origin for putative foreign miRNAs: underestimated effects of contamination in NGS. *RNA (New York, N.Y.)*, 20(6), pp.754–7. Available at: http://www.ncbi.nlm.nih.gov/pubmed/24729469.

Trapnell et al., 2010. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature biotechnology*, 28(5), pp.511–515.

Trapnell, Pachter & Salzberg, 2009. TopHat: Discovering splice junctions with RNA-Seq. *Bioinformatics*, 25(9), pp.1105–1111.

Turcatti et al., 2008. A new class of cleavable fluorescent nucleotides: Synthesis and optimization as reversible terminators for DNA sequencing by synthesis. *Nucleic Acids Research*, 36(4).

Turnbull et al., 1967. Cancer of the colon: the influence of the no-touch isolation technic on survival rates. *Annals of surgery*, 166(3), pp.420–427.

Ueno & Yeung, 1994. Simultaneous monitoring of DNA fragments separated by electrophoresis in a multiplexed array of 100 capillaries. *Analytical Chemistry*, 66(9), pp.1424–1431.

Uribe-Lewis et al., 2015. 5-hydroxymethylcytosine marks promoters in colon that resist DNA hypermethylation in cancer. *Genome biology*, 16(1), p.69. Available at: http://genomebiology.com/2015/16/1/69.

Venter et al., 2001. The sequence of the human genome. *Science (New York, N.Y.)*, 291(5507), pp.1304–51. Available at: http://www.ncbi.nlm.nih.gov/pubmed/11181995 [Accessed February 28, 2013].

Virella & Lopes-Virella, 2012. The pathogenic role of the adaptive immune response to modified LDL in diabetes. *Frontiers in Endocrinology*, 3(JUN).

Voelkerding, Dames & Durtschi, 2009. Next-generation sequencing: from basic research to diagnostics. *Clinical chemistry*, 55, pp.641–658.

Vogelstein et al., 1988. Genetic alterations during colorectal-tumor development. *The New England journal of medicine*, 319(9), pp.525–532.

Waring et al., 2015. RAS Mutations as Predictive Biomarkers in Clinical

Management of Metastatic Colorectal Cancer. *Clinical Colorectal Cancer*.

Warren et al., 2007. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, 23(4), pp.500–501.

Warusavitarne et al., 2006. 5-Fluorouracil (5FU) treatment does not influence invasion and metastasis in microsatellite unstable (MSI-H) colorectal cancer. *International Journal of Colorectal Disease*, 21(7), pp.625–631.

Webber et al., 2015. Systematic review of the predictive effect of MSI status in colorectal cancer patients undergoing 5FU-based chemotherapy. *BMC Cancer*, 15(1), p.156. Available at: http://www.biomedcentral.com/1471-2407/15/156.

Wicker et al., 2006. 454 sequencing put to the test using the complex genome of barley. *BMC genomics*, 7, p.275.

Wicki, Herrmann & Christofori, 2010. Kras in metastatic colorectal cancer. *Swiss Medical Weekly*, 140(DECEMBER), pp.14–19.

Williams et al., 1999. A high frequency of sequence alterations is due to formalin fixation of archival specimens. *The American journal of pathology*, 155(5), pp.1467–1471.

Wong et al., 2013. Targeted-capture massively-parallel sequencing enables robust detection of clinically informative mutations from formalin-fixed tumours. *Scientific reports*, 3(3), p.3494. Available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3861801&tool=pmcentrez&rendertype=abstract.

Yagi et al., 1996. The role of DNase and EDTA on DNA degradation in formaldehyde fixed tissues. *Biotechnic & histochemistry : official publication of the Biological Stain Commission*, 71(3), pp.123–129.

Yarden & Sliwkowski, 2001. Untangling the ErbB signalling network. *Nature reviews. Molecular cell biology*, 2(2), pp.127–37. Available at: http://www.ncbi.nlm.nih.gov/pubmed/11252954.

Yun et al., 2008. The T790M mutation in EGFR kinase causes drug resistance by increasing the affinity for ATP. *Proceedings of the National Academy of Sciences of the United States of America*, 105(6), pp.2070–2075.

Zerbino et al., 2009. Pebble and rock band: Heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. *PLoS ONE*, 4(12).

Zerbino & Birney, 2008. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5), pp.821–829.

Zhang et al., 2011. A practical comparison of De Novo genome assembly software tools for next-generation sequencing technologies. *PLoS ONE*, 6(3).

Zhang et al., 2015. Comparison and evaluation of two exome capture kits and sequencing platforms for variant calling. *BMC Genomics*, 16(1), p.581. Available at: http://www.biomedcentral.com/1471-2164/16/581.

Zhu & Waggoner, 1997. Molecular mechanism controlling the incorporation of fluorescent nucleotides into DNA by PCR. *Cytometry*, 28(3), pp.206–211.

Appendix

Custom Script 1

```
#VarScape: Calling variants from the NGS data landscape
#Modified version of var_gate.pl, which was originally modified from exomeExpress.pl.
#Importantly this pipeline can handle exome and targeted resequencing data - with a few minor
tweaks. To switch to targeted reseq analysis:
#	- obviously change the input/output folders for the dataset.
#	- remove the samtools rmdup step, and use the fixmate_sorted output as input for the
realignment steps (will have to create an index for fixmate sorted output).
#	- for VQSR, disable VariantRecalibrator and ApplyRecalibration steps and instead
activate the VariantFiltration tool.
#Reverse the changes when handling exome data.

#Calls variants with either UnifiedGenotyper or HaplotypeCaller. With HaplotypeCaller, use
VariantFiltration instead of VariantRecalibrator (and therefore don't use ApplyRecalibration). For
a list of best practice filters, see the GATK website.

#Although the best practices for VQSR may specify to filter for mapping quality < 40 and
haplotype score > 13, with the pilot project using Illumina's TruSeq Custom Amplicon kit, we find
that known variants are labelled as filtered for the majority of samples - with the main culprit
being haplotype score.
#Known variants that passed the HS filter failed mainly on mapping quality. To increase
specifity after VQSR, I therefore suggest removing the HaplotypeScore filter altogether and
setting the MQ filter = 35.

#Updated steps from exomeExpress.pl (applied after the analysis of the first 50 exomes):
#The pipeline now uses a variant validation step, GATK's ValidateVariants tool.
#The CountCovariates walker has been replaced by BaseRecalibrator and output is applied via
PrintReads instead of TableRecalibration.
#VQSR is now also applied to indels.

#!/usr/bin/perl

#This module allows you to create directories, useful when creating folders for the pipeline
output.
use File::Path;

use Term::ANSIColor;

#sub user definitions - Create output directories. This array must be in the main program as it is
used by several subroutines.
my @output_dirs = (
	"02_bwa_aln", "03_bwa_sam", "04_SAMtools_view", "05_SAMtools_sort",
"06_SAMtools_fixmate",
	"07_SAMtools_rmdup", "08_GATK_realignertargetcreator", "09_GATK_indelrealigner",
"10_SAMtools_pileup",
	"11_GATK_countcovariates", "12_GATK_tablerecalibration",
"13_GATK_unifiedgenotyper", "14_GATK_selectvariants",
	"15_GATK_variantrecalibrator", "16_GATK_applyrecalibration",
"17_GATK_combinevariants", "18_snpEff", "19_GATK_variantannotator"
);

#create a reference of the output_dirs array for passing to subroutines.
my $output_dirs = \@output_dirs;
```

```
#Call the subroutine that takes STDIN from the command line. User answers about sequencing
data (e.g. location of reads, platform used, paired/single ends, whether to add annotation, etc).
my ($system_calls_input, $exomes, $system_calls) = user_definitions($output_dirs);
my @system_calls_input = @$system_calls_input;
my %exomes = %{$exomes};

#Displays the user defined input to the user, so that he/she can check the chosen parameters
before running the pipeline. The user enters 'y' to proceed, 'q' to quit or 'n' to re-enter the
required input.
while ($proceed ne "y"){
	print "Would you like to execute the pipeline using these parameters? Enter 'y' or 'n'.
Enter 'q' to quit.\n";

	chomp($proceed = <STDIN>);

	#convert $proceed to lower case.
	$proceed =~ s/[A-Z]/[a-z]/;

	#stop the script if the user enters "q"
	exit if $proceed eq "q";

	#if the user is happy with their input, then go ahead and run the pipeline.
	if ($proceed eq "y"){system_calls($output_dirs, $system_calls_input, $exomes,
$system_calls);}

	#if the user enters something other than y or n (or q), tell them to rechoose,
	if ($proceed ne "y" && $proceed ne "n"){
		$proceed = "";
		print "Please enter 'y' or 'n'.\n";
	}

	#If the user is not happy with their input, get them to re-enter it,
	if ($proceed eq "n"){
		#call &user definitions again and capture the return values.
		my ($system_calls_input, $exomes, $system_calls) =
user_definitions($output_dirs);

		#Take the returned references and turn them into proper arrays/hashes.
		@system_calls_input = @$system_calls_input;
		%exomes = %{$exomes};
		%system_calls = %{$system_calls};
	}
}

##########################################################################
#########################
#subroutine user_defitions
#User enters the variables needed for the pipeline including:
#	number of processing cores available
#	location of the fastq input files
#	location to create output directories (and a check to confirm)/store output.
#	the reference genome file
#	the GATK.jar file
#	a file of known polymorphic sites (eg. from dbsnp hapmap and Mills & Devine) in '.vcf'
format
#	the platform used for sequencing
#	the algorithm for alignment
#	paired ends or single ends or bwasw
#	the types of variants to call
#	...
```

```
#The subroutine also gathers input fastq file names, names of output files and the read group
header information.
#
##########################################################################
#########################
sub user_definitions{
	my $output_dirs = shift @_;
	my @output_dirs = @$output_dirs;

	#create local variables for each of the user inputs.
	my ($bed_file, $call_regions, $trim_reads, $trimmomatic_jar, $seq_adapters,
$create_index, $add_ann, $fastq_reads_dir, $output_dir, $create_dir, $reference_genome,
$gatk_jar, $snpEff_jar, $dbsnp, $hapmap_vcf, $omni_vcf, $mills_Devine_vcf, $platform,
$index_algorithm, $bwa_ended, $bigS, $who_ya_gonna_call, $cores,);

	print "Please enter the choices asked for below. Files or directories should be entered
as: \"/path/to/file/or/directory\" without the quotation marks.\nEntering 'q' or 's' will allow you to
quit this program or skip a question, respectively.\nIf you do skip the question then one or more
downstream processes may not work as expected!\n\n";

	#sub user definitions - fastq input location
	#fastq dir is entered via command line (or user enters 'Q' to exit) and the existence of
the dir is checked.

	while ($fastq_reads_dir eq "" && $fastq_reads_dir ne "s"){
		print "\nPlease enter filepath to the folder containing your sequencing reads in
FASTQ format.\n";
	$fastq_reads_dir = &ask;

	      unless (-d $fastq_reads_dir){
			if ($fastq_reads_dir ne "s"){
				$fastq_reads_dir = "";
				print "Directory $fastq_reads_dir not found. Please try
again.\n";
			}
		}
      }

      while ($trim_reads ne "y" && $trim_reads ne "n" && $trim_reads ne "s"){
	print"Would you like to trim adapter sequences from the reads? Please enter \"y\" or
\"n\".\n";

	$trim_reads = &ask;

	$trim_reads =~ tr/A-Z/a-z/;

	if ($trim_reads ne "y" && $trim_reads ne "n" && $trim_reads ne "s"){
		$trim_reads = "";
		print "Please enter 'y' or 'n'.\n";
	}
      }

      #sub user definitions - GenomeAnalysisTK.jar location
	print "\nPlease type the filepath to the GenomeAnalysisToolKit.jar file.\n";
	while ($trimmomatic_jar eq "" && $trimmomatic_jar ne "s"){
	$trimmomatic_jar = &ask;

	      unless (-f $trimmomatic_jar ){
		$trimmomatic_jar =~ tr/A-Z/a-z/;
```

```
                    if ($trimmomatic_jar ne "s"){
                        $trimmomatic_jar = "";

                        print "Trimmomatic.jar not found! Try again or enter 'q' to
quit\n";
                    }
                }
        }

        while ($seq_adapters eq "" && $seq_adapters ne "s"){
            print "\nPlease enter filepath to the sequencing adapter file.\n";
            $seq_adapters = &ask;

            unless (-f $seq_adapters){
                    unless ($seq_adapters eq "s"){
                        $seq_adapters = "";
                        print "Sequencing adapter file not found! Try again or enter 'q'
to quit\n";
                    }
                }
            }

        while ($output_dir eq "" && $output_dir ne "s"){
            print "\nPlease enter filepath to the folder you would like your output storing. You may
type the name of a new folder and it will be made.\n";
            $output_dir = &ask;

            if ($output_dir eq "s"){$output_dir = "contamination";}
            unless ($output_dir eq "s"){
                    #cheat - enter bgix\d\d or unc to create output directories for BGI or
North Carolina respectively
                    if ($output_dir eq "bgix10"){$output_dir =
"/media/marc/Triceratops/Exomes/BGI output"};
                    if ($output_dir eq "bgix13"){$output_dir =
"/media/marc/Triceratops/Exomes/BGIx13 output"};
                    if ($output_dir eq "bgix16"){$output_dir =
"/media/marc/Triceratops/Exomes/BGIx16 output"};
                    if ($output_dir eq "bgix18"){$output_dir =
"/media/marc/Triceratops/Exomes/BGIx18 output"};
                    if ($output_dir eq "unc"){$output_dir =
"/media/marc/Triceratops/Exomes/UNC output"};
                    if ($output_dir eq "nhs"){$output_dir =
"/media/marc/Triceratops/Exomes/Pilot project output/NHS output"};
                    if ($output_dir eq "nhs_trim"){$output_dir =
"/media/marc/Triceratops/Exomes/NHS trimmed output"};
                    if ($output_dir eq "nhs_repeat"){$output_dir =
"/media/marc/Triceratops/Exomes/NHS repeat output"};
                    if ($output_dir eq "wthd"){$output_dir =
"/media/marc/Triceratops/Exomes/Pilot project output/WT (HD) output"};
                    if ($output_dir eq "wtld"){$output_dir =
"/media/marc/Triceratops/Exomes/WT (LD) output"};
                    if ($output_dir eq "eden"){$output_dir =
"/media/marc/Triceratops/Exomes/Eden output"};
                    if ($output_dir eq "ingram"){$output_dir =
"/media/marc/Triceratops/Exomes/Ingram 14-6697 output"};
                    if ($output_dir eq "miseq"){$output_dir =
"/media/marc/Triceratops/Exomes/Miseq output"};
                    if ($output_dir eq "nr"){$output_dir =
"/media/marc/Triceratops/Exomes/Non responder output"};
```

```
                                if ($output_dir eq "contamination"){$output_dir =
"/media/marc/Triceratops/Exomes/Contamination output"};
                        }

                        my ($output_dir, $create_dir) = let_there_be_output($output_dir, $output_dirs);
        }

        #sub user definitions - reference genome location
            print "\nPlease type the filepath to the reference genome.\n";

            while ($reference_genome eq "" && $reference_genome ne "s"){
            $reference_genome = &ask;

                    unless (-f $reference_genome ){
                        $reference_genome =~ tr/A-Z/a-z/;
                                unless ($reference_genome eq "s"){
                                        $reference_genome = "";
                                        print "Reference genome not found! Try again or enter 'q' to
quit\n";
                                }
                        }
        }

        #sub user definitions - GenomeAnalysisTK.jar location
            print "\nPlease type the filepath to the GenomeAnalysisToolKit.jar file.\n";
            while ($gatk_jar eq "" && $gatk_jar ne "s"){
            $gatk_jar = &ask;

                    unless (-e $gatk_jar ){
                        $gatk_jar =~ tr/A-Z/a-z/;

                                if ($gatk_jar ne "s"){
                                        $gatk_jar = "";

                                        print "GenomeAnalysisTK.jar not found! Try again or enter 'q' to
quit\n";
                                }
                        }
        }

        #sub user definitions - snpEff.jar and snpEff.config locations
            print "\nPlease type the filepath to the snpEff.jar file.\n";
            while ($snpEff_jar eq "" && $snpEff_jar ne "s"){
                        $snpEff_jar = &ask;

                        unless (-e $snpEff_jar ){
                                $snpEff_jar =~ tr/A-Z/a-z/;

                                if ($snpEff_jar ne "s"){
                                        $snpEff_jar = "";
                                        print "snpEff.jar not found! Try again or enter 'q' to quit\n";
                                }
                        }

            }

             ##sub user definitions - Enter the regions for variant calling.
             while ($call_regions !~ /^[yns]$/i){
            print "Would you like to call variants at specific regions? Please enter 'y' or 'n'.\n";
            $call_regions = &ask;
            $call_regions =~ tr/A-Z/a-z/;

            unless ($call_regions =~ /^[yns]$/){
                    $call_regions = "";
                    print "Please enter 'y' or 'n'.\n";
            }

            if ($call_regions eq "y" || $call_regions eq "s"){
                            print "\nEnter BED file containing regions for variant calling.\n";

                            while ($bed_file eq "" && $bed_file ne "s"){
                                    $bed_file = &ask;

                                    unless (-f $bed_file){
                                            unless ($bed_file eq "s"){
                                                    $bed_file = "";

                                                    print "Please enter the filepath to regions for
variant calling.\n";
                                            }
                                    }
                            }
                    }
            }

        #option to index reference genome
        print "\nWould you like to create an index for the reference genome?\n If you have not
done this before you must do it before the pipeline can proceed.\n";
        while ($create_index eq "" && $create_index ne "y" && $create_index ne "n" &&
$create_index ne "s"){
            $create_index = &ask;
            $create_index =~ tr/A-Z/a-z/;

            if ($create_index ne "y" && $create_index ne "n" && $create_index ne "s"){
                    $create_index = "";
                    print "Please enter 'y' or 'n'.\n";
            }
        }

        #sub user definitions - annotate variants?
            print "\nWould you like to add annotations to your variant calls? This step is advised.
Enter 'y' or 'n'\n";
            while ($add_ann eq "" && $add_ann ne "y" && $add_ann ne "n" && $add_ann ne "s"){
            $add_ann = &ask;
            $add_ann =~ tr/A-Z/a-z/;

            if ($add_ann ne "y" && $add_ann ne "n" && $add_ann ne "s"){
                    $add_ann = "";
                    print "Please enter 'y' or 'n'.\n";
            }
        }

        #sub user definitions - training dbSNP_Vxxx.vcf location
            print "\nPlease type the filepath to the known polymorphic sites dbSNP '.vcf' file.\n";
            while ($dbsnp eq "" && $dbsnp ne "s"){
            $dbsnp = &ask;

                unless (-e $dbsnp ){
                    $dbsnp =~ tr/A-Z/a-z/;
                if ($dbsnp ne "s"){
                        $dbsnp = "";

                        print "dbSNP '.vcf' file not found! Try again or enter 'q' to quit\n";
                }
        }
}

#sub user definitions - training hapmap vcf location
print "\nPlease type the filepath to the known polymorphic sites hapmap '.vcf' file.\n";

   while ($hapmap_vcf eq "" && $hapmap_vcf ne "s"){
   $hapmap_vcf = &ask;

        unless (-e $hapmap_vcf ){
           $hapmap_vcf =~ tr/A-Z/a-z/;

                if ($hapmap_vcf ne "s"){
                        $hapmap_vcf = "";

                        print "hapmap '.vcf' file not found! Try again or enter 'q' to
quit\n";
                }
        }
}

#sub user definitions - training 1000 genomes omni chip 2.5M vcf location
print "\nPlease type the filepath to the known polymorphic sites 1000genomes omni chip
'.vcf' file.\n";
   while ($omni_vcf eq "" && $omni_vcf ne "s"){
   $omni_vcf = &ask;

        unless (-e $omni_vcf ){
           $omni_vcf =~ tr/A-Z/a-z/;

                if ($omni_vcf ne "s"){
                        $omni_vcf = "";

                        print "omni '.vcf' file not found! Try again or enter 'q' to quit\n";
                }
        }
}

#sub user definitions - training 1000 genomes omni chip 2.5M vcf location
print "\nPlease type the filepath to the 1000g Phase 1 high confidence SNPs '.vcf' file.\n";

   while ($phase1 eq "" && $phase1 ne "s"){
   $phase1 = &ask;

        unless (-e $phase1 ){
           $phase1 =~ tr/A-Z/a-z/;

                if ($phase1 ne "s"){
                        $phase1 = "";

                        print "1000G Phase 1 SNPs file not found! Try again or enter 'q'
to quit\n";
                }
        }
```

```
	}

	#sub user definitions - training Mills & Devine vcf location
	print "\nPlease type the filepath to the known polymorphic sites Mills & Devine '.vcf' file.\n";

	    while ($mills_Devine_vcf eq "" && $mills_Devine_vcf ne "s"){
	    $mills_Devine_vcf = &ask;

		    unless (-e $mills_Devine_vcf ){
		        $mille_Devine_vcf =~ tr/A-Z/a-z/;

				if ($mills_Devine_vcf ne "s"){
					$mills_Devine_vcf = "";

					print "Mills & Devine '.vcf' file not found! Try again or enter 'q' to
quit\n";
				}
			}
	}

	#Sub user definitions - Choose the sequencing platform
	    print "\nPlease enter the sequencing platform; either 'CAPILLARY' 'LS454' 'ILLUMINA'
'SOLID' 'HELICOS' IOTORRENT' or 'PACBIO'.\n";

	    while ($platform eq "" && $platform ne "S"){
	    $platform = &ask;
	    $platform =~ tr/[a-z]/[A-Z]/;

		    if (
				$platform ne "CAPILLARY" && $platform ne "LS454" && $platform ne
"ILLUMINA" &&
				$platform ne "SOLID" && $platform ne "HELICOS" && $platform ne
"IOTORRENT" &&
				$platform ne "PACBIO" && $platform ne "s" && $platform ne "Q" &&
$platform ne "S"
			){
				$platform = "";

				print "\nError: please select from 'CAPILLARY', 'LS454', 'ILLUMINA',
'SOLID', 'HELICOS', 'IOTORRENT' or 'PACBIO'.\n";
			}
	}

	##sub user definitions - Enter the bwa index algorithm
	print "\nEnter algorithm to use for bwa index either 'is' or 'bwtsw' (recommended).\n";

	    while ($index_algorithm eq "" && $index_algorithm ne "s" && $platform ne "LS454"){
	    $index_algorithm = &ask;

	    $index_algorithm =~ tr/[A-Z]/[a-z]/;

		    if ($index_algorithm ne "is" && $index_algorithm ne "bwtsw" && $index_algorithm
ne "s"){
			$index_algorithm = "";
			print "Please select either 'is' or 'bwtsw'.\n";
		    }
	     }

	    #sub user definitions - choose samse sampe or bwasw based on platform choice
	unless ($platform eq "LS454"){ #LS454 uses bwasw not bwa samse/sampe - colour
reads not space reads!
		print "\nAre your reads single- or paired-ended? Enter either 'samse' (single
ends) or 'sampe' (paired ends).\n";

		while ($bwa_ended eq "" && $bwa_ended ne "s"){
			$bwa_ended = &ask;
			$bwa_ended =~ tr/[A-Z]/[a-z]/;

			if ($bwa_ended ne "sampe" && $bwa_ended ne "samse" &&
$bwa_ended ne "s"){
				$bwa_ended = "";
				print "Please select either 'samse' or 'sampe'. Enter 'q' to quit or
's' to skip.\n";
			}
		}

		#cheat - default setting on panther
		if ($bwa_ended eq "s"){$bwa_ended = "sampe";}
    }

    #use bwasw instead of bwa sampe/samse.
    else{
		$bwa_ended = "bwasw";
	}

	if ($bwa_ended eq "sampe"){$bigS = "S";}
    else {$bigS = "s";}

	#Sub user definitions - Choose the variants to call
	print "\nWould you like to call SNPs indels or both?\n";
	while ($who_ya_gonna_call eq "" && $who_ya_gonna_call ne "S"){
	$who_ya_gonna_call = &ask;
	$who_ya_gonna_call =~ tr/[a-z]/[A-Z]/;

		if ($who_ya_gonna_call ne "SNP" && $who_ya_gonna_call ne "INDEL" &&
$who_ya_gonna_call ne "BOTH" && $who_ya_gonna_call ne "S"){
			$who_ya_gonna_call = "";
			print "Error please select from 'SNP' 'INDEL' or 'BOTH'.\n";
		}
    }

	#Sub user definitions - Choose the variants to call
	print "\nWould you like to combine samples to call variants? .\n";
	while ($pro_multi eq "" && $pro_multi ne "s"){
	$pro_multi = &ask;
	$pro_multi =~ tr/[A-Z]/[a-z]/;

		if ($pro_multi ne "y" && $pro_multi ne "n" && $pro_multi ne "s"){
			$pro_multi = "";
			print "Error: please select from 'y' or 'n".\n";
		}
    }

    ##sub user definitions - Enter the number of cores available.
    print "\nEnter number of processing cores you have available for use.\n";
	while ($cores eq "" && $cores ne "s"){
	$cores = &ask;
	$cores =~ tr/A-Z/a-z/;
```

```
		if ($cores =~ /\D/ && $cores >= 10 && $cores ne "s"){
			$cores = "";

			print "Please select a realistic (<10) numerical value. Note that you should not
enter more processors than you have!\n";
		}
	 }

	#Default settings on Panther.
	if ($fastq_reads_dir eq "s"){$fastq_reads_dir = "/media/marc/Brachiosaur/NGS
data/Contamination profile/Fastq files";}
	if ($reference_genome eq "s"){$reference_genome =
"/media/marc/Brachiosaur/Exomes/reference genome/human_g1k_v37.fasta";}#NGS data/Pilot
project (KRAS, NRAS, BRAF & PIK3CA)/NHS Miseq/Misc Miseq generated
files/Reference/Chris-Pilot.txt.fa";}
	if ($trimmomatic_jar eq "s"){$trimmomatic_jar = "/home/marc/Documents/bioinformatics
software/Trimmomatic-0.32/trimmomatic-0.32.jar";}
	if ($seq_adapters eq "s"){$seq_adapters = "/home/marc/Documents/bioinformatics
software/Trimmomatic-0.32/adapters/TruSeq3-PE-2.fa";}
	if ($gatk_jar eq "s"){$gatk_jar = "/home/marc/Documents/bioinformatics
software/GenomeAnalysisTK-3.3-0/GenomeAnalysisTK.jar";}
	if ($dbsnp eq "s"){$dbsnp = "/media/marc/Brachiosaur/Exomes/database
resources/dbSNP_141_b37.vcf";}
	if ($snpEff_jar eq "s"){$snpEff_jar = "/home/marc/Documents/bioinformatics
software/snpEff_3_6/snpEff.jar";}
	if ($hapmap_vcf eq "s"){$hapmap_vcf = "/media/marc/Brachiosaur/Exomes/database
resources/hapmap_3.3.b37.sites.vcf";}
	if ($omni_vcf eq "s"){$omni_vcf = "/media/marc/Brachiosaur/Exomes/database
resources/1000G_omni2.5.b37.sites.vcf";}
	if ($phase1 eq "s"){$phase1 = "/media/marc/Brachiosaur/Exomes/database
resources/1000G_phase1_snps_high_confidence_b37.vcf";}
	if ($mills_Devine_vcf eq "s"){$mills_Devine_vcf =
"/media/marc/Brachiosaur/Exomes/database resources/Mills_Devine_2hit.indels.b37.sites.vcf";}
	if ($bed_file eq "s"){$bed_file = "/home/marc/Documents/work/reseq analysis/pilot
project/Miscellaneous/hotspot codon regions.BED";}
	if ($call_regions eq "s"){$call_regions = "n";}
	if ($create_index eq "s"){$create_index = "y";}
    if ($add_ann eq "s"){$add_ann = "y";}
    if ($platform eq "S"){$platform = "ILLUMINA";}
    if ($index_algorithm eq "s"){$index_algorithm = "bwtsw";}
    if ($who_ya_gonna_call eq "S"){$who_ya_gonna_call = "BOTH";}
    if ($pro_multi eq "s"){$pro_multi = "n";}
    if ($cores eq "s"){$cores = 13;}

    #The SNPEff config file is also required. Assuming a standard install of SNPEff, this config
file should be in the same folder as the snpEff.jar file.
    $snpEff_config = $snpEff_jar;
	$snpEff_config =~ s/jar/config/;

	#The neccessary input to read_exomes sub. Although the @output_dirs here is stored
in @inputs as simple variables, they can be collected back into an array in read_exomes.
	my @read_exome_inputs = ($fastq_reads_dir, $output_dir, $bwa_ended, $platform,
$who_ya_gonna_call);
	my $read_exome_inputs = \@read_exome_inputs;

	#calls &read_exomes sub using @read_exome_inputs as input and traps the returned
references as scalars, which can then be turned back into arrays or hashes.
	my $exomes = read_fq($read_exome_inputs, $output_dirs);

	my $system_calls;
	# ($exomes, $system_calls) = trim($exomes, $trimmomatic_jar, $seq_adapters,
$cores);

	#Turn the returned hash references into a proper hash.
	my %exomes = %{$exomes};

	#visible checks for varibles which have just been input.
	print  "\nInput directory:\n$fastq_reads_dir\n\n";
	print  "Output directory:\n$output_dir\n\n";
	print  "Create dir check:\n$create_dir\n\n";
	print "Trimmomatic.jar:\n$trimmomatic_jar\n\n";
	print "Sequencing adapter file:\n$seq_adapters\n\n";
	print  "Reference genome:\n$reference_genome\n\n";
	print  "Index reference?:\n$create_index\n\n";
	print  "GATK.jar:\n$gatk_jar\n\n";
	print  "dbSNP.vcf:\n$dbsnp\n\n";
	print  "Hapmap.vcf:\n$hapmap_vcf\n\n";
	print  "Mills & Devine.vcf:\n$mills_Devine_vcf\n\n";
	print  "Sequencing platform:\n$platform\n\n";
	print  "Index algorithm:\n$index_algorithm\n\n";
	print  "Endings:\n$bwa_ended\n\n";
	print  "Variants to call:\n$who_ya_gonna_call\n\n";
	print "Regions to call (y/n: $call_regions): $bed_file\n\n";
	print  "Cores:\n$cores\n\n";

	my @system_calls_input = ($call_regions, $bed_file, $trimmomatic_jar, $seq_adapters,
$create_index, $add_ann, $fastq_reads_dir, $output_dir, $reference_genome, $gatk_jar,
$snpEff_jar, $dbsnp, $hapmap_vcf, $omni_vcf, $phase1, $mills_Devine_vcf, $platform,
$index_algorithm, $bwa_ended, $bigS, $who_ya_gonna_call, $pro_multi, $cores, $output_dirs);

	return (\@system_calls_input, \%exomes, $system_calls);
}
################################################################################
##########################
#end of user_definitions subroutine.
################################################################################
##########################

################################################################################
##########################
#subroutine ask
#Gets STDIN from the user, and exits the script if the input is 'q'.
################################################################################
##########################
sub ask{
		my $input;

	# $input = "s";

	chomp ($input = <STDIN>);

	exit if ($input eq "q" || $input eq "Q");
	return $input;
}
################################################################################
##########################
#end of ask subroutine.
################################################################################
##########################
```

```
##########################################################################
#########################
#subroutine read_exomes
#reads in files from the specified exome directory.
#Returns the name of the output files in a hash:
#%exomes = (
#	$sample => {
#		output_step_1
#		output_step_2 ..
#		output_step_n
#	}
#)
##########################################################################
#########################
sub read_fq{
	#name the local variables passed on from &user_definitions for each of the user inputs.
	#@output_dirs captures the variables from the previous @output_dirs array, so the
parameters passed on are essentially identical.
	my ($read_exome_inputs, $output_dirs) = @_;
	my @read_exome_inputs = @$read_exome_inputs;
	my @output_dirs = @$output_dirs;

	my ($fastq_reads_dir, $output_dir, $bwa_ended, $platform, $who_ya_gonna_call) =
@read_exome_inputs;

	#initialise the %exomes hash but don't populate it yet. Gives lexical scope.
	#This hash will contain names of input/output files for each sample.
	my (%exomes, $sample_count);

	#Search the $fastq_reads_dir and its subdirectories for any FASTQ or FQ files.
	#The sub_dirs scalar returned here is of no further use.
    my ($fq_reads, $sub_dirs) = subway($fastq_reads_dir);
    my %fq_reads = %{$fq_reads};

    ##sub user definitions - loop over the hash of input fastq files for various purposes.
    for my $coinID (sort {"\L$a" cmp "\L$b"} keys %fq_reads){
		foreach my $fq_file (sort {"\L$a" cmp "\L$b"} @{$fq_reads{$coinID}}){
			my (%RG_header, $sampleID, $lane, $mate, $dna_type);

			#count how many sample files have been analysed so far
			$sample_count++;

			if ($coinID =~ /\-[a-z]+/){$dna_type = $&;}

			#set the variables for this sample.
			$sampleID = $fq_file;

			$sampleID =~ s/\_(trimmed|repeat)//i;

			#remove the file path and file extension, along with "_pe" that some file
names may have. Also remove the run, COIN ID and tissue type.
			$sampleID =~ s/^.+\///g;
			$sampleID =~ s/\..+$//;
			$sampleID =~ s/^.+\-.+\-[a-z]+\_//i;
			if ($sampleID =~ /\_pe/i){$sampleID =~ s/$&//;}

			my @sampleID = split (/\_/, $sampleID);
```

```
            #remove the mate info for 'multi_sample'.
            if ($sampleID =~ /multiSample/i){$sampleID =~ s/\_\d$//;}

            if ($fq_file =~ /Chapel Hill Data/){
                $lane = $sampleID[3];
                $mate = $sampleID[4];
            }

            elsif ($fq_file =~ /(Non
responder)|(Contamination)|(PIK3CA\)\/NHS)|(NHS Miseq)|(NHS repeat)|(NHS
trimmed)|(Eden)|(Ingram)/){
                $lane = $sampleID[1];
                $lane =~ s/L00//i;

                $mate = $sampleID[2];
                $mate =~ s/R//i;

            }

            else{
                #set a generic lane for now. You should be able to determine
lane from fastq files.
                $lane = 1;
                $mate = $sampleID[1];
            }

            $sampleID = $coinID."\_".$mate;

            %RG_header = (
                $sampleID => {
                    ID => $sampleID,
                    SM => $coinID,
                    Mate => $mate,
                    Lane => $lane,
                    PL => $platform
                }
            );

            print "sampleid: $sampleID\n";

            #Populate the %exomes hash with RG info for this sample.
            ${$exomes{$sampleID}{RG_headline}} =
"\@RG\tID:$RG_header{$sampleID}{ID}\tSM:$RG_header{$sampleID}{SM}\tPL:$RG_header{$
sampleID}{PL}";

            #sub user definitions - make names of input/output files ('.sai' '.sam'
'.bam'  '.intervals' '.tranches' '.csv' '.vcf', etc) and store in an exome unique hash.
            #the hash is built so that each element has the required information for
calling systems later. i.e. bwa aln has info for both mate pairs whereas samtools sort has info for
just the exome name/ID
            #this is because bwa sampe will merge mate pairs into a single file for
the exome.
            ${$exomes{$sampleID}{aln_sai}} =
"\"$output_dir/$output_dirs[0]/$sampleID\_aln.sai\"";
        #step 2 output (2nd mate) bwa aln
            ${$exomes{$sampleID}{mate_1_file}} = "\"${$fq_reads{$coinID}}[0]\"";
                                                            #original 1st
mate query filename
```

```
                              ${$exomes{$sampleID}{mate_2_file}} = "\"${$fq_reads{$coinID}}[1]\"";
                                                                                      #original 2nd
mate query filename

                              ${$exomes{$sampleID}{mate_1_paired}} =
"\"$fastq_reads_dir\/$coinID\_1\_paired.fq\"";
                              ${$exomes{$sampleID}{mate_1_unpaired}} =
"\"$fastq_reads_dir\/$coinID\_1\_unpaired.fq\"";
                              ${$exomes{$sampleID}{mate_2_paired}} =
"\"$fastq_reads_dir\/$coinID\_2\_paired.fq\"";
                              ${$exomes{$sampleID}{mate_2_unpaired}} =
"\"$fastq_reads_dir\/$coinID\_2\_unpaired.fq\"";

                              #list continued under separate array loop so that exome mate pairs are
not duplicated (step 3 merges mates 1 & 2 into a single file).
                              #finish making names of output files for step 3 (bwa
sampe/samse/bwasw) onwards.
                              #Will consider the 2nd mate pair only (1st mate pair not important at
this step).
                              if ($mate == 2){
                                        ${$exomes{$sampleID}{aln_sai_prev}} =
"${$exomes{\"$coinID\_1\"}{aln_sai}}";
                                                  #The aln.sai file for mate 1

                                        ${$exomes{$sampleID}{sam}} =
"\"$output_dir/$output_dirs[1]/$coinID\_$bwa_ended\.sam\"";
                                                            #step 3 output bwa sam
                                        ${$exomes{$sampleID}{view_bam}} =
"\"$output_dir/$output_dirs[2]/$coinID\_view.bam\"";
                                                  #step 4 output samtools view
                                        ${$exomes{$sampleID}{sorted_bam}} =
"\"$output_dir/$output_dirs[3]/$coinID\_sorted";
                                                  #step 5 output samtools sort. Notice the lack of
quotation mark at the end. This is added in system_calls.
                                        ${$exomes{$sampleID}{fixmate_bam}} =
"\"$output_dir/$output_dirs[4]/$coinID\_fixmate.bam\"";
                                        #step 6 output samtools fixmate
                                        ${$exomes{$sampleID}{postfixmate_sorted_bam}} =
"\"$output_dir/$output_dirs[4]/$coinID\_post_fixmate_sorted";
                    #step 6 output samtools sort (2). Notice the lack of quotation mark at the end.
This is added in system_calls.
                                        ${$exomes{$sampleID}{nodups_bam}} =
"\"$output_dir/$output_dirs[5]/$coinID\_nodups.bam\"";
                                        #step 7 output samtools rmdup
                                        ${$exomes{$sampleID}{indels_intervals}} =
"\"$output_dir/$output_dirs[6]/$coinID\_indels.intervals\"";
                                        #step 8 output gatk realigner target creator
                                        ${$exomes{$sampleID}{realigned_bam}} =
"\"$output_dir/$output_dirs[7]/$coinID\_realigned.bam\"";
                              #step 9 output gatk indel realigner
                                        ${$exomes{$sampleID}{mpileup_bam}} =
"\"$output_dir/$output_dirs[8]/$coinID\_mpileup_bam\"";
                                        #step 10 output samtools mpileup
                                        ${$exomes{$sampleID}{recalibrated_grp}} =
"\"$output_dir/$output_dirs[9]/$coinID\_recalibrated.grp\"";
                                        #step 11 output gatk base recalibrator
                                        ${$exomes{$sampleID}{recalibrated_bam}} =
"\"$output_dir/$output_dirs[10]/$coinID\_recalibrated.bam\"";
                              #step 12 output gatk table recalibrator
```

```
					${$exomes{$sampleID}{metrics}} =
"\"$output_dir/$output_dirs[11]/$coinID\_metrics.txt\"";
						#step 14 output gatk unified genotyper
					${$exomes{$sampleID}{variant_calls}} =
"\"$output_dir/$output_dirs[11]/$coinID\_$who_ya_gonna_call.vcf\"";
					#step 14 output gatk unified genotyper
					${$exomes{$sampleID}{SNPs_only}} =
"\"$output_dir/$output_dirs[12]/$coinID\_SNPs_only.vcf\"";
						#step 14 & 15 output SNPs gatk select variants
					${$exomes{$sampleID}{indels_only}} =
"\"$output_dir/$output_dirs[12]/$coinID\_indels_only.vcf\"";
						#step 14 & 15 output indels select variants
					${$exomes{$sampleID}{SNPs_recal}} =
"\"$output_dir/$output_dirs[13]/$coinID\_SNPs.csv\"";
					#step 16 output SNPs gatk variant recalibrator
					${$exomes{$sampleID}{indels_recal}} =
"\"$output_dir/$output_dirs[13]/$coinID\_indels.csv\"";
					#step 16 output SNPs gatk variant recalibrator
					${$exomes{$sampleID}{SNPs_tranches}} =
"\"$output_dir/$output_dirs[13]/$coinID\_SNPs.tranches\"";
					#step 16 output SNPs gatk variant recalibrator
					${$exomes{$sampleID}{indels_tranches}} =
"\"$output_dir/$output_dirs[13]/$coinID\_indels.tranches\"";
					#step 16 output indels gatk variant recalibrator
					${$exomes{$sampleID}{SNPs_recalibrated}} =
"\"$output_dir/$output_dirs[14]/$coinID\_SNPs_recalibrated.vcf\"";
				#step 17 output SNPs gatk apply recalibration
					${$exomes{$sampleID}{indels_recalibrated}} =
"\"$output_dir/$output_dirs[14]/$coinID\_indels_recalibrated.vcf\"";
				#step 17 output indels gatk apply recalibration
					${$exomes{$sampleID}{recalibrated_variants_combined}} =
"\"$output_dir/$output_dirs[15]/$coinID\_all_recalibrated_variants.vcf\"";			#step
18 output gatk combine variants
					${$exomes{$sampleID}{snpEff_output}} =
"\"$output_dir/$output_dirs[16]/$coinID\_snpEff_output.vcf\"";
					#step 19 output snpEff
					${$exomes{$sampleID}{variants_annotated}} =
"\"$output_dir/$output_dirs[17]/$coinID\_variants_annotated.vcf\"";
				#step 20 output gatk variant annotator
				}
			}

			# for $keys (sort keys %exomes){
				# print color ("blue"), "read_exomes (hash generation) keys: $keys\n";
				# print color 'reset';
			# }
		}

		return (\%exomes);
}
###############################################################################
####################################################
#end of sub read_exomes
###############################################################################
####################################################

###############################################################################
####################################################
#subroutine subway
#Search subdirectories of a specified directory for FASTQ files.
```

```
###############################################################################
##################################################
sub subway{
	#get the directory.
	my $dir = shift;

	#open the directory.
	opendir FQ, "$dir" or die $!;

	#Read in all files from the directory (excluding the "." and ".." files which may be
present), and store the full filepaths in an array.
	my @sub_dirs = map{$dir."/".$_} grep {"$dir/$_" && !/^\.{1,2}$/} readdir FQ;

	#foreach file and sub directory within the current directory.
	foreach $sub_dir (sort {"\L$a" cmp "\L$b"} @sub_dirs){
		#If the current file matches a certain criteria ('-f' specifies a file).
		if (-f "$sub_dir" && $sub_dir =~ /(fastq|fq)$/i){
			#Get the sample name (COIN ID) from the full file path.
			my $coin = $sub_dir;
			my $multi_sample;

			#remove the file path and file extension, along with "_pe" that some file
names may have.
			$coin =~ s/^.+\///g;
			$coin =~ s/\..+$//;
			if ($coin =~ /\_pe/i){$coin =~ s/$&//;}

			my @coin = split (/\-/, $coin);

			$coin[2] =~ s/\_.+$//;

			$coin = "$coin[0]-$coin[1]-$coin[2]";

			next if $coin =~ /001/i;print "$coin\n";
			# next unless $coin =~ /NR6/i;

			if ($coin =~ /\-[a-z]+|Undetermined/){$multi_sample =
"multiSample".$&;}
			else{$multi_sample = "multiSample";}

			#Captures FASTQ filepaths in a hash of arrays.
			unless (grep {$sub_dir eq $_} sort {"\L$a" cmp "\L$b"}
@{$fq_reads{$coin}}){
				push @{$fq_reads{$coin}}, $sub_dir;
			}

			unless (grep {"$multi_sample\_2" eq $_} sort{"\L$a" cmp "\L$b"}
@{$fq_reads{$multi_sample}}){
				push @{$fq_reads{$multi_sample}}, "$multi_sample\_2";
			}
		}

		#if the directory contains sub directories, store them in the array and call this
sub routine on them.
		#the result is each sub directory within a directory will be searched before
moving onto the next directory;
		#e.g. a/1/i and a/1/ii will be searched before a/2/i, etc.
		if (-d $sub_dir){push @sub_dirs, subway($sub_dir);}
	}
```

```
    return (\%fq_reads, \@sub_dirs);
}
###############################################################################
####################################################
#end of sub subway
###############################################################################
####################################################

###############################################################################
####################################################
#subroutine let_there_be_output
#creates directories is necessary for the output from the pipeline.
###############################################################################
####################################################
sub let_there_be_output{
    my ($output_dir, $output_dirs) = @_;
    my @output_dirs = @$output_dirs;
    my $create_dir;

    while ($create_dir ne "Y" && $create_dir ne "n" && $create_dir ne "s"){
        print "\nAre you sure you want to create output folders in ".$output_dir."? Enter
'Y' for yes or 'n' for no.\n";
        $create_dir = &ask;

        if ($create_dir eq "Y"){
            #Unless it already exists create an output directory for each folder
name in @output_dir
            foreach my $sub_dir (sort {"\L$a" cmp "\L$b"} @output_dirs){
                unless (-d "$output_dir/$sub_dir"){
                    mkpath("$output_dir/$sub_dir") or print "could not
create $output_dir/$sub_dir\n";
                    print "Creating folder: \"$output_dir/$sub_dir\".\n";
                }
            }
        }

        #user doesn't want to create dir.
        if ($create_dir ne "Y" && $create_dir ne "n" && $create_dir ne "s"){
            print "$output_dir not created. Please enter either 'Y' or 'n' only.\n";
            $create_dir = "";
        }
    }

    return ($output_dir, $create_dir);
}
###############################################################################
####################################################
#end of sub read_exomes.
###############################################################################
####################################################

###############################################################################
##############################################################
#subroutine trim.
#Calls trimmomatic to trim Illumina adapter sequences from the ends of reads.
###############################################################################
##############################################################
sub trim{
    my ($exomes, $trimmomatic_jar, $seq_adapters, $cores) = @_;
    my %exomes = %{$exomes};
```

```
	for my $sampleID (sort keys %exomes){
		next if $sampleID =~ /multiSample/;
		my ($coinID, $mate, $RG_headline, $mate_file);

		#Copies the name of the fastq file and then removes the sequencing/mate
information fom the copied variable.
	#It is essential to still retain the mate information for the samples as 'bwa aln' will use
this. The copied variable will be used for naming output files for steps further in the pipeline.
		if ($sampleID =~ /(\_.+$)/){
			$mate = $&;
			$coinID = $sampleID;

			$coinID =~ s/$&//;
			$mate =~ s/\_//;
		}

		$mate_1_trimmed = ${$exomes{$sampleID}{mate_1_file}};
		$mate_2_trimmed = ${$exomes{$sampleID}{mate_2_file}};

		$mate_1_trimmed =~ s/\.(fq|fastq)/\_trimmed\.fq/i;
		$mate_2_trimmed =~ s/\.(fq|fastq)/\_trimmed\.fq/i;

		my $trimlog = $mate_2_trimmed;
		$trimlog =~ s/trimmed\.fq/trimlog/i;

		if ($mate == 2){
			unless (-f "$mate_1_trimmed"){
				push @{$system_calls{$coinID}},
					# "java -jar \"$trimmomatic_jar\" PE -threads $cores -
phred33 -trimlog $trimlog ${$exomes{$sampleID}{mate_1_file}}
${$exomes{$sampleID}{mate_2_file}} ${$exomes{$sampleID}{mate_1_paired}}
${$exomes{$sampleID}{mate_1_unpaired}} ${$exomes{$sampleID}{mate_2_paired}}
${$exomes{$sampleID}{mate_2_unpaired}} ILLUMINACLIP:\"$seq_adapters\":2:30:10:1:true
MINLEN:2",
					# "cat ${$exomes{$sampleID}{mate_1_paired}}
${$exomes{$sampleID}{mate_1_unpaired}} > $mate_1_trimmed",
					# "rm ${$exomes{$sampleID}{mate_1_paired}}
${$exomes{$sampleID}{mate_1_unpaired}}",
					# "cat ${$exomes{$sampleID}{mate_2_paired}}
${$exomes{$samp=leID}{mate_2_unpaired}} > $mate_2_trimmed",
					# "rm ${$exomes{$sampleID}{mate_2_paired}}
${$exomes{$sampleID}{mate_2_unpaired}}";
			}
		}

		${$exomes{$sampleID}{mate_1_file}} = $mate_1_trimmed;
		${$exomes{$sampleID}{mate_2_file}} = $mate_2_trimmed;
	}

	return (\%exomes, \%system_calls);
}

##############################################################################
##################################################
#end of sub trim.
##############################################################################
##################################################
```

```
##########################################################################
########################################################
#subroutine system_calls
#Generates a list of commands for each sample, which are then executed sample by sample.
##########################################################################
########################################################
sub system_calls{
	my ($output_dirs, $system_calls_input, $exomes, $system_calls) = @_;

	my $output_dirs = @$output_dirs;
	my @system_calls_input = @$system_calls_input;
	my %exomes = %{$exomes};
	my %system_calls = %{$system_calls};

	my ($call_regions, $bed_file, $trimmomatic_jar, $seq_adapters, $create_index,
$add_ann, $fastq_reads_dir, $output_dir, $reference_genome, $gatk_jar, $snpEff_jar, $dbsnp,
$hapmap_vcf, $omni_vcf, $phase1, $mills_Devine_vcf, $platform, $index_algorithm,
$bwa_ended, $bigS, $who_ya_gonna_call, $pro_multi, $cores) = @system_calls_input;

	my @bam;

	my %months = (
		Jan => 1,  Feb => 2,  Mar => 3,  Apr => 4, May => 5,  Jun => 6,
		Jul => 7, Aug => 8, Sep => 9, Oct => 10, Nov => 11, Dec => 12
	);

	my @log_time = split (/\s+/, localtime);
	my ($day, $month, $date, $time, $year) = @log_time;

	$month = $months{$month};

	my $log_date = "$year\-$month\-$date\_$time";

	my $log = "$output_dir/command log $log_date.txt";



	my @dna_types;

	my $threads = $cores / 2;

	if ($threads =~ /\.\d+$/){
		$threads =~ s/$&//;
		$threads++;
	}

	#####################################################################
	#This section of &system_calls calls BWA index to index the reference genome.
	#####################################################################

      #1.	Create reference genome index and dictionary with bwa index:
	#a.	-p			Prefix of the output, same as reference filename inc. file
extension.
	#b.	-a			Algorithm for constructing BWT index.  Available options
are:
		#b.i.	is		For small (<2Gb) genomes.
		#b.ii.	bwtsw		For larger (>2Gb) genomes.
	#c.	<input_Reference_Genome.fasta>  The reference genome.
	#Time: 87 mins   Size: 7.36GB (inc. Downloaded reference).
	if ($create_index eq "y"){
```

```
                if (-f "$reference_genome\.fai"){print "Genome has already been indexed,
skipping this step.\n";}
                else{
                        #BWA first creates the dictionary (.dict), etc. for the genome, then
SAMtools creates the '.fai. file. Both steps are necessary.
                        print "Now indexing the reference genome $reference_genome. This
takes a while so put the kettle on and do something productive!\n";
                        `bwa index -p \"$reference_genome\" -a $index_algorithm
\"$reference_genome\"`;
                        `samtools faidx \"$reference_genome\"`;
                };
        }
        ######################################################################

        ######################################################################
        #This section of &system_calls generates the commands for each step of the pipeline
        #for each sample.
        #       -       The first 2 steps (bwa aln and bwa sampe/samse/bwasw) are
performed
        #               for both mates, if you have paired end data.
        #       -       The remaining steps are performed only on 2nd mates as the mates
will be
        #               merged into a single file from here on.
        ######################################################################
        for my $sampleID (sort {"\L$a" cmp "\L$b"} keys %exomes){
                next if $sampleID =~ /multiSample/;
                my ($coinID, $mate, $RG_headline, $mate_file);

                my @sample = split (/\-/, $sampleID);

                my ($run, $coin, $tissue) = @sample;

                $tissue =~ s/\_.+$//;

                $coinID = "$run-$coin-$tissue";

                #Copies the name of the fastq file and then removes the sequencing/mate
information fom the copied variable.
        #Removes from UNC samples (e.g. 042031_02136_s_6_[1|2]_sequence) then from
BGI samples (e.g. 3018_2)
        #It is essential to still retain the mate information for the samples as 'bwa aln' will use
this. The copied variable will be used for naming output files for steps further in the pipeline.
                if ($sampleID =~ /(\_\d$)/){
                        $mate = $&;
                        $mate =~ s/\_//;
                }

                if ($coinID =~ /\-[a-z]+$/){
                        my $dna_type = $&;
                        $dna_type =~ s/^\-//;
                        #unless (grep {$dna_type eq $_} sort {"\L$a" cmp "\L$b"}
@dna_types){push @dna_types, "$dna_type";}
                }

                #the read group information for the sample.
                $RG_headline = ${$exomes{$sampleID}{RG_headline}};

                $mate_file = "mate\_".$mate."\_file";

                #2. With bwa aln find SA coordinates for input reads:
```

```
        #a. < reference_Genome.fasta>                The reference genome.
        #b. < query_Reads.fastq>                     Raw sequence reads should
be in fastq format.
        #c. > <Ouput.sai>                     The output from this step.
        #Note: using the '- t xx' switch allows you to specify the number of threads used
by BWA, even though it is not documented on their website.
        push @{$system_calls{$coinID}}, "bwa aln -t $cores \"$reference_genome\"
${$exomes{$sampleID}{$mate_file}} > ${$exomes{$sampleID}{aln_sai}}";

        if ($mate == 2){
            if ($platform ne "LS454"){
                #3.     If you have Illimina/Solexa reads: Use bwa sampe (for
paired end reads, or bwa samse for single ends) to produce alignments into SAM format:
                #a.     -r "@RG\tID:x\tSM:x\tPL:x"                Adds read
group header to output. Replace 'x' withID/sample/platform names.
                #b.     < reference_Genome.fasta>                  The reference
genome.
                #c.     <query_Reads.sai>                          Output from the
prev. Step, for both mates.
                #d.     <query_Reads.fastq>                        The raw
sequencing reads in FASTQ format. For both mates, in same order as input for .sai files.
                #e.     > <Ouput.sam>                   The output from this step.
                # push @{$system_calls{$coinID}}, "bwa $bwa_ended -r
\"$RG_headline\" \"$reference_genome\" ${$exomes{$sampleID}{aln_sai_prev}}
${$exomes{$sampleID}{aln_sai}} ${$exomes{$sampleID}{mate_1_file}}
${$exomes{$sampleID}{mate_2_file}} > ${$exomes{$sampleID}{sam}}";
            }

            else{ #use bwasw instead of bwa samse/sampe
                #If you have LifeScience 454 reads: Use bwasw to align reads
onto reference genome:
                #a.     <reference_Genome.fasta>
                #b.     <query_Reads.fastq>
                # push @{$system_calls{$coinID}}, "bwasw
$reference_genome $fastq_reads_dir/$fastq_reads.$file_extension";
            }


    ######################################################################
################################################################
            push @{$system_calls{$coinID}},
            #All the quoted lines before the next line of hashes (#) will be added to
the hash of arrays for the sample.

    ######################################################################
################################################################
                #4. Convert output of step 3 into BAM format using samtools
view:
                #a. -bSh                                   b outputs to BAM
format S indicates input is SAM format. h adds header to the .bam file.
                #b. <query_Reads.sam>                            Input
                #c. > <query_Reads.bam>                          A
"Â˜.bam"Â™ file will be created using the same name as this file.
                "samtools view -bSh ${$exomes{$sampleID}{sam}} >
${$exomes{$sampleID}{view_bam}}",
                # "rm ${$exomes{$sampleID}{sam}}",     #Delete the output
from the previous step for this sample as it is now not needed. These rm steps are not
necessary but save a LOT of disk space!

                #5.     Sort reads in BAM file by name with samtools sort:
```

#a. -n Sort by read name instead of chr position (this is necessary at first).
#b. <query_Reads.bam> The output file from step 4.
#c. <Sorted.bam> The output from this step.
"samtools sort -n ${$exomes{$sampleID}{view_bam}} ${$exomes{$sampleID}{sorted_bam}}\"",
"rm ${$exomes{$sampleID}{view_bam}}", #Delete the directory containing the output from the previous step as it is now not needed.

#6. Use samtools fixmate to fill in read mate details:
#a. <Sorted.bam> The output from prev. step.
#b. <Fixmate.bam> The output from this step.
"samtools fixmate ${$exomes{$sampleID}{sorted_bam}}\.bam\" ${$exomes{$sampleID}{fixmate_bam}}",
"rm ${$exomes{$sampleID}{sorted_bam}}\.bam\"",

#Note: if you ran step 5 with the "-n" switch then run step 5 again without "-n" (using the output from step 6 as the input)
#before proceeding to step 7._
"samtools sort ${$exomes{$sampleID}{fixmate_bam}} ${$exomes{$sampleID}{postfixmate_sorted_bam}}\"",

#7. use samtools rmdup to remove duplicate PCR reads:
#a. -s or -S For single/paired end reads.
#b. <Fixmate.bam> The output from step 6 (after running step 5 again).
#c. <NoDups.bam> The output from this step.
# "samtools rmdup -$bigS ${$exomes{$sampleID}{postfixmate_sorted_bam}}\.bam\" ${$exomes{$sampleID}{nodups_bam}}",

#8. Use samtools index to create an index of the latest output:
#a. <noDups.bam> The output from the prev. Step.
"samtools index ${$exomes{$sampleID}{postfixmate_sorted_bam}}\.bam\"",

#9. Advised optional step: indel realignment, to ensure regions around indels do not contain false SNPs. This is a 2 stage step, which requires first creating a file of known indel regions. This file is then used to align reads around these regions. Type java -Xmx2g -jar GenomeAnalysisTK.jar with the following switches:
#a. -T RealignerTargetCreator Name of the tool in the GATK package.
#b. -I <NoDups.bam> Input is the output from prev. step.
#c. -R <reference_Genome.fasta> The reference genome.
#d. -o <Indels.intervals> Output from this step. Notice lower case 'o'.
#Note: if the fastq files contain Illumina-encoded qualities instead of Sanger-encoded qualities, use -fixMisencodedQuals (specify IMMEDIATELY AFTER the input file, with just a space inbetween). Hiseq data usually need this switch on, Miseq data do not.
#Sometimes, if using modified BAM files (i.e. merged from multiple samples or selected reads from FASTQ/BAM files) GATK may throw an error regarding

a mix of differently encoded quality scores. In these cases, try also specifying -allowPotentiallyMisencodedQuals next to -fixMisencodedQuals.
"java -jar \"$gatk_jar\" -T RealignerTargetCreator -I ${$exomes{$sampleID}{postfixmate_sorted_bam}}\.bam\" -R \"$reference_genome\" -o ${$exomes{$sampleID}{indels_intervals}} -nt $threads -filterNoBases",

#10. For the second stage in indel realignment, type java -Xmx4g -jar GenomeAnalysisTK.jar and use the following options:
#a. -T IndelRealigner Tool in the GATK package.
#b. -I <NoDups.bam> Input is output from step 7.
#c. -R <reference_Genome.fasta> The reference genome.
#d. -targetIntervals <Indels.intervals> Input is output from prev. step.
#e. -o <Realigned.bam> Output of this step.
#Note: if the fastq files contain Illumina-encoded qualities instead of Sanger-encoded qualities, use -fixMisencodedQuals (specify IMMEDIATELY AFTER the input file, with just a space inbetween). Hiseq data usually need this switch on, Miseq data do not.
#Sometimes, if using modified BAM files (i.e. merged from multiple samples or selected reads from FASTQ/BAM files) GATK may throw an error regarding a mix of differently encoded quality scores. In these cases, try also specifying -allowPotentiallyMisencodedQuals next to -fixMisencodedQuals.
"java -jar \"$gatk_jar\" -T IndelRealigner -I ${$exomes{$sampleID}{postfixmate_sorted_bam}}\.bam\" -R \"$reference_genome\" -targetIntervals ${$exomes{$sampleID}{indels_intervals}} -o ${$exomes{$sampleID}{realigned_bam}} -filterNoBases",

#11. Step may not be needed! Using SAMtools again now, type samtools mpileup to generate a pileup format file (easier to use for variant calls):
#a. -E -Uses extended BAQ computation.
#b. <Realigned.bam> Input is output from prev. Step.
#c. > <mPileUp.bam> Output from this step.
"samtools mpileup -E ${$exomes{$sampleID}{realigned_bam}} > ${$exomes{$sampleID}{mpileup_bam}}",

#12. Advised optional step: quality score recalibration, first creates a table of scores. Type java -Xmx4g -jar GenomeAnalysisTK.jar and use the following switches:
#a. -T BaseRecalibrator The tool in the GATK package.
#b. -I <Input realigned.bam > Input is output from step 10.
#c. -R <reference_Genome.fasta> The reference genome.-R /media/marc/Brachiosaur/Exomes/reference genome/human_g1k_v37.fasta
#d. -l INFO Lower case 'L', not a capital 'i'.
#e. -recalFile <output file.csv> Output (comma separated variable .csv file).
#f. -knownSites <dbsnp_132.hg19.vcf> File of known variants from a database (e.g. dbSNP).
#g. --default_platform illumina May be needed if not given in step 3 '-r' switch.
#h. -cov Type '-cov', a space and 1 of the following options (do this for all 4 options):
#h.i. ReadGroupCovariate

```
                                        #h.ii.   QualityScoreCovariate
                                        #h.iii.  CycleCovariate
                                        #h.iv.   DinucCovariate
                                        "java -jar \"$gatk_jar\" -T BaseRecalibrator -I
${$exomes{$sampleID}{realigned_bam}} -R \"$reference_genome\" -l INFO -o
${$exomes{$sampleID}{recalibrated_grp}} -knownSites \"$dbsnp\" -cov ReadGroupCovariate -
cov QualityScoreCovariate -cov CycleCovariate -cov ContextCovariate -nct $cores -
filterNoBases",

                                        #13.     Second part of quality score recalibration, merges new
Q scores into a recalibrated '.bam' file. Use java -Xmxg4 -jar GenomeAnalysisTK.jar:
                                        #a.      -T TableRecalibration                          The
tool in the GATK package.
                                        #b.      -I <Input '.bam' file>                    Input is the
output from step 10.
                                        #c.      -R <reference genome.fasta>                    The
reference genome.
                                        #d.      -l INFO
            Lower case 'L', not a capital 'i'.
                                        #e.      -recalFile <'.csv' file>                       Input
is the '.csv' from prev. Step.
                                        #f.     -o <Recalibrated.bam>     Name of the new recalibrated
'.bam' file.
                                        #          -baq RECALCULATE                             If
using after GATK version 2.5, include this switch. Introduced to the pipeline for version 2.4-9 but
caused a bug.
                                        "java -jar \"$gatk_jar\" -T PrintReads -I
${$exomes{$sampleID}{realigned_bam}} -R \"$reference_genome\" -l INFO -BQSR
${$exomes{$sampleID}{recalibrated_grp}} -o ${$exomes{$sampleID}{recalibrated_bam}} -nct
$cores -filterNoBases";

            ######################################################################
##################################################################

                        push @bam, ${$exomes{$sampleID}{recalibrated_bam}};
                }
        }

        #At this stage, you may wish to pool the data so far into a single bam file. This will allow
for higher confidence in rare variants or those with low depth. You can separate the VCF file
into individual samples after variant calling using SelectVariants.
        #The GSA recommend analysing all samples separately up to this point.

        # opendir BAM, "$output_dir\/$output_dirs[10]" or die "Cannot open
$output_dir\/$output_dirs[10].\n";
        # my @bam = map {$output_dir."/".$output_dirs[10]."/".$_} grep {-f
"$output_dir\/$output_dirs[10]/$_" && /\.bam$/i} readdir (BAM);

        #Multi-sample processing, if the DNA is from several tissue sources (e.g. somatic
FFPE, normal FFPE, germline, etc).
        if (@dna_types){
                foreach my $dna_type (sort {"\L$a" cmp "\L$b"} @dna_types){
                        # next unless $dna_type eq "f";
                        my $multi_samples = "";

                        foreach my $bam (sort {"\L$a" cmp "\L$b"} @bam){
                                next unless $bam =~ /\-$dna_type/;

                                $multi_samples .= "\-I \"$bam\" ";
                        }
```

```
				my $sampleID = "multiSample\-".$dna_type."\_2";
				my $coinID = "multiSample\-".$dna_type;

				# $system_calls = multi_sample_calls(\%exomes,
\@system_calls_input, \%system_calls, $multi_samples, $sampleID, $coinID, $threads);
				%system_calls = %{$system_calls};
			}

			execute_sub(\%system_calls, $log);
		}

		#Multi sample processing if the DNA is all from the same tissue source (e.g. all
germline, etc).
		elsif ($pro_multi eq "y"){
			my $multi_samples = "";

			foreach $bam (sort {"\L$a" cmp "\L$b"} @bam){$multi_samples .= "\-I \"$bam\"
";}

			my $sampleID = "multiSample\_2";
			my $coinID = "multiSample";

			my $system_calls = multi_sample_calls(\%exomes, \@system_calls_input,
\%system_calls, $multi_samples, $sampleID, $coinID, $threads);
			%system_calls = %{$system_calls};

			execute_sub(\%system_calls, $log);
		}

		#currently set up to do single sample processing. Can easily be switched back to do
multi-sample work - see comments.
		else{
			my $multi_samples = "";
			print "multi calling steps\n";
			foreach my $bam (sort{"\L$a" cmp "\L$b"} @bam){
				$multi_samples = "\-I $bam\"";

				if ($bam =~ /\_recalibrated\.bam/i){
					$sampleID = $bam;
					$sampleID =~ s/$&//i;
					$sampleID =~ s/\"//g;
					$sampleID =~ s/.+\///g;
					$coinID = $sampleID;
					$sampleID = "$sampleID\_2";
				}

				next unless grep {$sampleID eq $_} sort {"\L$a" cmp "\L$b"} keys
%exomes;

				my $system_calls = multi_sample_calls(\%exomes,
\@system_calls_input, \%system_calls, $multi_samples, $sampleID, $coinID, $threads);
				%system_calls = %{$system_calls};
			}

			execute_sub(\%system_calls, $log);
		}
}
#end of sub system_calls
```

```
##########################################################################
##########################################################################
##########

##########################################################################
##########################################################################
##########
#multi_sample_calls
##########################################################################
##########################################################################
##########
sub multi_sample_calls{
	my ($exomes, $system_calls_input, $system_calls, $multi_samples, $sampleID,
$coinID, $threads) = @_;
	my %exomes = %{$exomes};
	my @system_calls_input = @$system_calls_input;
	my %system_calls = %{$system_calls};

	chop $multi_samples;
	$multi_samples =~ s/^\-I //;

	my ($call_regions, $bed_file, $trimmomatic_jar, $seq_adapters, $create_index,
$add_ann, $fastq_reads_dir, $output_dir, $reference_genome, $gatk_jar, $snpEff_jar, $dbsnp,
$hapmap_vcf, $omni_vcf, $phase1, $mills_Devine_vcf, $platform, $index_algorithm,
$bwa_ended, $bigS, $who_ya_gonna_call, $pro_multi, $cores) = @system_calls_input;

	my $region_switch = "";
	if ($call_regions eq "y"){$region_switch = "-L \"$bed_file\"";}

	#14.	Use java -jar GenomeAnalysisTK.jar to call variants in the pileup file from
previous step:
	#a.	-T UnifiedGenotyper		The tool in the GATK package.
	#b.	-R <reference genome.fasta>		The reference genome.
	#c.	-glm				Calls following type of variant:
		#c.i.	SNP
		#c.ii.	INDEL
		#c.iii.	BOTH
	#d.	-D <dbsnp_xxx.vcf>			Adds rsIDs to output file
	#e.	-I <input '.bam' file>		Input is the output from step 13.
	#f.	-o <output.vcf>			Output file (in VCF format).
	#g.	-nct x				where x is number of cores available.
	#h.	-metrics <metrics.txt>		metrics file on the variant calls
	# if ($who_ya_gonna_call eq "SNP"){
		# push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T UnifiedGenotyper -
R \"$reference_genome\" -glm $who_ya_gonna_call $region_switch -mbq 10 -stand_call_conf
30 -stand_emit_conf 10 -baq CALCULATE_AS_NECESSARY -D \"$dbsnp\" -I $multi_samples -
o ${$exomes{$sampleID}{SNPs_only}} -metrics ${$exomes{$sampleID}{metrics}} -nt $cores -
nct $threads";
	# }
	# elsif ($who_ya_gonna_call eq "INDEL"){
		# push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T UnifiedGenotyper -
R \"$reference_genome\" -glm $who_ya_gonna_call $region_switch -mbq 10 -stand_call_conf
30 -stand_emit_conf 10 -baq CALCULATE_AS_NECESSARY -D \"$dbsnp\" -I $multi_samples -
o ${$exomes{$sampleID}{indels_only}} -metrics ${$exomes{$sampleID}{metrics}} -nt $cores -
nct $threads";
	# }
	# else{
		# push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T UnifiedGenotyper -
R \"$reference_genome\" -glm $who_ya_gonna_call $region_switch -mbq 10 -stand_call_conf
30 -stand_emit_conf 10 -baq CALCULATE_AS_NECESSARY -D \"$dbsnp\" -I $multi_samples -
o ${$exomes{$sampleID}{variant_calls}} -metrics ${$exomes{$sampleID}{metrics}} -nt $cores -
nct $threads";
	# }
	#use HaplotypeCaller instead of UnifedGenotyper.
	push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T HaplotypeCaller -R
\"$reference_genome\" $region_switch -stand_call_conf 30 -stand_emit_conf 10 -D \"$dbsnp\" -I
$multi_samples -o ${$exomes{$sampleID}{variant_calls}} -nct $cores";

	#15. Use java -jar GenomeAnalysisTK.jar to validate variants called by the
UnifiedGenotyper. This step is optional but I'd recommend it.
	#a.	-T ValidateVariants
	#b.	-R <reference genome.fasta>
	#c.	-V <input.vcf>
	#d.	-D <dbSNP_xxx.vcf>
	if ($who_ya_gonna_call eq "SNP"){
		push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T ValidateVariants -R
\"$reference_genome\" -V ${$exomes{$sampleID}{SNPs_only}} -D \"$dbsnp\"";
	}
	elsif ($who_ya_gonna_call eq "INDEL"){
		push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T ValidateVariants -R
\"$reference_genome\" -V ${$exomes{$sampleID}{indels_only}} -D \"$dbsnp\"";
	}
	else{
		push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T ValidateVariants -R
\"$reference_genome\" -V ${$exomes{$sampleID}{variant_calls}} -D \"$dbsnp\"";
	}

    #16.   Use  java -jar GenomeAnalysisTK.jar to separate out indels from SNPs. Run it
twice to create 2 files, one for each type of variant:
	#a.     -T SelectVariants                    The tool in the GATK package.
	#b.     -R <reference genome.fasta>          The reference genome.
	#c.     -V <input.vcf>                       Input is output from prev. step/step 14.
	#d.     -selectType
	          #d.i.   SNP
	          #d.ii.  INDEL
	#e.     -o <output.vcf>                      Output from this step.
	push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T SelectVariants -R
\"$reference_genome\" -V ${$exomes{$sampleID}{variant_calls}} -selectType SNP -o
${$exomes{$sampleID}{SNPs_only}}",
	"java -jar \"$gatk_jar\" -T SelectVariants -R \"$reference_genome\" -V
${$exomes{$sampleID}{variant_calls}} -selectType INDEL -o
${$exomes{$sampleID}{indels_only}}";

    #17.	Use java -jar GenomeAnalysisTK.jar to produce a table for variant recalibration.
This will need to be run twice, once for SNPs and again for indels:
	#a.	-T VariantRecalibrator                  The tool in the GATK package.
	#b.	-R <reference genome.fasta>             The reference genome.
	#c.	-input <input.vcf>                      Output from prev. step.
	#d.	-recalFile <output.csv>                 Output for use in next step.
	#e.	-tranchesFile <output.tranches>         Output for use in next step.
	#f.	-mode                                   Enter only SNP or INDEL.
	#	f.i.	SNP
	#	f.ii.	INDEL
	#g.	-an
	#	g.ii.	HaplotypeScore
	#	g.iii.	MQRankSum
	#	g.iv.	ReadPosRankSum
	#	g.v.	MQ
	#	g.vi.	DP
	#h.	-resource:				Training data (first 3 for SNPs, fourth
for
	#						indels only?). Leave no space between
';' and
	#						the following options (type -'resource:'
for
	#						each).
	#	h.i.	-resource:hapmap,VCF,known=false,training=true,truth=true,prior=15.0
<hapmap_3.3.hg19.vcf>
	#	h.ii.	-resource:dbsnp,VCF,known=true,training=false,truth=false,prior=8.0
<dbsnp_132_vcf>
	#	h.iii	-resource:omni,known=false,training=true,truth=false,prior=12.0
<1000G_omni2.5.b37.sites.vcf>
	#	h.iiv.	-resource:mills,VCF,known=true,training=true,truth=true,prior=12.0
<Mills_Devine_indels.sites.vcf>
      #i.	-mG 4				Use for small (<30 exomes) datasets.
	#k.	-nt x					where x is number of threads
specified to the tool. Note this is not -nct, the number of cores specified.
	if ($who_ya_gonna_call eq "SNP"){
		# push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T
VariantRecalibrator -R \"$reference_genome\" -input ${$exomes{$sampleID}{SNPs_only}} -
recalFile ${$exomes{$sampleID}{SNPs_recal}} -tranchesFile
${$exomes{$sampleID}{SNPs_tranches}} -mode SNP -an MQRankSum -an ReadPosRankSum
-an FS -an QD -resource:hapmap,VCF,known=false,training=true,truth=true,prior=15.0
\"$hapmap_vcf\" -resource:omni,known=false,training=true,truth=false,prior=12.0 \"$omni_vcf\" -
resource:1000G,known=false,training=true,truth=false,prior=10.0 \"$phase1\" -
resource:dbsnp,VCF,known=true,training=false,truth=false,prior=8.0 \"$dbsnp\" -mG 4";
		push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T VariantFiltration -R
\"$reference_genome\" -V ${$exomes{$sampleID}{SNPs_only}} --filterExpression \"QD < 2.0 ||
FS > 60.0 || MQ < 35.0 || MappingQualityRankSum < -12.5 || ReadPosRankSum < -8.0\" --
filterName \"Filter\"  -o ${$exomes{$sampleID}{SNPs_recalibrated}}";
	}
	elsif ($who_ya_gonna_call eq "INDEL"){
		# push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T
VariantRecalibrator -R \"$reference_genome\" -input ${$exomes{$sampleID}{indels_only}} -
recalFile ${$exomes{$sampleID}{indels_recal}} -tranchesFile
${$exomes{$sampleID}{indels_tranches}} -mode INDEL -an MQRankSum -an
ReadPosRankSum -an FS -resource:mills,VCF,known=true,training=true,truth=true,prior=12.0
\"$mills_Devine_vcf\" -mG 4";
		push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T VariantFiltration -R
\"$reference_genome\" -V ${$exomes{$sampleID}{indels_only}}  --filterExpression \"QD < 2.0 ||
FS > 200.0 || ReadPosRankSum < -20.0\" --filterName \"Filter\" -o
${$exomes{$sampleID}{indels_recalibrated}}";
	}
	else{
		# push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T
VariantRecalibrator -R \"$reference_genome\" -input ${$exomes{$sampleID}{SNPs_only}} -
recalFile ${$exomes{$sampleID}{SNPs_recal}} -tranchesFile
${$exomes{$sampleID}{SNPs_tranches}} -mode SNP -an MQRankSum -an ReadPosRankSum
-an FS -an QD -resource:hapmap,VCF,known=false,training=true,truth=true,prior=15.0
\"$hapmap_vcf\" -resource:omni,known=false,training=true,truth=false,prior=12.0 \"$omni_vcf\" -
resource:1000G,known=false,training=true,truth=false,prior=10.0 \"$phase1\" -
resource:dbsnp,VCF,known=true,training=false,truth=false,prior=8.0 \"$dbsnp\" -mG 4",
		# "java -jar \"$gatk_jar\" -T VariantRecalibrator -R \"$reference_genome\" -input
${$exomes{$sampleID}{indels_only}} -recalFile ${$exomes{$sampleID}{indels_recal}} -
tranchesFile ${$exomes{$sampleID}{indels_tranches}} -mode INDEL -an MQRankSum -an
ReadPosRankSum -an FS -resource:mills,VCF,known=true,training=true,truth=true,prior=12.0
\"$mills_Devine_vcf\" -mG 4";
		push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T VariantFiltration -R
\"$reference_genome\" -V ${$exomes{$sampleID}{SNPs_only}} --filterExpression \"QD < 2.0 ||
FS > 60.0 || MQ < 35.0 || MappingQualityRankSum < -12.5 || ReadPosRankSum < -8.0\" --
filterName \"Filter\" -o ${$exomes{$sampleID}{SNPs_recalibrated}}";
                push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T VariantFiltration -R
\"$reference_genome\" -V ${$exomes{$sampleID}{indels_only}} --filterExpression \"QD < 2.0 ||
FS > 200.0 || ReadPosRankSum < -20.0\" --filterName \"Filter\" -o
${$exomes{$sampleID}{indels_recalibrated}}";
        }

    #18.        Use java -jar GenomeAnalysisTK.jar to apply the tables from the previous step
to your SNPs and indels (run this twice, once for each):
        #a.     -T ApplyRecalibration                   The tool in the GATK package.
        #b.     -R <reference genome.fasta>             The reference genome.
                #c.     -input <input.vcf>                      Input is output from step 14.
        #d.     -recalFile <input.csv>                  Input is output from prev. step.
        #e.     -tranchesFile <input.tranches>          Input is output from prev. step.
        #f.     -o <ouput_recalibrated.vcf>             Output of this step.
        # if ($who_ya_gonna_call eq "SNP"){
                # push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T ApplyRecalibration
-R \"$reference_genome\" -input ${$exomes{$sampleID}{SNPs_only}} -recalFile
${$exomes{$sampleID}{SNPs_recalibrated}} -tranchesFile
${$exomes{$sampleID}{SNPs_tranches}} -o ${$exomes{$sampleID}{SNPs_recalibrated}}";
        # }
        # elsif($who_ya_gonna_call eq "INDEL"){
                # push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T ApplyRecalibration
-R \"$reference_genome\" -input ${$exomes{$sampleID}{indels_only}} -recalFile
${$exomes{$sampleID}{indels_recalibrated}} -tranchesFile
${$exomes{$sampleID}{indels_tranches}} -o ${$exomes{$sampleID}{indels_recalibrated}}";
        # }
        # else{
                # push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T ApplyRecalibration
-R \"$reference_genome\" -input ${$exomes{$sampleID}{SNPs_only}} -recalFile
${$exomes{$sampleID}{SNPs_recalibrated}} -tranchesFile
${$exomes{$sampleID}{SNPs_tranches}} -o ${$exomes{$sampleID}{SNPs_recalibrated}}",
                # "java -jar \"$gatk_jar\" -T ApplyRecalibration -R \"$reference_genome\" -input
${$exomes{$sampleID}{indels_only}} -recalFile ${$exomes{$sampleID}{indels_recalibratedre}} -
tranchesFile ${$exomes{$sampleID}{indels_tranches}} -o
${$exomes{$sampleID}{indels_recalibrated}}";
        # }

        #19.    Use  java -jar GenomeAnalysisTK.jar to recombine the data for SNPs and
indels. This can also be used to check the output from this pipeline to that of others:
    #You can use CombineVariants to combine your SNPs with those from another pipeline or
lab followed by SelectVariants using -select 'set == "Intersection"Â• ' to output a file with just the
SNPs found in both datasets.
        #a.     -T CombineVariants              The tool in the GATK package.
        #b.     -R <reference genome.fasta>     The reference genome.
        #c.     -V                              Output from the prev.
Step."/media/marc/Brachiosaur/Exomes/CHD
output/09_gatk_indelrealigner/004031_02145_s_1_realigned.bam"
        #       c.i.    <input_SNPs.vcf>
        #       c.ii.   <input_INDELs.vcf>
        #d.     -genotypeMergeOptions           May not have to specify, but if you do you're
not bothered about genotypes then use 'unsorted'.
        #       d.i.    -UNSORTED
        #       d.ii.   -PRIORITIZE
        #       d.iii.  -UNIQUIFY
        #       d.iv.   -REQUIRE_UNIQUE
        #e.     -o <output_recombined.vcf>
        if ($who_ya_gonna_call eq "BOTH"){push @{$system_calls{$coinID}}, "java -jar
\"$gatk_jar\" -T CombineVariants -R \"$reference_genome\" -V:SNP
```

```
${$exomes{$sampleID}{SNPs_recalibrated}} -V:Indel
${$exomes{$sampleID}{indels_recalibrated}} -genotypeMergeOptions UNSORTED -o
${$exomes{$sampleID}{recalibrated_variants_combined}}";}

        #20.    use snpEff to do some snp effect stuff.
        #a.     snpEff.jar
        #b      eff
        #c      -c path to/snpEff.config
        #d      -i vcf
        #e      -o vcf
        #f      GRCh37.63
        #g      <input vcf>
        #h      > snpEff_output.vcf
        if ($add_ann eq "y"){push @{$system_calls{$coinID}}, "java -jar \"$snpEff_jar\" eff -c
\"$snpEff_config\" -i vcf -o gatk GRCh37.75
${$exomes{$sampleID}{recalibrated_variants_combined}} >
${$exomes{$sampleID}{snpEff_output}}";}

        #21.    Use  java -jar GenomeAnalysisTK.jar to add annotation to the variant file:
        #a.     -T VariantAnnotator             The tool in the GATK package.
        #b.     -R <reference genome.fasta>     The reference genome.
        #c.     -A SnpEff                       God only knows what this is for.
        #d.     -V <input.vcf>                  Input is the output from combine variants step.
        #e.     -snpEffFile <input.vcf>         The file generated by running snpEff.
        #f.     -o <output_annotated.vcf>       Annotated output from this step.
        if ($add_ann eq "y"){push @{$system_calls{$coinID}}, "java -jar \"$gatk_jar\" -T
VariantAnnotator -R \"$reference_genome\" -A SnpEff -V
${$exomes{$sampleID}{recalibrated_variants_combined}} -snpEffFile
${$exomes{$sampleID}{snpEff_output}} -o ${$exomes{$sampleID}{variants_annotated}}";}

        return \%system_calls;
}
##########################################################################
##########################################################################
##########
#End of sub multi_sample_calls.
##########################################################################
##########################################################################
##########

##########################################################################
##########################################################################
##########
#subroutine executre
#Executes the system commands for the pipeline.
##########################################################################
##########################################################################
##########
sub execute_sub{
        my ($system_calls, $log) = @_;
        my %system_calls = %{$system_calls};

        for my $sample (sort {"\L$a" cmp "\L$b"} keys %system_calls){
                my $step_no = 0;
                my $total_steps = scalar @{$system_calls{$sample}};

                my ($prev_time, $date, $time_elapsed, $total_hours, $total_minutes,
$total_seconds) = outta_time($prev_time, time(), $total_hours, $total_minutes, $total_seconds);

                open LOG, ">>$log" or die $!;
```

```
			foreach my $call (@{$system_calls{$sample}}){
				$step_no++;

				$call_print = $call;
				#$call_print =~ s/\s/\n/g;

				print color ("blue"), "Sample: $sample.\tNow performing step
$step_no/$total_steps\nDate: $date command line:\n";
				print color ("green"), "$call_print\n";
				print color 'reset';

				print LOG "--------------------------------------------------------------------------
----\n\>>Date: $date\tSample: $sample.\nNow performing step $step_no/$total_steps, command
as follows:\n------------------------------------------------------------------------------------\n";
				print LOG "$call_print\n\n";

				# system ($call); #Execute the current command


				($prev_time, $date, $time_elapsed, $total_hours, $total_minutes,
$total_seconds) = outta_time($prev_time, time(), $total_hours, $total_minutes, $total_seconds);
				print color ("yellow"), "The date and time is now: $date. The previous
step took $time_elapsed seconds to run.\nTotal run time (hh:mm:ss):
$total_hours:$total_minutes:$total_seconds\n\n";
				print color 'reset';
			}

			print "\n\n\n";
			print LOG "\n\n";
	}

	close LOG;
}
################################################################################
################################################################################
##########
#End of sub execute.
################################################################################
################################################################################
##########

################################################################################
################################################################################
##########
#sub outta_time
#calculates the time taken to perform a step of the pipeline.
################################################################################
################################################################################
##########
sub outta_time{
	my ($prev_time, $current_time, $total_hours, $total_minutes, $total_seconds) = @_;
	my ($elapsed_hours, $elapsed_minutes, $elapsed_seconds);

	#get the current time and date and reorder for printing.
	my $localtime = localtime;
	my @time = split (/\s+/, $localtime);
	my ($day, $month, $date, $time, $year) = @time;
	my $date = "$year\-$month\-$date\-$time";
	#calculate how long has elapsed since the last time this subroutine was called,
therefore revealing how long the last pipeline step took.
	if (defined $prev_time){
		$elapsed_seconds = $current_time - $prev_time;
	}

	else{$elapsed_seconds = 0;}

	$elapsed_hours = sprintf("%02d", int($elapsed_seconds/3600)%3600);
	$elapsed_minutes = sprintf("%02d", int($elapsed_seconds/60)%60);
	$elapsed_seconds = sprintf("%02d", ($elapsed_seconds%60));

	$total_hours += $elapsed_hours;
	$total_minutes += $elapsed_minutes;
	$total_seconds += $elapsed_seconds;

	$total_hours = sprintf("%02d", int($total_seconds/3600)%3600);
	$total_minutes = sprintf("%02d", int($total_seconds/60)%60);
	$total_seconds = sprintf("%02d", ($total_seconds%60));

	my $time_elapsed = "$elapsed_hours\:$elapsed_minutes\:$elapsed_seconds";

	return ($current_time, $date, $time_elapsed, $total_hours, $total_minutes,
$total_seconds);
}
################################################################################
################################################################################
##########
#End of sub outta_time
################################################################################
################################################################################
##########
```

Custom Script 2

```
#filter_tuner.pl
#This script is designed to read in variant from a VCF file and print them out to a CSV file along
with their pass/filter status for a number of metrics used in the pipeline. Filters according to the
best practices for VQSR unclude:
	#Mapping quality >40 (>35 for the adjusted pilot project parameters).
	#Quality by depth >2.
	#Fisher strand bias <60.
	#Haplotype score <13 (this filter is ignored for the adjusted parameters in the pilot
project to maximise the number of known variants passing).
	#Mapping quality rank sum >-12.5.
	#Read pos rank sum >-8.
#It can also read in a list of known variants and retrieve just this list from the VCF files along
with filter status. This script is therefore useful for checking parameter sets used to filter out
potential false positives/negatives.
#It also tells which of the other tissues a variant was called in. If given a set of coordinates for
CCDS, the script can potentially also inform you if a variant lies in an exon or not.

($date, $time) = &delorean;

print "$date\n";

#The CCDS file.
$ccds = "/media/marc/Brachiosaur/Exomes/database resources/CCDS_2013_10_24.txt";

$dbSNP = "/media/marc/Brachiosaur/Exomes/database resources/dbSNP_141_b37.vcf";
$kg = "/media/marc/Brachiosaur/Exomes/database
resources/ALL.wgs.phase1_release_v3.20101123.snps_indels_sv.sites.vcf";
$esp_dir = "/media/marc/Brachiosaur/Exomes/database resources/Exome sequencing project
6500SI_V2_SSA137";
$cosmic = "/media/marc/Brachiosaur/Exomes/database
resources/CosmicCompleteExport_v70_100814 sorted.tsv";

$ref_file = "/media/marc/Brachiosaur/Exomes/reference genome/human_g1k_v37.fasta";

#The output file that the final list of variants is printed to.
$out_file = "/home/marc/Documents/work/reseq analysis/non responder project/Output/MuTect
variants - alien amplicons ($date).csv";
$variant_list = "/home/marc/Documents/work/reseq analysis/pilot project/Output/pilot project -
list of variants ($date).csv";

#The VCF directories.
$nr_dir = "/media/marc/Triceratops/Exomes/Non responder output/19_GATK_variantannotator";
$mutect_dir = "/media/marc/Triceratops/Exomes/Non responder output/MuTect output/alien
amplicons";

#This array will be used to pass the VCF dirs to a dir reader subroutine later.
@vcf_dirs = ($mutect_dir);

%tissues = (
	a => "tFFPE",
	d => "nFFPE",
	e => "blood",
	f => "WGA tFFPE",
	u => "UDG tFFPE",
	du => "UDG nFFPE",
	Undetermined => "Undetermined"
);

%tissue_presence = (
	"blood" => "N",
	"nFFPE" => "N",
	"tFFPE" => "N",
	"UDG nFFPE" => "N",
	"UDG tFFPE" => "N",
	"Undetermined" => "N",
	"WGA tFFPE" => "N"
);

($variants, $runs, $final_chr) = get_variants(\@vcf_dirs, \%tissues, $ref_file);

$variants = dbSNP_reader($variants, $final_chr, $dbSNP);
$variants = kg_reader($variants, $final_chr, $kg);
$variants = esp_reader($variants, $final_chr, $esp_dir);
$variants = cosmic_reader($variants, $final_chr, $cosmic);

printer($variants, $out_file, $variant_list, \%tissues, \%tissue_presence, $runs);

###########################################################################
#subroutine delorean
#Gets the local time and converts it into a date (format: yyyy-mm-dd) and time (hh:mm:ss).
#These are then returned,
###########################################################################
sub delorean{
	my @flux_capacitor = localtime(time);

	#Set the variables for the time and date. Note: year is a count of years after 1900 (e.g.
2015 = 115), and month is 0-based integer (0 = Jan, 11 = Dec).
	my ($sec, $min, $hour, $date, $month, $year) = @flux_capacitor;

	$month += 1;
	$year += 1900;

	$month = sprintf ("%02d", $month);
	$date = sprintf ("%02d", $date);
	$day = sprintf ("%02d", $day);
	$sec = sprintf ("%02d", $sec);
	$min = sprintf ("%02d", $min);
	$hour = sprintf ("%02d", $hour);

	my $return_date = "$year-$month-$date";

	my $return_time = "$hour:$min:$sec";

	return ($return_date, $return_time);
}
###########################################################################
#End of subroutine delorean.
###########################################################################

################################################################################
####################
#Subroutine get_variants.
#Opens up a VCF directory, reads in the VCF files and then reads in the variants and returns
them in a hash.
```

```
##############################################################################
####################
sub get_variants{
	my ($vcf_dirs, $tissues, $ref_file) = @_;

	my @vcf_dirs = @$vcf_dirs;

	my %tissues = %{$tissues};

	my (%variants, @runs);

	my $final_chr;

	foreach my $vcf_dir (sort {"\L$a" cmp "\L$b"} @vcf_dirs){
		my ($vcf_count, @vcf_files);

		print "Reading variants:\n$vcf_dir\n";

		opendir VCF_DIR, "$vcf_dir" or die "$vcf_dir not found!\n";

		if ($vcf_dir =~ /Brachiosaur/){@vcf_files = map {"$vcf_dir/$_"} grep {-f
"$vcf_dir/$_" && !/genome/i && /\.vcf$/i} readdir VCF_DIR;}
		else{@vcf_files = map {"$vcf_dir/$_"} grep {-f "$vcf_dir/$_" &&
/(variants\_annotated)?\.vcf$/i} readdir VCF_DIR;}

		closedir VCF_DIR;

		foreach my $vcf (sort {"\L$a" cmp "\L$b"} @vcf_files){
			$vcf_count++;

			my $sample = $vcf;

			$sample =~ s/^.+\///g;
			$sample =~ s/\_.+$//g;

			my @sample = split (/-/, $sample);

			my ($run, $coin, $tissue) = @sample;

			if ($tissue =~ /\s.+/){$tissue =~ s/$&//;}print "$run\t$coin\t$tissue\n";

			unless (grep {$run eq $_} sort {"\L$a" cmp "\L$b"} @runs){push @runs,
$run;}

			unless (defined $tissue){$tissue = "Undetermined";}

			$tissue = $tissues{$tissue};

			# next unless $coin =~ /(14030|62015)/;
			# next if $run =~ /NR3/;

			print "Sample $vcf_count: $coin\tTissue: $tissue\tRun: $run\n";

			open VCF, "$vcf" or die $!;

			while (my $line = <VCF>){
				chomp $line;

				my ($gene_name, $amino_acid, $transcript, $depth_ref,
$depth_alt, $preceeding_base, $proceeding_base);
```

```
				my $allele_i = 0;

				next if $line =~ /^\#/;

				my @fields = split (/\s+/, $line);

				my ($chr, $pos, $rsID, $ref_allele, $alt_allele, $qual, $filter,
$info, $format_header, $format) = @fields;

				if ($chr =~ /^chr/i){$chr =~ s/$&//;}

				# last if $chr > 1;
				# next if $filter eq "REJECT";

				if ($chr > $final_chr){$final_chr = $chr;}

				my $pos_end = ($pos + length $ref_allele) - 1;

				#samtools faidx
				my $flank_start = $pos - 1;
				my $flank_end = $pos_end + 1;

				my $region = $chr.":".$flank_start."-".$flank_end;

				my $flank_sequence = `samtools faidx "$ref_file" $region`;

				$flank_sequence =~ s/^\>$region//i;
				$flank_sequence =~ s/\n//g;

				if ($flank_sequence =~ /^[A-Z]/i){$preceeding_base = $&;}
				if ($flank_sequence =~ /[A-Z]$/i){$proceeding_base = $&;}

				$flank_sequence = $preceeding_base."_".$proceeding_base;

				if ($info =~ /SNPEFF_GENE_NAME\=[A-Z0-9\-\.]+\;/i){
					$gene_name = $&;

					$gene_name =~ s/^.+\=//;
					$gene_name =~ s/\;$//;
				}

				else{$gene_name = "Unspecified";}

				# next unless exists $known_variants{$coin}{$gene_name};

				if ($info =~ /AMINO_ACID_CHANGE\=[A-Z0-9\*]+\;/i){
					$amino_acid = $&;

					$amino_acid =~ s/^.+\=//;
					$amino_acid =~ s/\;$//;
				}

				else{$amino_acid = "Unspecified";}

				if ($info =~ /SNPEFF_TRANSCRIPT_ID\=[A-Z\d]+\;/i){
					$transcript = $&;

					$transcript =~ s/^.+\=//;
					$transcript =~ s/\;$//;
				}
```

```
					else{$transcript = "Unspecified";}

					my @format = split (/\:/, $format);
					my $allele_depths = $format[1];

					if ($run =~ /miseq/i){$allele_depths = $format[2];}

					my @allele_depths = split (/,/, $allele_depths);

					my $ref_depth = $allele_depths[0];

					# next unless (exists
$known_variants{$coin}{$gene_name}{$amino_acid} && exists
$known_variants{$coin}{$gene_name}{$amino_acid}{$chr} && exists
$known_variants{$coin}{$gene_name}{$amino_acid}{$chr}{$pos});

					my @alt_alleles = split (/,/, $alt_allele);

					foreach my $var_allele (sort {"\L$a" cmp "\L$b"} @alt_alleles){
						my ($mq, $qd, $fs, $haplotype_score, $mqrs,
$read_pos_rank, $snpeff_effect);

						$allele_i++;

						my $allele_depth = $allele_depths[$allele_i];

						my ($mq_filter, $qd_filter, $fs_filter,
$haplotype_score_filter, $mqrs_filter, $read_pos_rank_filter) = ("Pass", "Pass", "Pass", "Pass",
"Pass", "Pass");

						my $nucleotide_change = $ref_allele.">".$var_allele;

						# next unless
$known_variants{$coin}{$gene_name}{$amino_acid}{$chr}{$pos} = $nucleotide_change;

						if ($info =~ /MQ\=[0-9\.]+\;/i){
							$mq = $&;

							$mq =~ s/^MQ\=//i;
							$mq =~ s/\;$//;
						}

						else{$mq = "Unspecified";}

						if ($info =~ /QD\=[0-9\.]+\;/i){
							$qd = $&;

							$qd =~ s/^QD\=//i;
							$qd =~ s/\;$//;
						}

						else{$qd = "Unspecified";}

						if ($info =~ /FS\=[0-9\.]+\;/i){
							$fs = $&;

							$fs =~ s/^FS\=//i;
							$fs =~ s/\;$//;
						}
					else{$fs = "Unspecified";}

					if ($info =~ /HaplotypeScore\=[0-9\.]+\;/i){
						$haplotype_score = $&;

						$haplotype_score =~ s/^HaplotypeScore\=//i;
						$haplotype_score =~ s/\;$//;
					}

					else{$haplotype_score = "Unspecified";}

					if ($info =~ /MQRankSum\=[0-9\.]+\;/i){
						$mqrs = $&;

						$mqrs =~ s/^MQRankSum\=//i;
						$mqrs =~ s/\;$//;
					}

					else{$mqrs = "Unspecified";}

					if ($info =~ /ReadPosRankSum\=[0-9\.]+\;/i){
						$read_pos_rank = $&;

						$read_pos_rank =~ s/^ReadPosRankSum\=//i;
						$read_pos_rank =~ s/\;$//;
					}

					else{$read_pos_rank = "Unspecified";}

					if ($info =~ /SNPEFF_EFFECT\=[a-z0-9\-\_]+\;/i){
						$snpeff_effect = $&;

						$snpeff_effect =~ s/^.+\=//i;
						$snpeff_effect =~ s/\;$//;
					}

					else{$snpeff_effect = "Unspecified";}

					#These are the filters used to mark variants as filtered
or pass.
					if ($mq < 35){$mq_filter = "Filter";}
					if ($qd < 2){$qd_filter = "Filter";}
					if ($fs > 60){$fs_filter = "Filter";}
					if ($haplotype_score > 13){$haplotype_score_filter =
"Filter";}
					if ($mqrs < -12.5){$mqrs_filter = "Filter";}
					if ($read_pos_rank < -8){$read_pos_rank_filter =
"Filter";}


	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"gene
name"} = $gene_name;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"transcr
ipt"} = $transcript;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"snpEff
effect"} = $snpeff_effect;
```

```
	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"rsID"}
= $rsID;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"flankin
g sequence"} = $flank_sequence;


	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"filter"}
= $filter;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"mappi
ng quality"} = $mq;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"quality
by depth"} = $qd;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"strand
bias"} = $fs;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"haplot
ype score"} = $haplotype_score;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"map
quality rank sum"} = $mqrs;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"read
pos rank sum"} = $read_pos_rank;


	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"mappi
ng quality filter"} = $mq_filter;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"quality
by depth filter"} = $qd_filter;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"strand
bias filter"} = $fs_filter;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"haplot
ype score filter"} = $haplotype_score_filter;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"map
quality rank sum filter"} = $mqrs_filter;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"read
pos rank sum filter"} = $read_pos_rank_filter;


	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"ref
depth"} = $ref_depth;

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"var
depth"} = $allele_depth;


	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"dbSNP
presence"} = "N";
```

```
	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"KG
presence"} = "N";

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"ESP
presence"} = "N";

	$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSM
IC presence"} = "N";
					}
				}

				close VCF;
			}
	}

	return (\%variants, \@runs, $final_chr);
}
################################################################################
####################
#End of subroutine get_variants.
################################################################################
####################

################################################################################
################################################################################
#######
#dbSNP_reader
#Opens dbSNP and checks whether each dbSNP line is in the %variants hash.
#If it is in the hash, then it is deleted so that what remains in the hash are novel variants with
respect to dbSNP.
################################################################################
################################################################################
#######
sub dbSNP_reader{
	my ($variants, $final_chr, $dbSNP) = @_;
	my %variants = %{$variants};

	my $chromosome;

	#open the dbSNP file and read it in line by line.
	open dbSNP, "$dbSNP" or die "Cannot open dbSNP, did you mount the drive?\n";

	while ($dbSNP_line = <dbSNP>){
		chomp $dbSNP_line;
		next if $dbSNP_line =~ /^\#/;

		my @dbSNP_line = split (/\t/, $dbSNP_line);

		my $chr = $dbSNP_line[0];
		my $pos = $dbSNP_line[1];
		my $rsID = $dbSNP_line[2];
		my $ref_allele = $dbSNP_line[3];
		my $alt_allele = $dbSNP_line[4];
		my $info = $dbSNP_line[7];

		last if $chr > $final_chr;

		unless ($chromosome eq $chr){print "Now searching dbSNP, chromosome
$chr.\n";}
```

```
			$chromosome = $chr;

			#set the allele frequencies for each world population for the variant.
			if ($info =~ /AF=[\d\.]+/i){
				$af = $&;
				$af =~ s/AF\=//i;
			}

			else{$af = "NA";}

			#although the majority of cases are mono-allelic, some are multi-allelic and so
must be accounted for.
			#this will split all alleles for a variant into an array (the array will only contain 1
allele if mono-allelic)
			my @alt_alleles = split (/\,/, $alt_allele);

			#foreach alt allele at the site...
			foreach my $alt_allele (@alt_alleles){
				#...make a unique key to identify variants, as one position can have
many different variants (e.g. 2 SNPs, many indels or a SNP and an indel).
				my $allele_key = $ref_allele."\>".$alt_allele;

				#if the exomes hash contains the allele_key at the chromosomal
position, then the variant is not unique and must be marked.
				#I think it is necessary to have these 3 exist statements as if you just
have the final 1, you risk creating $exomes{$chr}{$pos} if $exomes{$chr}{$pos}{$allele_key}
				#doesn't exist.
				#$variants{$chr}{$pos}{$allele_key}{$sample}}
				if ((exists $variants{$chr}) && (exists $variants{$chr}{$pos}) && (exists
$variants{$chr}{$pos}{$allele_key})){
					for $amino_acid (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}}){
						for $run (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}{$amino_acid}}){
							for $tissue (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}}){
								for $coin (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}}){

	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"dbSNP
presence"} = "Y";

	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"dbSNP AF"} =
$af;
								}
							}
						}
					}
				}
			}
		}

		close dbSNP;

		#return %variants with only novel variants (that appear in multiple samples).
		return \%variants;
}
###############################################################################
###############################################################################
#######
```

```
#End of dbSNP_reader
#############################################################################
#############################################################################
#######

#############################################################################
#############################################################################
#######
#kg_reader
#Opens 1000 genomes and checks whether each 1kg line is in the %variants hash.
#If it is in the hash, then it is deleted so that what remains in the hash are novel variants with
respect to 1kg.
#############################################################################
#############################################################################
#######
sub kg_reader{
        my ($variants, $final_chr, $kg) = @_;
        my %variants = %{$variants};

        my ($af, $asn_af, $afr_af, $eur_af);

        #open the kg file and read it in line by line.
        open KG, "$kg" or die "Cannot open kg, did you mount the drive?\n";

        while ($kg_line = <KG>){
                chomp $kg_line;
                next if $kg_line =~ /^\#/;

                my @kg_line = split (/\t/, $kg_line);

                my $chr = $kg_line[0];
                my $pos = $kg_line[1];
                my $rsID = $kg_line[2];
                my $ref_allele = $kg_line[3];
                my $alt_allele = $kg_line[4];
                my $info = $kg_line[7];

                last if $chr > $final_chr;

                unless ($chromosome eq $chr){print "Now searching kg, chromosome $chr.\n";}
                $chromosome = $chr;

                #set the allele frequencies for each world population for the variant.
                if ($info =~ /AF=[\d\.]+/i){
                        $af = $&;
                        $af =~ s/AF\=//i;
                }

                else{$af = "NA";}

                if ($info =~ /ASN_AF=[\d\.]+/i){
                        $asn_af = $&;
                        $asn_af =~ s/ASN_AF\=//i;
                }

                else{$asn_af = "NA";}

                if ($info =~ /AMR_AF=[\d\.]+/i){
                        $amr_af = $&;
                        $amr_af =~s/AMR_AF\=//i;
```

```
		}

		else{$amr_af = "NA";}

		if ($info =~ /AFR_AF=[\d\.]+/i){
			$afr_af = $&;
			$afr_af =~ s/AFR_AF\=//i;
		}

		else{$afr_af = "NA";}

		if ($info =~ /EUR_AF=[\d\.]+/i){
			$eur_af = $&;
			$eur_af =~ s/EUR_AF\=//i;
		}

		else{$eur_af = "NA";}

		#although the majority of cases are mono-allelic, some are multi-allelic and so
must be accounted for.
		#this will split all alleles for a variant into an array (the array will only contain 1
allele if mono-allelic)
		my @alt_alleles = split (/\,/, $alt_allele);

		#foreach alt allele at the site...
		foreach my $alt_allele (@alt_alleles){
			#...make a unique key to identify variants, as one position can have
many different variants (e.g. 2 SNPs, many indels or a SNP and an indel).
			my $allele_key = $ref_allele."\>".$alt_allele;

			if ((exists $variants{$chr}) && (exists $variants{$chr}{$pos}) && (exists
$variants{$chr}{$pos}{$allele_key})){
				for $amino_acid (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}}){
					for $run (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}{$amino_acid}}){
						for $tissue (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}}){
							for $coin (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}}){

	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"KG presence"}
= "Y";


	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"KG AF"} = $af;

	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"KG AFR AF"} =
$afr_af;

	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"KG AMR AF"} =
$amr_af;

	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"KG ASN AF"} =
$asn_af;

	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"KG EUR AF"} =
$eur_af;
							}
						}
```

```
                                        }
                                }
                        }
                }
        }

        close KG;

        #return %variants with only novel variants.
        return \%variants;
}
###############################################################################
###############################################################################
#######
#End of kg_reader
###############################################################################
###############################################################################
#######

###############################################################################
###############################################################################
#######
#esp_reader
#Opens esp and checks whether each esp line is in the %variants hash.
#If it is in the hash, then it is deleted so that what remains in the hash are novel variants with
respect to esp.
###############################################################################
###############################################################################
#######
sub esp_reader{
        my ($variants, $final_chr, $esp_dir) = @_;
        my %variants = %{$variants};

        opendir ESP, "$esp_dir" or die "Cannot open $esp_dir.\n";

        my @esp_files = grep {-f "$esp_dir/$_" && /\.(vcf)$/i} readdir(ESP);

        closedir ESP;

        #open the esp file and read it in line by line.
        foreach $esp_file (sort {"\L$a" cmp "\L$b"} @esp_files){
                open esp_file, "$esp_dir/$esp_file" or die "Cannot open ESP file, did you mount
the drive?\n";

                while ($esp_line = <esp_file>){
                        chomp $esp_line;
                        next if $esp_line =~ /^\#/;

                        my @esp_line = split (/\t/, $esp_line);

                        my $chr = $esp_line[0];
                        my $pos = $esp_line[1];
                        my $rsID = $esp_line[2];
                        my $ref_allele =$esp_line[3];
                        my $alt_allele = $esp_line[4];
                        my $info = $esp_line[7];

                        my @info = split (/\;/, $info);

                        my $maf = $info[4];
```

```
				$maf =~ s/^MAF\=//;

				my @maf = split (/\,/, $maf);

				my $ea_maf = $maf[0];
				my $aa_maf = $maf[1];
				my $total_maf = $maf[2];

				if ($total_maf eq ""){$total_maf = "NA";}
				if ($ea_maf eq ""){$ea_maf = "NA";}
				if ($aa_maf eq ""){$aa_maf = "NA";}

				last if $chr > $final_chr;

				unless ($chromosome eq $chr){print "Now searching esp, chromosome
$chr.\n";}
				$chromosome = $chr;

				#although the majority of cases are mono-allelic, some are multi-allelic
and so must be accounted for.
				#this will split all alleles for a variant into an array (the array will only
contain 1 allele if mono-allelic)
				my @alt_alleles = split (/\,/, $alt_allele);

				#foreach alt allele at the site...
				foreach my $alt_allele (@alt_alleles){
					#...make a unique key to identify variants, as one position can
have many different variants (e.g. 2 SNPs, many indels or a SNP and an indel).
					my $allele_key = $ref_allele."\>".$alt_allele;

					if ((exists $variants{$chr}) && (exists $variants{$chr}{$pos}) &&
(exists $variants{$chr}{$pos}{$allele_key})){
						for $amino_acid (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}}){
							for $run (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}{$amino_acid}}){
								for $tissue (sort {"\L$a" cmp "\L$b"}
keys %{$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}}){
									for $coin (sort {"\L$a" cmp
"\L$b"} keys %{$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}}){

	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"ESP presence"}
= "Y";


	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"ESP AF"} =
$total_maf;

	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"ESP AA AF"} =
$aa_maf;

	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"ESP EA AF"} =
$ea_maf;
									}
								}
							}
						}
					}
				}
```

```
		}

		close esp_file;
	}

	#return %variants with only novel variants (that appear in multiple samples).
	return \%variants;
}
###########################################################################
###########################################################################
#######
#End of esp_reader
###########################################################################
###########################################################################
#######

###########################################################################
###########################################################################
#######
#sub cosmic_reader
#Reads in COSMIC .tsv and extracts info on relevent variants.
###########################################################################
###########################################################################
#######
sub cosmic_reader{
	my ($variants, $final_chr, $cosmic_file) = @_;

	my %variants = %{$variants};

	my %complement = (
		A => "T",
		C => "G",
		G => "C",
		T => "A"
	);

	my %cosmic_counts;

	my $chromosome;

	print "Now searching COSMIC. Please be patient.\n";

	open COSMIC, "$cosmic_file" or die $!;

	while (my $cosmic_line = <COSMIC>){
		chomp $cosmic_line;

		my ($allele_key, $complementary_key);

		next if $cosmic_line =~ /Gene name/;

		my @fields = split (/\t/, $cosmic_line);

		my $gene = $fields[0];
		my $accession_id = $fields[1];
		my $primary_site = $fields[7];
		my $site_subtype  = $fields[8];
		my $mutation_CDS = $fields[13];
		my $mutation_AA = $fields[14];
		my $mutation_description = $fields[15];
```

```
			my $genome_coords = $fields[17];
			my $fathmm_prediction = $fields[21];
			my $somatic_status = $fields[22];
			my $pubmed_id = $fields[23];
			my $id_study = $fields[24];
			my $sample_source = $fields[25];
			my $tumour_origin = $fields[26];

			my @genome_coords = split (/[\:\-]/, $genome_coords);
			my $chr = $genome_coords[0];
			my $pos = $genome_coords[1];

			last if $chr > $final_chr;

			unless ($chromosome eq $chr){print "Now searching COSMIC, chromosome
$chr.\n";}
			$chromosome = $chr;

			if ($mutation_CDS =~ /([A-Z]+\>[A-Z]+)|([\d\_]+(ins|del)[A-Z]+)/i){
				$allele_key = $&;
			}

			$alllele_key =~ tr/a-z/A-Z/;

			if ($allele_key =~ /\>/){
				my @alleles = split (/\>/, $allele_key);

				my $ref_allele = $alleles[0];
				my $var_allele = $alleles[1];

				my $ref_comp = $complement{$ref_allele};
				my $var_comp = $complement{$var_allele};

				$complementary_key = "$ref_comp>$var_comp";

				if ((exists $variants{$chr}) && (exists $variants{$chr}{$pos}) && (exists
$variants{$chr}{$pos}{$complementary_key})){$allele_key = $complementary_key;}
			}

			if ((exists $variants{$chr}) && (exists $variants{$chr}{$pos}) && (exists
$variants{$chr}{$pos}{$allele_key})){
				for $amino_acid (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}}){
					for $run (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}{$amino_acid}}){
						for $tissue (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}}){
							for $coin (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}}){

	$variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC
presence"} = "Y";

								$cosmic_counts{"$run $tissue $coin
$chr:$pos $allele_key"}++;

								my $cosmic_count =
$cosmic_counts{"$run $tissue $coin $chr:$pos $allele_key"};
```

```
                                                        #Due to limits imposed by MS excel
and LibreOffice Calc, you cannot have too many characters per cell (limit unknown).
                                                        #COSMIC may contain many entries
for a given variant, so limiting the number of entries stored in the output file will hopefully not
overload the end product.
                                                        next if $cosmic_count >= 10;


        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC
accession ID"} .= "[$cosmic_count]$accession_id;";

        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC
primary site"} .= "[$cosmic_count]$primary_site;";

        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC site
subtype"} .= "[$cosmic_count]$site_subtype;";

        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC CDS"}
.= "[$cosmic_count]$mutation_CDS;";

        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC AA"}
.= "[$cosmic_count]$mutation_AA;";

        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC
description"} .= "[$cosmic_count]$mutation_description;";

        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC
coords"} .= "[$cosmic_count]$genome_coords;";

        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC
FATHMM prediction"} .= "[$cosmic_count]$fathmm_prediction;";

        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC
somatic status"} .= "[$cosmic_count]$somatic_status;";

        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC
pubmed ID"} .= "[$cosmic_count]$pubmed_id;";

        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC study
ID"} .= "[$cosmic_count]$id_study;";

        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC
sample source"} .= "[$cosmic_count]$sample_source;";

        $variants{$chr}{$pos}{$allele_key}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC
tumour origin"} .= "[$cosmic_count]$tumour_origin;";
                                                }
                                        }
                                }
                        }
                }
        }

        close COSMIC;

        return \%variants;
}
################################################################################
################################################################################
#######
```

```
#End of cosmic_reader
#############################################################################
#############################################################################
#######

#############################################################################
#####################
#Subroutine printer.
#Takes a hash of variants and prints them out to a CSV file.
#############################################################################
#####################
sub printer{
        my ($variants, $out_file, $variant_list, $tissues, $tissue_presence, $runs) = @_;

        my %variants = %{$variants};

        my %tissues = %{$tissues};

        my @runs = @$runs;

        my %rev_tissues = reverse %tissues;

        open OUT, ">$out_file" or die $!;

        print OUT "\#Run,Tissue,COIN ID,Gene,Transcript,Chr,Pos,rsID,Nucleotide change,AA
change,No. samples called in,Ref allele depth,Variant allele depth,snpEff effect,Flanking
nucleotides,Overall Filter,Map Qual,MQ Filter,Qual by Depth,QD Filter, Fisher Strand Bias,FS
Filter,Haplotype Score,HS Filter,MQ Rank Sum,MQRS Filter,Read Pos Rank Sum,RPRS
Filter,";

        foreach my $run (sort {"\L$a" cmp "\L$b"} @runs){print OUT "Present in $run,"}

        print OUT "Present in germline (any sample),Present in nFFPE (any sample),Present in
tFFPE (any sample),Present in UDG nFFPE (any sample),Present in UDG tFFPE (any
sample),Present in undetermined (any sample),Present in WGA tFFPE (any sample),Present in
germline (by sample),Present in nFFPE (by sample),Present in tFFPE (by sample),Present in
UDG nFFPE (by sample),Present in UDG tFFPE (by sample),Present in undetermined (by
sample),Present in WGA tFFPE (by sample),Present in dbSNP,Present in 1000
Genomes,Present in Exome Sequencing Project,Present in COSMIC,dbSNP allele
frequency,1KG global allele frequency,1KG AFR frequency,1KG AMR frequency,1KG ASN
frequency,1KG EUR frequency,ESP total frequency,ESP AA frequency,ESP EA
frequency,COSMIC accession ID,COSMIC primary site,COSMIC site subtype,COSMIC
coords,COSMIC CDS,COSMIC AA,COSMIC description,COSMIC FATHMM
prediction,COSMIC somatic status,COSMIC pubmed ID,COSMIC study ID,COSMIC sample
source,COSMIC tumour origin\n";

        for my $chr (sort {"\L$a" cmp "\L$b"} keys %variants){
                for my $pos (sort {$a <=> $b} keys %{$variants{$chr}}){
                        for my $nucleotide_change (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}}){
                                my $sample_count = 0;

                                for my $amino_acid (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$nucleotide_change}}){
                                        for my $run (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}}){
                                                for my $tissue (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}}){
                                                        $sample_count += scalar keys
%{$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}}
```

```
                                                }
                                            }
                                        }

                                        for my $amino_acid (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$nucleotide_change}}){
                                            for my $run (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}}){
                                                for my $tissue (sort {"\L$a" cmp "\L$b"} keys
%{$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}}){
                                                    #Use these lines of code if you want to
look at presence of a variant in a tissue across all samples.
                                                    my %tissue_presence_overall =
%{$tissue_presence};

                                                    for my $other_run (sort {"\L$a" cmp
"\L$b"} @runs){
                                                        for my $other_tissue (sort
{"\L$a" cmp "\L$b"} keys %rev_tissues){
                                                            #Is the variant present
in any other tissue across all runs?
                                                            if (exists
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$other_run} && exists
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$other_run}{$other_tissue}){

    $tissue_presence_overall{$other_tissue} = "Y";
                                                            }
                                                        }
                                                    }

                                                    for my $coin (sort {"\L$a" cmp "\L$b"}
keys %{$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}}){
                                                        #Use this if you want to look at
presence of a variant in a tissue for a specific sample.
                                                        my %tissue_presence =
%{$tissue_presence};

                                                        my $snpeff_effect =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"snpEff effect"};
                                                        my $flanking_sequence =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"flanking
sequence"};
                                                        my $rsID =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"rsID"};

                                                        my $gene_name =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"gene name"};
                                                        my $transcript =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"transcript"};
                                                        my $filter =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"filter"};
                                                        my $mq =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"mapping
quality"};
                                                        my $qd =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"quality by
depth"};
                                                        my $fs =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"strand bias"};
```

```
                                        my $haplotype_score =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"haplotype
score"};
                                        my $mqrs =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"map quality
rank sum"};
                                        my $read_pos_rank =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"read pos rank
sum"};

                                        my $mq_filter =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"mapping
quality filter"};
                                        my $qd_filter =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"quality by
depth filter"};
                                        my $fs_filter =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"strand bias
filter"};
                                        my $haplotype_score_filter =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"haplotype
score filter"};
                                        my $mqrs_filter =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"map quality
rank sum filter"};
                                        my $read_pos_rank_filter =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"read pos rank
sum filter"};

                                        my $ref_depth =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"ref depth"};
                                        my $var_depth =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"var depth"};

                                        my $dbSNP_presence =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"dbSNP
presence"};
                                        my $kg_presence =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"KG
presence"};
                                        my $esp_presence =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"ESP
presence"};
                                        my $cosmic_presence =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC
presence"};

                                        my $dbSNP_af =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"dbSNP AF"};
                                        my $kg_af =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"KG AF"};
                                        my $kg_afr_af =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"KG AFR AF"};
                                        my $kg_amr_af =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"KG AMR AF"};
                                        my $kg_asn_af =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"KG ASN AF"};
                                        my $kg_eur_af =
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"KG EUR AF"};
```

```
						my $esp_af = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"ESP AF"};
						my $esp_aa_af = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"ESP AA AF"};
						my $esp_ea_af = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"ESP EA AF"};

						my $accession_id = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC 
accession ID"};
						my $primary_site = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC 
primary site"};
						my $site_subtype = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC site 
subtype"};
						my $genome_coords = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC 
coords"};
						my $cosmic_cds = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC 
CDS"};
						my $cosmic_aa = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC AA"};
						my $cosmic_decription = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC 
description"};
						my $fathmm_prediction = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC 
FATHMM prediction"};
						my $somatic_status = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC 
somatic status"};
						my $pubmed_id = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC 
pubmed ID"};
						my $study_id = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC study 
ID"};
						my $sample_source = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC 
sample source"};
						my $tumour_origin = 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$tissue}{$coin}{"COSMIC 
tumour origin"};

						print OUT 
"$run,$tissue,$coin,$gene_name,$transcript,$chr,$pos,$rsID,$nucleotide_change,$amino_acid,
$sample_count,$ref_depth,$var_depth,$snpeff_effect,$flanking_sequence,$filter,$mq,$mq_filter
,$qd,$qd_filter,$fs,$fs_filter,$haplotype_score,$haplotype_score_filter,$mqrs,$mqrs_filter,$read
_pos_rank,$read_pos_rank_filter";

						#Is the variant in this sample 
found in the same sample from another run?
						for my $other_run (sort {"\L$a" 
cmp "\L$b"} @runs){
							if (exists 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$other_run} && exists 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$other_run}{$tissue} && exists 
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$other_run}{$tissue}{$coin}){
```

```
                                                                                                    print OUT
",Y";
                                                                                            }

                                                                                            else{print OUT ",N";}

                                                                                            #Use these lines of
code if you want to look at variants in a patient that are found across all runs and tissues for that
patient.
                                                                                            for my $other_tissue
(sort {"\L$a" cmp "\L$b"} keys %tissue_presence){
                                                                                                    if (exists
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$other_run} && exists
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$other_run}{$other_tissue} && exists
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$other_run}{$other_tissue}{$coin}){

        $tissue_presence{$other_tissue} = "Y";
                                                                                                    }
                                                                                            }
                                                                                    }

                                                                                    for my $other_tissue (sort
{"\L$a" cmp "\L$b"} keys %tissue_presence_overall){
                                                                                            #use this line if you're
looking at variants in other tissues for this sample only.
                                                                                            # if (exists
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$other_tissue}){$tissue_presen
ce{$other_tissue} = "Y";}

                                                                                            print OUT
",$tissue_presence_overall{$other_tissue}";
                                                                                    }

                                                                                    for my $other_tissue (sort
{"\L$a" cmp "\L$b"} keys %tissue_presence){
                                                                                            #use this line if you're
looking at variants in other tissues for this sample only.
                                                                                            # if (exists
$variants{$chr}{$pos}{$nucleotide_change}{$amino_acid}{$run}{$other_tissue}){$tissue_presen
ce{$other_tissue} = "Y";}

                                                                                            print OUT
",$tissue_presence{$other_tissue}";
                                                                                    }

                                                                                    print OUT
",$dbSNP_presence,$kg_presence,$esp_presence,$cosmic_presence,$dbSNP_af,$kg_af,$kg_
afr_af,$kg_amr_af,$kg_asn_af,$kg_eur_af,$esp_af,$esp_aa_af,$esp_ea_af,$accession_id,$pri
mary_site,$site_subtype,$genome_coords,$cosmic_cds,$cosmic_aa,$cosmic_description,$fath
mm_prediction,$somatic_status,$pubmed_id,$study_id,$sample_source,$tumour_origin\n";
                                                                            }
                                                                    }
                                                            }
                                                    }
                                            }
                                    }
                            }

        close OUT;
}
```

```
############################################################################
#####################
#End of subroutine printer.
############################################################################
#####################
```

Custom Script 3

```
#DOCTOR - Depth Of Coverage In Translated Only Regions.
#Calculate the percent of each ORF of each gene of interest (GOI) at 4x, 10x, 20x 30x, 40x
coverage.
#Also calculate the average coverage for each gene of interest. Two major steps, including:
#Diagnosis - gathering information about coding regions from a given dataset (in this case
NCBI's CCDS)
#and genes of interest from a specified list. A line of code is used to look only at these GOIs or
every
#gene in the dataset (the 'exome').

#Treatment - Using SAMtools mpileup to get the coverage at each nucleotide in the CDS for
every GOI.
#Search the output for the coverage at only the coding bases and calculate the coverage across
the gene
#(divide by the number of coding bases). Also works out the coverage across the exome. The
printing subroutines
#are called from this subroutine, which initialise the output file (&admit) and print the output
(&discharge).

##############################################################################
################

#use this script to test writing an excel file.
use Spreadsheet::WriteExcel;

#prints things in pretty colours.
use Term::ANSIColor;

#Tells perl we're going to be using R.
use Statistics::R;

($date, $time) = &delorean;

#name of the output table (note the full filepath is given later).
$doc_table = "contamination profile ($date)";

@depths = (1, 10, 20, 50, 100, 200, 300, 400, 500, 1000, 2000, 2500, 3000);

#used to convert the BGIx10 samples to their COIN IDs.
#%bgi_exomes = (
	#x => "110926_I174_FCC0343ACXX_L3_HUMvejXAAAAAPEI-9", x =>
#"110930_I810_FCC02KFACXX_L6_HUMvejXABAAAPEI-12",
	#x => "110926_I174_FCC0343ACXX_L3_HUMvejXACAAAPEI-1", x =>
#"110926_I174_FCC0343ACXX_L3_HUMvejXADAAAPEI-2",
	#x => "110930_I810_FCC0335ACXX_L5_HUMvejXAEAAAPEI-3", x =>
#"110930_I810_FCC02KFACXX_L6_HUMvejXAFAAAPEI-4",
	#x => "110930_I810_FCC02KFACXX_L6_HUMvejXAGAAAPEI-5", x =>
#"110930_I810_FCC0335ACXX_L5_HUMvejXAHAAAPEI-7",
	#x => "110930_I810_FCC0335ACXX_L5_HUMvejXAIAAAPEI-8", x =>
#"110926_I174_FCC0343ACXX_L4_HUMvejXAJAAAPEI-9"
#);

%rev_bgi = reverse %bgi_exomes;

#load sample names into an array.
```

```
#look up all files (realigned.bam files) in specified directories and store filepaths for each in an
array (will contain files from each sequencing set).
$unc = "/media/marc/Triceratops/Exomes/UNC output/09_GATK_indelrealigner";
$bgix10 = "/media/marc/Triceratops/Exomes/BGIx10 output/09_GATK_indelrealigner";
$bgix13 = "/media/marc/Triceratops/Exomes/BGIx13 output/09_GATK_indelrealigner";
$bgix16 = "/media/marc/Triceratops/Exomes/BGIx16 output/09_GATK_indelrealigner";
$bgix18 = "/media/marc/Triceratops/Exomes/BGIx18 output/09_GATK_indelrealigner";
$nhs = "/media/marc/Triceratops/Exomes/Pilot project output/NHS
output/09_GATK_indelrealigner/isolated amplicons";
$nhs_repeat = "/media/marc/Triceratops/Exomes/Pilot project output/NHS repeat
output/09_GATK_indelrealigner";
$nhs_trimmed = "/media/marc/Triceratops/Exomes/Pilot project output/NHS trimmed
output/09_GATK_indelrealigner";
$wthd = "/media/marc/Triceratops/Exomes/Pilot project output/WT (HD)
output/09_GATK_indelrealigner/isolated amplicons";
$wtld = "/media/marc/Triceratops/Exomes/Pilot project output/WT (LD)
output/09_GATK_indelrealigner";
$nhs_raw= "/media/marc/Brachiosaur/NGS data/Pilot project (KRAS, NRAS, BRAF &
PIK3CA)/NHS Miseq/Misc MiSeq generated files";
$nr = "/media/marc/Triceratops/Exomes/Non responder output/09_GATK_indelrealigner/alien
amplicons";
$ingram = "/media/marc/Triceratops/Exomes/Ingram 14-5546 output/09_GATK_indelrealigner";
$contamination = "/media/marc/Triceratops/Exomes/Contamination
output/09_GATK_indelrealigner";

$amplicon_file = "/home/marc/Documents/work/reseq analysis/pilot
project/Miscellaneous/DesignStudio amplicons.csv";
$panel_file = "/home/marc/Documents/work/reseq analysis/non responder
project/Miscellaneous/NR & NHS panel genes.csv";

@sample_sets = ($contamination);

foreach my $data_dir (sort {"\L$a" cmp "\L$b"} @sample_sets){
	#Read in the 'realigned.bam' file names for each sample, and store them in the
@data_wave array.
	opendir DATA, "$data_dir" or die "$data_dir not found. Did you mount the drive, you
tool?";

	#@data_wave is a temporary collection of files in the current directory (containing the
relaigned.bams for the centre).
	#could just push these straight into @data_pool but this method takes virtually no time
anyway and is only performed once.
	my @data_wave = map {$data_dir."/".$_} grep { -f "$data_dir/$_" &&
/(isolated_amplicons_sorted|realigned)\.bam$/i} readdir DATA;

	#for each sample, create the path to its BAM file and store this in a new array,
@data_pool.
	foreach my $file (sort {"\L$a" cmp "\L$b"} @data_wave){
		my $sample = $file;

		$sample =~ s/.+\///g;

		my @sample = split (/-/, $sample);

		my ($run, $coin, $tissue) = @sample;

		$tissue =~ s/\_.+$//;

		#skip x, it is a re-sequenced copy of x.
		# next if $run =~ /[^6]run2/i;
```

```
			# next if $coin =~ /Undetermined/;
			# next unless $sample =~ /x\-e/i;

			#@data_pool will contain the full filepath.
			push (@{$data_pool{$run}{$coin}{$tissue}}, $file);
			@{$data_pool{$run}{$coin}{$tissue}} = sort {"\L$a" cmp "\L$b"}
@{$data_pool{$run}{$coin}{$tissue}};
		}

		closedir DATA;
}

# $targets = panel_reader($panel_file);

#Call the sub routines.
$gene_coordinates = diagnosis($targets);
# $gene_coordinates = second_diagnosis($gene_coordinates, $amplicon_file);

if (%gene_coordinates){$plot = treatment($gene_coordinates, \%rev_bgi, \%data_pool,
$doc_table, \@depths);}

print "No of bases for resequencing: $seq_target.\n";

# plot($plot);


sub panel_reader{
		my $panel_file = shift;

		my %targets;

		open PANEL, "$panel_file" or die $!;

		while (my $line = <PANEL>){
			chomp $line;

			next if $line =~ /^#/;

			my @fields = split (/,/, $line);

			my ($gene, $panel) = @fields;

			unless (grep {$panel eq $_} sort @{$targets{$gene}}){push
@{$targets{$gene}}, $panel;}

			@{$targets{$gene}} = sort {"\L$a" cmp "\L$b"} @{$targets{$gene}};print
"@{$targets{$gene}}\n";
		}

		close PANEL;

		return (\%targets);
}

################################################################################
################################################################################
#########
#diagnosis
#step 1: find exon regions for each gene. open gene list, store gene names in an array
```

```
#	open up the CCDS file and store genes of interest (GOIs) in an array. the exon starts
and ends can be split into a hash.
#	if a GOI isn't in the CCDS, then the GOI will not be used any further.
################################################################################
################################################################################
#########
sub diagnosis{
	my $targets = shift;

	my %targets = %{$targets};

	print "Diagnosis\n";

	#the files containing the GOIs and the CCDS dataset.
	my $gene_list = "/home/marc/Documents/work/exome analysis/Miscellaneous/team
requests/Hannah/coverage_cmt_20x_2013-6-20.csv";
	my $ncbi_ccds = "/media/marc/Brachiosaur/Exomes/database
resources/CCDS_2013_6_16.txt";
	my $ref_seq = "/media/marc/Brachiosaur/Exomes/database
resources/refseq_genes_GRCh37_2013-06-20";

	#Open them.
	open GENE_LIST, $gene_list or die "The gene_list file $gene_list is not in the specified
location";
	open CCDS, $ncbi_ccds or die "The NCBI file $ncbi_ccds is not in the specified
location";
	open REFSEQ, $ref_seq or die "The RefSeq file $ref_seq is not in the specified
location";

	#read the gene list line by line. split each line into an array, use the first element to find
the gene name.
	#store each gene name into an array.
	while ($gene_line = <GENE_LIST>){
		#ignore the first line.
		next if ($gene_line =~ /^Gene Name/i);

		#remove new line characters from end of line.
		chomp $gene_line;

		#split the line into an array, by commas.
		my @gene_line = split (/\,/, $gene_line);

		my $gene = $gene_line[0];
		$gene =~ s/\'//g;

		#converts the gene name to upper case.
		#gene names in CCDS = all caps (and start with a letter).
		$gene =~ tr/a-z/A-Z/;

		#last if $gene eq "";
		#last if $gene eq "";

		#store the GOIs in an array.
		push (@GOI, $gene);
	}

	close GENE_LIST;

	#use this if you want to look at coverage in specified genes.
```

```
        @GOI = qw(AKT1      AKT2    AKT3    ARAF    AREG    AXL     BCL2L11         BRAF
        CBL     CDH1    DOK2    EGF     EGFR    ERBB2   ERBB3   EREG    FCGR2A
        FCGR3A          FLT1    GAB1    GRB2    HRAS    IGF1    IGF1R   JAK1    JAK2    KDR
        KRAS    MAP2K1          MAP2K2          MAPK1           MAPK3           MET     NCK1
        NRAS    PDK1    PIK3CA          PLCG1   PLCG2   PRKCA           PRKCD           PTEN
        RAF1    SHC1    SOS1    SRC     STAT1   STAT3   TP53    DDR2    KIT);
        # @GOI = qw(KRAS NRAS BRAF PIK3CA);

        #Sort the GOI array alphabetically.
        @GOI = sort {"\L$a" cmp "\L$b"} @GOI;
        # @chr = qw(6 22 X);

        #read in from the CCDS file to extract exon regions for the genes of interest. Some
genes have several transcripts.
        #Some of these transcripts share the same name but have different coding regions on
the chromosome.
        #all transcripts have a unique transcript ID.
        while (my $ccds_line = <CCDS>){
                #reset the number of exons for the gene/transcript. Necessary for hash
creation.
                my $no_exons = 0;
                my $gene_length = 0;
                my $no_bases = 0;
                my $holding_gene = 0;
                my %holding_bases = ();

                # my $gene_panel;

                #ignore the first line
                next if $ccds_line =~ /^\#chromosome/i;

                #remove any new line characters from the line in the CCDS file.
                chomp $ccds_line;

                #split the CCDS line into an array (by spaces).
                @ccds_line = split (/\s/, $ccds_line);

                #the number of elements in the CCDS line array.
                $ccds_line_size = scalar (@ccds_line);

                #Assign scalars to CCDS line array elements.
                my $chr = $ccds_line[0];
                my $gene = $ccds_line[2];
                my $transcript_id = $ccds_line[4];
                my $ccds_status = $ccds_line[5];

                #translate the gene name to upper case.
                $gene =~ tr/a-z/A-Z/;

                # if (grep {$gene eq $_} sort {"\L$a" cmp "\L$b"} keys %targets){$gene_panel =
${$targets{$gene}}[0];}

                #store vital information from the CCDS line (chr/gene/transcript name).
                my $chr_gene = $chr."\_".$gene;
                my $gene_transcript = $gene."\_".$transcript_id;
                my $chr_gene_transcript = $chr."\_".$gene_transcript;

                if (@{$holding_lengths{$chr_gene}}){$holding_gene = 1;}
                #ignore CCDS statuses if they are not public (eg. are withdrawn, pending
withdrawal, etc).
                #non public genes typically have low coverage or the CCDS line contains
insufficient data to calculate coverage for the transcript.
                next unless $ccds_status eq "Public";

                #number of genes in the CCDS
                $total_transcripts++;

                #skips the line if the gene is not a GOI. Ensures only GOIs are added to the
GOI hash.
                #silence this line to include all genes in CCDS.
                # next unless (grep {$gene eq $_} sort @GOI);
                # next unless (grep {$chr eq $_} sort @chr);

                #count the number of GOI transcripts in the CCDS.
                $goi++;

                #this checks the current transcript against the previous line, to see whether or
not it is a duplicate
                $is_duplicate = $chr_gene_transcript;
                $is_duplicate =~ s/\_[a-z\d\.]+$//i;
                if ($is_duplicate =~ /^$chr\_$gene$/){$duplicate_transcripts++;}
                $unique_transcripts = $goi - $duplicate_transcripts;

                #store genes in the CCDS file in an array, to see how many genes (not
transcripts) there are in CCDS.
                #unless (grep {$gene eq $_} sort @genes){push (@genes, $gene);}

                #create a hash of hash of arrays for all genes: %gene_coordinates =
{%chr_gene_transcripts => (@exon_no = (start pos, stop pos))}
                #for first exon in line to last exon in line, where $i is the (exon number + 8).
                for $i (9 .. ($ccds_line_size-2)){
                    #remove the [], characters from the exon positions.
                    $ccds_line[$i] =~ s/[\[\]\,]//g;

                    #split the exon into start and stop positions.
                    @exon_start_end = split (/\-/, $ccds_line[$i]);

                    #push the start and stop positions into the hash, if the positions are
numerical (some may have errors in, rendering the CCDS line useless).
                    if ($exon_start_end[0] =~ /^\d+$/ && $exon_start_end[1] =~ /^\d+$/){
                        #foreach (coding) nucleotide in the exon.
                        for $base ($exon_start_end[0] .. $exon_start_end[1]){
                            #store each base in an array in a hash.
                            push @{$holding_bases{$chr_gene}}, $base;
                        }

                        #the gene length is the total number of coding bases.
                        #Note the +1 at the end is essential, or the length will be (1 *
number of exons) too short.
                        $gene_length += ($exon_start_end[1] - $exon_start_end[0]) +
1;

                        #count how many exons there are in the line - need to know
this.
                        $no_exons++;
                    }
                }
                #if the gene has many transcripts, use this to store details for the transcript with
the most coding bases.
                #the first time a gene is encountered, store the details. If another transcripts is
encountered, replace the prev. details with the new ones.
                if ($holding_gene == 1){
                    if ($gene_length > ${$holding_lengths{$chr_gene}}[1]){
                        #this stores the lengths of the transcripts.
                        @{$holding_lengths{$chr_gene}} = ($chr_gene, $gene_length);

                        @{$gene_coordinates{$chr}{$chr_gene}{"bases"}} =
@{$holding_bases{$chr_gene}};
                        ${$gene_coordinates{$chr}{$chr_gene}{"transcript"}} =
$ccds_line[4];
                        ${$gene_coordinates{$chr}{$chr_gene}{"gene_length"}} =
$gene_length;
                        ${$gene_coordinates{$chr}{$chr_gene}{"no_exons"}} =
$no_exons;
                        $no_bases = scalar
@{$gene_coordinates{$chr}{$chr_gene}{"bases"}};
                        $cds_end = ${$holding_bases{$chr_gene}}[$no_bases - 1];
                        $cds_start = ${$holding_bases{$chr_gene}}[0];
                        push @{$gene_coordinates{$chr}{$chr_gene}{"CDS"}},
$cds_start, $cds_end;

                        $seq_target += $gene_length;
                    }
                }

                #the first time a gene is encountered
                else{
                    #keep a record of every unique gene in CCDS.
                    push @genes, $gene;

                    #this stores the lengths of the transcripts.
                    @{$holding_lengths{$chr_gene}} = ($chr_gene, $gene_length);

                    #store details of the gene.
                    @{$gene_coordinates{$chr}{$chr_gene}{"bases"}} =
@{$holding_bases{$chr_gene}};
                    ${$gene_coordinates{$chr}{$chr_gene}{"transcript"}} = $ccds_line[4];
                    ${$gene_coordinates{$chr}{$chr_gene}{"gene_length"}} =
$gene_length;
                    ${$gene_coordinates{$chr}{$chr_gene}{"no_exons"}} = $no_exons;
                    $no_bases = scalar @{$gene_coordinates{$chr}{$chr_gene}{"bases"}};
                    $cds_end = ${$holding_bases{$chr_gene}}[$no_bases - 1];
                    $cds_start = ${$holding_bases{$chr_gene}}[0];
                    push @{$gene_coordinates{$chr}{$chr_gene}{"CDS"}}, $cds_start,
$cds_end;

                    #print BED "$chr

                    $seq_target += $gene_length;

                    #special cirumstances only: keep a record of genes in CCDS so that
they can be ignored in the refseq file.
                    #essentially stored the same data as @genes (plus chr data) but in a
way more efficient for searching.
                    #push @{$refseq_check{$chr}}, $gene;
                }
```

```
	}

	close CCDS;

	#sort the genes array.
	@genes = sort (@genes);

	#Check the number of GOI that aren't in the database file.
	print "Checking whether GOIs are in the CCDS database\n";
	foreach my $gene_of_interest (@GOI){
		unless (grep {$gene_of_interest eq $_} sort @genes){
			#special circumstances only: used by refseq to look up GOIs that
appear in refseq but not in CCDS.
			#stores GOIs that do not appear in CCDS.
			push (@ref_seek, $gene_of_interest);

			$absent++;
			print "$absent\t$gene_of_interest is not in the CCDS file\n";}
	}

	print "reading in missing genes from RefSeq.\n";
	#read in from the RefSeq file to get exon boundaries for genes which weren't in the
CCDS.
	while ($refseq_line = <REFSEQ>){
		#reset the number of exons for the gene/transcript. Necessary for hash
creation.
		my $no_exons = 0;
		my $gene_length = 0;
		my $no_bases = 0;
		my $holding_gene = 0;
		my %holding_bases = ();

		# my $gene_panel;

		#ignore the first line
		next if $refseq_line =~ /^\#bin/i;

		#remove any new line characters from the line in the CCDS file.
		chomp $refseq_line;

		#split the CCDS line into an array (by spaces).
		@refseq_line = split (/\t/, $refseq_line);

		#the number of elements in the CCDS line array.
		$refseq_line_size = scalar (@refseq_line);

		#Assign scalars to CCDS line array elements.
		$chr = $refseq_line[2];
		$chr =~ s/^chr//i;
		$gene = $refseq_line[12];
		$nm_accession = $refseq_line[1];
		$nm_accession =~ s/\_/\-/;

		#only looks at chr 1 - 22, X & Y.
		next if $chr =~ /\_/;

		#translate the gene name to upper case.
		$gene =~ tr/a-z/A-Z/;
```

```
			# if (grep {$gene eq $_} sort {"\L$a" cmp "\L$b"} keys %targets){$gene_panel =
${$targets{$gene}}[0];}

			#store vital information from the refseq line (chr/gene/transcript name).
			$chr_gene = $chr."\_".$gene;
			$gene_accession = $gene."\_".$nm_accession;
			$chr_gene_transcript = $chr."\_".$gene_accession;

			if (@{$holding_lengths{$chr_gene}}){$holding_gene = 1;}

			#number of genes in the CCDS
			$total_accessions++;

			#silence this line to include all genes in refseq, otherwise will look at only GOIs.
			# next unless (grep {$gene eq $_} sort @GOI);
			# next unless (grep {$chr eq $_} sort @chr);

			#skips genes which have already been added by the CCDS. can ignore unless
adding to %WET in CCDS.
			next if (grep {$chr_gene eq $_} sort {"\L$a" cmp "\L$b"} keys
%{$gene_coordinates{$chr}});

			#count the number of GOI transcripts in RefSeq.
			$goi++;

			@exon_starts = split (/\,/, $refseq_line[9]);
			@exon_ends = split (/\,/, $refseq_line[10]);

			#store base numbers for each exon.
			for $exon_no (0 .. ((scalar @exon_ends) - 1)){
				$exon_start = $exon_starts[$exon_no];
				$exon_end = $exon_ends[$exon_no];

				for $base ($exon_start .. $exon_end){push
@{$holding_bases{$chr_gene}}, $base;}

				$gene_length += ($exon_end - $exon_start) + 1;

				$no_exons++;
			}

			if ($holding_gene == 1){
				if ($gene_length > ${$holding_lengths{$chr_gene}}[1]){
					#since the transcript in the refseq line is the longest transcript
for this gene, update %holding lengths and %WET.
					@{$holding_lengths{$chr_gene}} = ($chr_gene, $gene_length);

					@{$gene_coordinates{$chr}{$chr_gene}{"bases"}} =
@{$holding_bases{$chr_gene}};
					${$gene_coordinates{$chr}{$chr_gene}{"transcript"}} =
$nm_accession;
					${$gene_coordinates{$chr}{$chr_gene}{"gene_length"}} =
$gene_length;
					${$gene_coordinates{$chr}{$chr_gene}{"no_exons"}} =
$no_exons;
					$no_bases = scalar
@{$gene_coordinates{$chr}{$chr_gene}{"bases"}};
					$cds_end = ${$holding_bases{$chr_gene}}[$no_bases - 1];
					$cds_start = ${$holding_bases{$chr_gene}}[0];
					push @{$gene_coordinates{$chr}{$chr_gene}{"CDS"}},
$cds_start, $cds_end;

					$seq_target += $gene_length;
				}
			}

			else{
				#since the transcript in the refseq line is the longest transcript for this
gene, update %holding lengths and %WET.
				@{$holding_lengths{$chr_gene}} = ($chr_gene, $gene_length);

				@{$gene_coordinates{$chr}{$chr_gene}{"bases"}} =
@{$holding_bases{$chr_gene}};
				${$gene_coordinates{$chr}{$chr_gene}{"transcript"}} = $nm_accession;
				${$gene_coordinates{$chr}{$chr_gene}{"gene_length"}} =
$gene_length;
				${$gene_coordinates{$chr}{$chr_gene}{"no_exons"}} = $no_exons;
				$no_bases = scalar @{$gene_coordinates{$chr}{$chr_gene}{"bases"}};
				$cds_end = ${$holding_bases{$chr_gene}}[$no_bases - 1];
				$cds_start = ${$holding_bases{$chr_gene}}[0];
				push @{$gene_coordinates{$chr}{$chr_gene}{"CDS"}}, $cds_start,
$cds_end;

				push @refseq_genes, $gene;

				$seq_target += $gene_length;
			}
		}

		close REFSEQ;

		foreach my $gene_of_interest (@ref_seek){
			unless (grep {$gene_of_interest eq $_} sort @refseq_genes){
				$still_absent++;
				print "$still_absent\t$gene_of_interest is not in the CCDS or RefSeq
datasets.\n";}
		}


		#special circumstances: looks only at refseq genes.
		# for $chr (sort {"\L$a" cmp "\L$b"} keys %refseq_check)
		# {
			# foreach $gene (sort @{$refseq_check{$chr}})
			# {
				# $chr_gene = $chr."\_".$gene;
				# delete ($gene_coordinates{$chr}{$chr_gene});
			# }
		# }

		#count how many genes/transcripts of interest there are.
		$goi = scalar (@GOI) - $absent;
		$no_genes = scalar (@genes);
		for $chr (sort {"\L$a" cmp "\L$b"} keys %gene_coordinates){
			$toi += scalar (keys %{$gene_coordinates{$chr}});
		}

		#print "There are $ref_genes\:$transcript_count refseq genes:transcripts\n";
```

```
	print "There are $goi GOIs ($toi TOIs) present in CCDS, $total_transcripts total
(\"public\") transcripts ($unique_transcripts unique and $duplicate_transcripts duplicate
transcripts)\n\n$seq_target\n\n";

	return \%gene_coordinates;
}

##########################################################################
##########################################################################
#########
#End of Diagnosis
##########################################################################
##########################################################################
#########

sub second_diagnosis{
	my ($gene_coordinates, $amplicon_file) = @_;

	my %gene_coordinates = %{$gene_coordinates};

	open AMP, "$amplicon_file" or die $!;

	while (my $line = <AMP>){
		chomp $line;

		next if $line =~ /^#/;

		my @bases;

		my @fields = split (/,/, $line);

		my $gene = $fields[0];
		my $amplicon = $fields[1];
		my $chr = $fields[2];
		my $start_coord = $fields[3];
		my $end_coord = $fields[4];

		my $chr_gene_amp = $chr."\_".$gene."\_".$amplicon;

		for (my $pos = $start_coord; $pos <= $end_coord; $pos++){
			push @{$gene_coordinates{$chr}{$chr_gene_amp}{"bases"}}, $pos;
		}

		${$gene_coordinates{$chr}{$chr_gene_amp}{"transcript"}} = $amplicon;
		${$gene_coordinates{$chr}{$chr_gene_amp}{"gene_length"}} = scalar
@{$gene_coordinates{$chr}{$chr_gene_amp}{"bases"}};
		@{$gene_coordinates{$chr}{$chr_gene_amp}{"CDS"}} = ($start_coord,
$end_coord);
	}

	return \%gene_coordinates;
}
##########################################################################
##########################################################################
#########
#treatment
#step 2:		Calculates the coverage for each transcript in %WET for each sample.
#		Printing output is performed by a further 2 called subroutines; admit &
discharge.
```

```
##############################################################################
#####s########################################################################
##########
sub treatment{
	my ($gene_coordinates, $rev_bgi, $data_pool, $doc_table, $depths) = @_;
	my %gene_coordinates = %{$gene_coordinates};
	my %rev_bgi = %{$rev_bgi};
	my %data_pool = %{$data_pool};
	my @depths = @$depths;

	my $sample_count;

	#used when printing - starts at -1 not 0. It's complicated but works perfectly.
	my ($row, $column) = (0, 0);

	print "Starting Treatment subroutine\n";

	#initialise the excel file. $workbook is returned by 'admit' and caught by the code here
so can be used downstream.
	my $workbook = admit(\%gene_coordinates, \%rev_bgi, \%data_pool, $doc_table,
\@depths);

	#foreach sample, calculate the coverage stats for each gene and print to the wookbook
the stats, gene by gene.
	#calls &discharge, which does the printing.
	for my $run (sort {"\L$a" cmp "\L$b"} keys %data_pool){
		for my $coin (sort {"\L$a" cmp "\L$b"} keys %{$data_pool{$run}}){
			for my $tissue (sort {"\L$a" cmp "\L$b"} keys
%{$data_pool{$run}{$coin}}){
				foreach my $filepath (sort {"\L$a" cmp "\L$b"}
@{$data_pool{$run}{$coin}{$tissue}}){
					#count the number of samples covered.
					$sample_count++;

					#set the necessary variables.
					$row = 1;
					$column += 1;
					$sample_print_count = 1;
					$gene_count = 0;
					$col_counter = 0;
					$no_genes = 0;

					my $index_file = $filepath;
					$index_file =~ s/bam$/bai/i;

					unless (-f "$index_file"){`samtools index "$filepath"`;}

					if (grep {$coin eq $_} sort {"\L$a" cmp "\L$b"} keys
%rev_bgi){$coin = $rev_bgi{$coin};}

					#loop over each gene in the hash of hash of arrays
					for my $chr (sort {"\L$a" cmp "\L$b"} keys
%gene_coordinates){
						#running count of the number of transcripts
analysed.
						$no_genes += scalar keys
%{$gene_coordinates{$chr}};

						#foreach transcript.
				for my $genes (sort {"\L$a" cmp "\L$b"} keys
%{$gene_coordinates{$chr}}){
					#count the number of genes in the
hash, and reset other variables if necessary.
					$gene_count++;
					$row++;
					$gene_coverage = 0;
					$base = 0;

					foreach my $depth (sort {$a <=> $b}
@depths){
						my $depth_var = "x".$depth;
						$$depth_var = 0;
					}

					%gene_bases = ();

					#get the chromosome number/name
					if ($genes =~ /\_[^\_]+$/i){$gene =
$&;$gene =~ s/\_//g;}

					#get some stuff about the CDS.
					$cds_start =
${$gene_coordinates{$chr}{$genes}{"CDS"}}[0];
					$cds_end =
${$gene_coordinates{$chr}{$genes}{"CDS"}}[1];
					$gene_length =
${$gene_coordinates{$chr}{$genes}{"gene_length"}};#length = number of bases in exons
					$no_exons =
${$gene_coordinates{$chr}{$genes}{"no_exons"}};

					#call samtools mpileup specifying the
whole gene region and the sample's realigned BAM file as input.
					#From the STDOUT, cut the position
and number of reads covering the position and save it to an array.
					#For more info on switches, google
them.
					(@coverage) = `samtools mpileup -A -
E -Q 0 -r $chr\:$cds_start\-$cds_end "$filepath" | cut -f2,4`;

					# next if scalar @coverage == 0;
					#loop over the @coverage array and
store the coverage at each base in a hash.
					foreach my $site (@coverage) {
						#split the array element into a
temporary array containing the position and the coverage at the pos.
						@pileup = split (/\s/, $site);
						$base = $pileup[0];
						$base_coverage = $pileup[1];

						#store the coverage at each
nucleotide in the gene into a hash.
						#memory usage is minimal as
this hash is emptied for every gene.
						${$gene_bases{$base}} =
$base_coverage;
					}

					#loop over the coding bases in the
gene and look up the coverage from the %gene_bases hash created above.
```

```
#count the number of bases with a
coverage > each depth threshold.
foreach my $base (sort {$a <=> $b}
@{$gene_coordinates{$chr}{$genes}{"bases"}}){
        #looks up the coverage at the
coding base.
        $base_coverage =
${$gene_bases{$base}};

        #keep a count of the total
coverage across the whole gene.
        #this will be averaged later.
        $gene_coverage +=
$base_coverage;

        #counts the number of bases
with coverage > thresholds.
        foreach my $depth (sort {$a
<=> $b} @depths){
                if ($base_coverage >=
$depth){
                        my
$depth_var = "x".$depth;
        $$depth_var++;

                }
        }
}

#Get the average gene coverage by
dividing gene coverage by the number of coding bases.
$gene_coverage /= $gene_length;

#turn the counts of bases with
coverage > thresholds into percent of the ORF.
foreach my $depth (sort {$a <=> $b}
@depths){
        my $depth_var = "x".$depth;
        $$depth_var = ($$depth_var /
$gene_length) * 100;

        $admissions{$depth} =
$$depth_var;
}

#keep a running count of the total
coverage for every gene in the exome,
#and average it by dividing by the
number of genes in the exome.
${$exome_coverage{$exome}} +=
$gene_coverage;
$exome_average =
${$exome_coverage{$exome}}/$gene_count;

#print some helpful stuff so you can
see what's occuring.
print "\nSample ";
print color ("blue"), "$sample_count";
print color ("red"), ": $coin\t";
print color 'reset';
```

```
                                                                      print "Transcript ";
                                                                      print color ("green"),
"$gene_count/$toi";
                                                                      print color 'reset';
                                                                      print ": ";
                                                                      print color ("blue"), "$genes\n";
                                                                      print color 'reset';
                                                                      print "CDS: $cds_start -
$cds_end\tGene length: ".$gene_length."bp\tExons: $no_exons
${$gene_coordinates{$chr}{$genes}{$no_exons - 1}}[0]\nAverage gene coverage:
$gene_coverage\nAverage exome coverage so far: $exome_average\nORF % at";

                                                                      #call discharge from here, passing a
reference to the workbook.
                                                                      my $workbook =
discharge(${$workbook}, \%admissions, $row, $column);

                                                                      #Plot bar charts to show data per
sample (or group of samples) extra-graphically. Not at all necessary for the xls file creation but
is a nice little add on.
                                                                      foreach my $depth (sort {$a <=> $b}
@depths){
                                                                              my $depth_key = $depth."x";
                                                                              my $depth_var = "x".$depth;

                                                                              print "\t$depth_key:
$$depth_var";

                                                                              $$depth_var = ($$depth_var /
$gene_length) * 100;

                                                                              $plot{$tissue}{"ORF
depths"}{$depth_key}{$genes} = $$depth_var;
                                                                      }

                                                                      print "\n\n";

                                                                      #sample group can be a group of
samples/patients (eg.  germline blood, non-responders, etc) or a single sample.
                                                                      #genes is '$chr\_$gene'.
                                                                      $plot{$tissue}{"average
coverage"}{$genes} = $gene_coverage;
                                                              }
                                                      }

                                                      #the average coverage for the exome
                                                      ${$exome_coverage{$exome}} /= $gene_count;
                                              }
                                      }
                              }
                      }

                      #set row and column for printing exome averages.
                      $row += 2;
                      $column = 0;

                      #write exome averages in a row at the bottom of the .xls file.
                      foreach my $sample (sort {"\L$a" cmp "\L$b"} @samples){
                              #leave a space so that averages are written underneath the average columns.
                              $column += 1;
```

```
		#print averages for every exome to every worksheet.
		foreach $worksheet ($workbook -> sheets()){
			$sheet_name = $worksheet -> get_name();
			if ($sheet_name eq "averages"){$worksheet -> write ($row, $column, 
${$exome_coverage{$sample}});}
		}
	}

	print "Treated $sample_count samples, each with $all_genes_genes genes 
($all_genes_trans transcripts)\n";

	return \%plot;
}
################################################################################
################################################################################
#########
#End of treatment
################################################################################
################################################################################
#########

################################################################################
################################################################################
#########
#discharge
#print the coverage stats to the output table.
################################################################################
################################################################################
#########
sub discharge{
	#take the reference from the treatment routine by using shift on the array of args.
	#(which isn't specified by a line of code but is how perl works).
	my ($workbook, $admissions, $row, $column) = @_;
	my %admissions = %{$admissions};

	#foreach worksheet in the workbook.
	foreach my $worksheet ($workbook -> sheets()){
		#inc. counter for printing to spreadsheet.
		$col_counter++;

		#get the name of the sheet using Spreadsheet::writeExcel::get_name.
		$sheet_name = $worksheet -> get_name();

		#print out blank cells for this gene in this sample as there is no coverage
		#(eg. the gene lies on chr Y and the patient is female).
		unless ($sheet_name eq "averages"){
			unless ($gene_coverage){
				for $col ($column .. ($column)){
					$worksheet -> write_blank ($row, $col);
				}
			}
		}

		#get the ORF % covered at $array_index fold coverage. the average coverage 
for the ORF is the i=5 index when you measure 4x, 10, 20x, 30x and 40x.
		#array index is used to get the number in the sheet name, which is useful as 
each sheet contains coverage at a given threshold.
		if ($sheet_name =~ /^\d+/){
			#stat is used to point to the ORF percent covered at any depth
threshold (the threshold being the sheet name with an 'x' in front).
			$stat = "x$&";

			#to avoid going over the limit of cell styles in excel, cut the remainders
of each stat so you have an integer.
			#(stat therefore being an integer percent between 0 - 100).
			$rgb = $$stat;
			if ($rgb =~ /\.\d+$/){$rgb =~ s/$&//;}

			#calculate each of the R and G values used in the colour format.
			#if the percent is higher than 50, more green should be used and fade
to yellow
			if ($rgb >= 50){
				#this equation increases the amount of red the lower the
percent gets
				#i.e. as % goes down, red goes up.
				#eg. at 100%, r = ((1 - 1) * 255) * 2 = 0
				#	50%, r = ((1 - 0.5) * 255) * 2 = 255.
				$r = ((1-($rgb/100))*255) * 2;
				$g = 255;
				$b = 0;
			}

			#if percent is less than 50, fade from yellow to red.
			else{
				#this equation decreases the amount of green the lower the
percentage gets
				#i.e. as % goes down, so does green.
				$g = (($rgb/100)*255) * 2;
				$r = 255;
				$b = 0;
			}

			#Excel uses a palette of 56 colours, ranging from 8 - 63. What I do here
is I replace each colour in the palette with a custom colour (based on the RGB values of the
ORF percent).
			#Since percents range from 0 - 100, I must scale them to fit into the 56
colour window.
			#This is done by dividing the percent by 100/56 (~1.786, any more d.p.
and it won't work quite right) and then adding 8.
			#e.g. % = 0, colour = 0 / 1.786 + 8 = 8. % = 100, colour = 100 / 1.786 +
8 = 63. % = 50, colour = 50 / 1.786 + 8 = 36.
			$colour = ($rgb / 1.786) + 8;
			$colour =~ s/\.\d+$//;

			#to stay within the cell style limit, store each of the custom formats in a
hash.
			#each colour, 8 - 63 is then associated with a format, giving a
maximum of 56 cell styles.
			if (grep {$colour eq $_} sort {"\L$a" cmp "\L$b"} keys %formats){
				$format = ${$formats{$colour}};
			}

			else{
				#if the colour isn't associated with a format yet, make it so.
				#shade is the RGB settings of the colour. Each There are a
maximum of 100 possible shades using this % system
				#which is well below excel's cell style limit yet allows for a good
looking gradient.
```

```
                                        $shade = $workbook -> set_custom_color ($colour, $r, $g, $b);

                                        #format of the cell.
                                        $format = $workbook -> add_format(bg_color => $shade);

                                        #store the format in a hash, will store up to 100 formats (as
intended).
                                        ${$formats{$colour}} = $format;
                                        push @{$gradient{$colour}}, $r, $g, $rgb;
                              }

                              #write the percent of the ORF covered, using the value-specific
formatting
                              $worksheet -> write ($row, $column, $$stat, $format);

                    }

                    #print out all the average coverages on the final sheet (named 'averages').
                    else{$worksheet -> write ($row, $column, $gene_coverage);}
          }

          #return the workbook to treatment sub.
          return \$workbook;
}
##############################################################################
##############################################################################
#########
#End of Discharge
##############################################################################
##############################################################################
#########

##############################################################################
##############################################################################
#########
#admit
#initialise the .xls output file; print samples across the top and gene/transcript names along the
side.
##############################################################################
##############################################################################
#########
sub admit{
          my ($gene_coordinates, $rev_bgi, $data_pool, $doc_table, $depths) = @_;
          my %gene_coordinates = %{$gene_coordinates};
          my %data_pool = %{$data_pool};
          my %rev_bgi = %{$rev_bgi};
          my @depths = @$depths;

          #initialise the .xls file.
          my $workbook = Spreadsheet::WriteExcel-
>new('/home/marc/Documents/work/DOCTOR/ORF coverage/'.$doc_table.'.xls');

          #add worksheets for each of the coverage thresholds.
          foreach my $depth (sort {$a <=> $b} @depths){
                    my $sheet = $depth."x";
                    my $depth = "x".$depth;

                    $$depth = $workbook -> add_worksheet($sheet);
          }
```

```
	$averages = $workbook -> add_worksheet('averages');

	#foreach sheet in the workbook.
	foreach my $worksheet ($workbook -> sheets()){
		#reset to row 1 (will be incremented before printing data).
		my $row = 1;

		#this is the column to print the sample name to, and the column to print the
column headers (eg. ORF% and seq depth) to.
		my $sample_print_count = 0;
		my $header_column = 0;

		#get the sheet name.
		my $sheet_name = $worksheet -> get_name();

		#print column/row titles in the top left corner cell and the cell below it.
		$worksheet->write(0, 0, 'Sample >');
		$worksheet->write(1, 0, 'Gene');

		my $orf_header = "ORF \% at $sheet_name";
		my $coverage_header = "Mean ORF depth";

		#loop over the samples
		for my $run (sort {"\L$a" cmp "\L$b"} keys %data_pool){
			for my $coin (sort {"\L$a" cmp "\L$b"} keys %{$data_pool{$run}}){
				for my $tissue (sort {"\L$a" cmp "\L$b"} keys
%{$data_pool{$run}{$coin}}){
					foreach my $filepath (sort {"\L$a" cmp "\L$b"}
@{$data_pool{$run}{$coin}{$tissue}}){
						#count the number of samples printed.
						#Effectively used as a column count when
printing sample names.
						$sample_print_count++;

						if (grep {$coin eq $_} sort {"\L$a" cmp "\L$b"}
keys %rev_bgi){$coin = $rev_bgi{$coin};}

						#write the sample names to the column heads.
						$worksheet -> write (0, $sample_print_count,
"$run-$coin-$tissue");

						#increment sample count again - leaves a
column for printing averages across the ORF.
						#$sample_print_count++;

						#prints out the headers for each column
(percent of ORF covered at Yx and average coverage across the ORF).
						if ($sheet_name =~ /\d+x/i){
							#starts at 0, but increases twice (2
elements in @column_headers) for every sample looped over.
							#So below every sample name is
'ORF% at Yx Average coverage across ORF'.
							$header_column++;
							$worksheet ->
write(1,$header_column,$orf_header);
						}

						else{
							$header_column++;
```

```
                                                        $worksheet ->
write(1,$header_column,$coverage_header);
                                                }
                                        }
                                }
                        }
                }

                #print out the genes down the first (0th) column.
                for my $chr (sort {"\L$a" cmp "\L$b"} keys %gene_coordinates){
                        for my $gene (sort {"\L$a" cmp "\L$b"} keys
%{$gene_coordinates{$chr}}){
                                my $print_transcript =
$gene."\_".${$gene_coordinates{$chr}{$gene}{"transcript"}};

                                $row++;
                                $worksheet -> write($row, 0, $print_transcript);
                        }
                }
        }

        print "Ending 'admit' subroutine\n";

        #returns the workbook to treatment subroutine so it can be used by the downstream
script.
        return \$workbook;
}
##########################################################################
##########################################################################
#########
#End of Admit
##########################################################################
##########################################################################
#########

sub plot{
        my $plot = shift;
        my %plot = %{$plot};

        my $r_tissue = "/home/marc/Documents/work/DOCTOR/Coverage plots/pilot project
test 2014-06-02.png";

        my (@samples, @sample_averages);

        #Used so you know which tissue code is which tissue type. Not essential but nice to see
the actual type printed.
        my %tissue = (
                a => "tFFPE",
                d => "nFFPE",
                e => "Germline",
                f => "WGA tFFPE",
                Undetermined => "Undetermined"
        );

        #Set $R as a new Statistics::R object and start the R session.
        my $R = Statistics::R -> new();
        $R -> start_sharedR;

        print "Loading ggplot2.\n";
        $R -> run(q`library(ggplot2)`);
```

```
	for my $tissue (sort {"\L$a" cmp "\L$b"} keys %plot){
		my $sample_count = scalar keys %plot;
		my $sample_coverage = 0;
		my $gene_count = 0;
		my ($colour, @genes, @gene_averages);
		my $tissue_code = $tissue{$tissue};

		my $r_gene = "/home/marc/Documents/work/DOCTOR/Coverage
plots/Coverage by genes (pilot project test 2014-06-02).png";

		if ($tissue_code eq "tFFPE"){$colour = "red";}
		if ($tissue_code eq "nFFPE"){$colour = "darkred";}
		if ($tissue_code eq "Germline"){$colour = "green";}
		if ($tissue_code eq "WGA tFFPE"){$colour = "yellow";}

		@colour_genes = ($colour);

		#Loop over the genes sequenced for this sample and sum the average
coverages.
		for my $chr_gene (sort {"\L$a" cmp "\L$b"} keys %{$plot{$tissue}{"average
coverage"}}){
			my $gene = $chr_gene;
			$gene =~ s/^.+\_//g;

			$gene_count++;

			#The averge coverage for this gene in this sample.
			my $gene_coverage = $plot{$tissue}{"average coverage"}{$chr_gene};

			#Running total coverage across all genes for this sample.
			$sample_coverage += $gene_coverage;

			push @genes,  $chr_gene;
			push @gene_averages, $gene_coverage;
		}

		$R -> run(qq`png("$r_gene")`);

		$R -> set('genes', \@genes);
		$R -> set('depths', \@gene_averages);

		$R -> set('plot_colours', \@colour_genes);

		#Plot the data by gene.
		$R -> run(q`#put the data into a dataframe, sorted by gene name.
			genes_df <- data.frame(Genes = genes, Depths = depths)
			genes_df <- genes_df[order(genes_df$Genes),]

			#create an object of the plot. Reordering the data will let you plot by
ascending/descending frequency.
			depth_plot <- ggplot(
				genes_df,
				aes(
					x=Genes,
					y=Depths
				)

			#It will be a barplot.
			) + geom_bar(
                        fill=plot_colours,
                        colour="black"

                #Give the plot a title.
                ) + labs(
                        title = "Average depth for each gene"

                #labels the axis.
                ) + xlab("Gene"

                #Rearrange the axis text.
                ) + opts(
                        axis.text.x = theme_text(
                                angle = 45,
                                hjust = 1
                        )

                ) + ylim(0, 5000)


                #Plot the graph.
                depth_plot`);

        #This is necessary to complete the plot.
        $R -> run (q`dev.off()`);

        #Calculate the sample's average coverage by dividing by the number of
samples.
        my $sample_average = $sample_coverage / $gene_count;

        $stats{$tissue_code}{"average"} = $sample_average;
        $stats{$tissue_code}{"colour"} = $colour;

        # push @samples, $tissue_code;
        # push @sample_averages, $sample_average;
    }

    for my $tissue (sort {"\L$a" cmp "\L$b"} keys %stats){
        push @samples, $tissue;
        push @sample_averages, $stats{$tissue}{"average"};
        push @colour_tissues, $stats{$tissue}{"colour"};
    }

    print "Creating and plotting graph now\n";
    #makes the output file.
    $R -> run(qq`png("$r_tissue")`);

    $R -> set('samples', \@samples);
    $R -> set('depths', \@sample_averages);
    $R -> set('plot_colours', \@colour_tissues);

    #Plot the coverage data by tissue type.
    #set the genes and frequency data as a dataframe object.
    $R -> run(q`#put the data into a dataframe, sorted by gene name.
        tissues_df <- data.frame(Samples = samples, Depths = depths)
        tissues_df <- tissues_df[order(tissues_df$Samples),]

        #create an object of the plot. Reordering the data will let you plot by
ascending/descending frequency.
        depth_plot <- ggplot(
				tissues_df,
				aes(
					x=Samples,
					y=Depths
				)

			#It will be a barplot.
			) + geom_bar(
				fill=plot_colours,
				colour="black"

			#Give the plot a title.
			) + labs(
				title = "Average depth in each tissue type"

			#labels the axis.r
			) + xlab("Tissue type"

			#Rearrange the axis text.
			) + opts(
				axis.text.x = theme_text(
					angle = 45,
					hjust = 1
				)
			)

			#Plot the graph.
			depth_plot`);



		#This is necessary to complete the plot.
		$R -> run (q`dev.off()`);

		#Stop R.
		$R -> stopR();
}

##########################################################################
#subroutine delorean
#Gets the local time and converts it into a date (format: yyyy-mm-dd) and time (hh:mm:ss).
#These are then returned,
##########################################################################
sub delorean{
		my @flux_capacitor = localtime(time);

		#Set the variables for the time and date. Note: year is a count of years after 1900 (e.g.
2015 = 115), and month is 0-based integer (0 = Jan, 11 = Dec).
		my ($sec, $min, $hour, $date, $month, $year) = @flux_capacitor;

		$month += 1;
		$year += 1900;

		$month = sprintf ("%02d", $month);
		$date = sprintf ("%02d", $date);
		$day = sprintf ("%02d", $day);
		$sec = sprintf ("%02d", $sec);
		$min = sprintf ("%02d", $min);
		$hour = sprintf ("%02d", $hour);
```

```
	my $return_date = "$year-$month-$date";

	my $return_time = "$hour:$min:$sec";

	return ($return_date, $return_time);
}
#############################################################################
#End of subroutine delorean.
#############################################################################
```

Custom Script 4

```
#Compare variants in my pipeline with variants from the University of North Carolina (UNC) and
Beijing Genomics Institute (BGI) pipelines.
#The percentage of SNPs and InDels between mine and their pipelines are then calculated.
#This is all done by reading through each of my variant call files (VCFs) and loading each
variant into a hash. Their respective VCFs can then be read and variants stored in hashes.
#All the variants are ultimately stored in a hash of hashes of hashes of arrays, as illustrated
below:
#%pipeline = (%sample = (%variant_type => {@chromosome_chrPosition = (ref allele, alt
allele)}))
#created by pushing ref and alt alleles into
@{$pipeline{$sample}{$variant_type}{$chromosome_chrPosition}}

#Hash of BGI x10 exomes, where sample name (key) is associated with sequencing reads
name (value)
%bgi_exomes = (          x => "110926_I174_FCC0343ACXX_L3_HUMvejXAAAAAPEI-9", x =>
"110930_I810_FCC02KFACXX_L6_HUMvejXABAAAPEI-12",
                         x => "110926_I174_FCC0343ACXX_L3_HUMvejXACAAAPEI-1", x =>
"110926_I174_FCC0343ACXX_L3_HUMvejXADAAAPEI-2",
                         x => "110930_I810_FCC0335ACXX_L5_HUMvejXAEAAAPEI-3", x =>
"110930_I810_FCC02KFACXX_L6_HUMvejXAFAAAPEI-4",
                         x => "110930_I810_FCC02KFACXX_L6_HUMvejXAGAAAPEI-5", x =>
"110930_I810_FCC0335ACXX_L5_HUMvejXAHAAAPEI-7",
                         x => "110930_I810_FCC0335ACXX_L5_HUMvejXAIAAAPEI-8", x =>
"110926_I174_FCC0343ACXX_L4_HUMvejXAJAAAPEI-9");

#Use the following code for comparing only certain genes.
$genes_file = "/home/marc/Documents/work/exome analysis/Miscellaneous/genes 1609.csv";

open GENES, $genes_file or die "gene file $genes_file cannot be opened.\n";

while ($line = <GENES>)
{
	next if $line =~ /^Gene name/i;

	@lines = split (/,/, $line);
	$gene_name = $line[0];

	push (@genes, $gene_name);
}

$ccds_file = "/media/Brachiosaur/Exomes/database resources/CCDS.current.txt";

open CCDS, $ccds_file or die "ccds file $ccds_file cannot be opened.\n";
while ($line = <CCDS>)
{
	next if $line =~ /^\#/i;

	@lines = split (/\s/, $line);

	$chromosome = $line[0];
	$gene_name = $line[2];
	$coordinates = $line[7]."\_".$line[8];

	$gene_coordinates{$chromosome}{$coordinates} = $gene_name;
}
```

```
reader();
concordance();
printer();


##########################################################################
##########################################################################
############
#sub routine to read in all data to hash
##########################################################################
##########################################################################
############
sub reader{
	print "Now reading in variants from variant files.\n";
	#hash containing filepaths to all variant directories.
	%vaDirs = (	BGIx10 => "/media/Brachiosaur/Exomes/sequencing 
data/BGI/HKC11050",
			BGIx13 => "/media/Brachiosaur/Exomes/sequencing data/BGI 
x13/HK11788/CleanData",
			BGIx18 => "/media/Brachiosaur/Exomes/sequencing data/BGI 
x18/HK11582",
			UNC => "/media/Brachiosaur/Exomes/sequencing data/Chapel Hill 
Data/2010");

	@variant_types = ("snp", "indel");

	for $centre (sort keys %vaDirs)
	{
		print "Centre: $centre.\n";

		opendir VaDir, $vaDirs{$centre} or die "$centre not found. Did you mount the 
drive?\n";

		if ($centre eq "BGIx10")
		{
			@bgix10_samples = grep {-d "$vaDirs{$centre}/$_" && !/^\.+/} readdir 
VaDir;
			foreach (@bgix10_samples)
			{
				$sample = $_;

				$sample_no++;

				foreach (@variant_types)
				{
					print "Sample $sample_no: $sample\tVariants: $_\n";

					$total_csv = 0;

					$variant_type = $_;

					opendir CSVdir, 
"$vaDirs{$centre}/$sample/result_variation/$variant_type" or die 
"$vaDirs{$centre}/$sample/result_variation/$variant_type could not be opened\n";
					@var_files = grep {/\.+csv$/} readdir CSVdir;

					foreach $csv_file (@var_files)
					{
```

```
                        open CSV,
"$vaDirs{$centre}/$sample/result_variation/$variant_type/$csv_file" or die
"$vaDirs{$centre}/$sample/result_variation/$variant_type/$csv_file could not be opened\n";
                        while ($csv_line = <CSV>)
                        {
                            #increase total variants for each line
                            $total_csv++;

                            #ignore header line
                            next if ($csv_line =~ /^Func/i);

                            #some lines in these CSV files contain
commas within quotation marks, so replace these to ensure they are not used to split the line
later.
                            @csv_line = split(//, $csv_line);

                            $new_line = "";

                            foreach (@csv_line)
                            {
                                if ($_ eq "\""){$inbetweener
++;}
                                if($_ eq "\," && $inbetweener
% 2 == 1){$_ =~ s/\,/\;/;}
                                $new_line .= $_;
                            }

                            @csv_line = split(/\,/, $new_line);

                            #remove chr from chromosome field
and quotation marks from fields.
                            $csv_line[15] =~ s/chr//i;
                            $csv_line[1] =~ s/\"//g;
                            $csv_line[2] =~ s/\"//g;
                            $csv_line[20] =~ s/\"//g;
                            $csv_line[21] =~ s/\"//g;
                            $csv_line[23] =~ s/\"//g;
                            $csv_line[24] =~ s/\"//g;
                            $csv_line[27] =~ s/\"//g;

                            #create a 'hash key' and alternate
allele 'value' which you can use to check against the hash of your SNPs.
                            #the key is the chromosome number
and chr. position.
                            #do not use $csv_line[16] for pos for
deletions, as the pos will be +1 base away from the corresponding position given by my VCF
output.
                            #The ref and variant bases from the
CSV file. Push them into the hash of hashes of hashes of hashes of arrays.
                            if ($variant_type eq "snp")
                            {
                                $csv_chr_pos =
$csv_line[15]."\_".$csv_line[16];

                                $csv_AC = $csv_line[20];

                                #sets the genotype for the
SNP called by BGI
                                #AC=0 won't ever be used for
BGI SNP data.
```

```
                    if ($csv_AC =~ /\=0/)
                    {
                        $csv_ref =
$csv_line[18];
                        $csv_alt =
$csv_line[18];
                    }

                    if ($csv_AC eq "het")
                    {
                        $csv_ref =
$csv_line[18];
                        $csv_alt =
$csv_line[19];
                    }

                    if ($csv_AC eq "hom")
                    {
                        $csv_ref =
$csv_line[19];
                        $csv_alt =
$csv_line[19];
                    }

                    $csv_ref = $csv_line[18];
                    $csv_alt = $csv_line[19];

                    if ($csv_line[21] =~
/dbSNP\=\d\;/i){$their_rsID = $csv_line[9].$&;}
                }

                else #the variant is an indel
                {
                    $csv_chr_pos =
$csv_line[15]."\_".$csv_line[21];

                    $csv_AC = $csv_line[27];
                    $csv_AC =~ s/\;.+$//;

                    #set the genotype for the indel
detected by BGI.
                    #AC=0 should never be
encountered in the BGI indel data

                    if ($csv_AC =~ /\=0/)
                    {
                        $csv_ref =
$csv_line[23];
                        $csv_alt =
$csv_line[23];
                    }

                    if ($csv_AC =~ /\=1/)
                    {
                        $csv_ref =
$csv_line[23];
                        $csv_alt =
$csv_line[24];
                    }

                    if ($csv_AC =~ /\=2/)
```

```
								{
									$csv_ref =
$csv_line[24];
									$csv_alt =
$csv_line[24];
								}

								$csv_ref = $csv_line[23];
								$csv_alt = $csv_line[24];

								$their_rsID = $csv_line[9];
							}

							$gene_name = $csv_line[1];
							$their_var_effect = $csv_line[2];

							#use this line to only look at particular
genes (specified at top of script).
							#next unless grep {$gene_name eq
$_} @genes;

							#use this line to only look at novel
variants (i.e. have no rsID).
							#next if $their_rsID =~ /^rs\d+/i;

							push
(@{$their_calls{$centre}{$sample}{$variant_type}{$csv_chr_pos}}, $csv_ref, $csv_alt,
$their_rsID, $their_var_effect);

						}#end of BGIx10 CSV file
					}
				}#end of variant types

				print "\n";

			}#end of each bgix10 samples
		}#end of bgix10

		if ($centre eq "UNC")
		{
			foreach (@variant_types)
			{
				#adds an 's' to the end of the variant, to match the directory
path in the Chapel Hill folders.
				$variant_type = $_;
				$variant_types = $variant_type."s";

				opendir UNC_samples, "$vaDirs{$centre}/$variant_types
unzipped" or die "$vaDirs{$centre}/$variant_types unzipped cannot be found. Is the drive
mounted?\n";

				#read in files from snps/indels unzipped, use raw.vcf files as
indels don't have filtered.vcf files (only difference is 'pass' column anyway).
				@unc_sample_read = grep {-f
"$vaDirs{$centre}/$variant_types unzipped/$_" && /raw\.vcf$/i} readdir UNC_samples;

				#adds full filepath to each sample name and adds to
@unc_samples.
				foreach $sample (sort @unc_sample_read){$sample =
"$vaDirs{$centre}/$variant_types unzipped/$sample";push (@unc_samples, $sample);}
```

```
                              }

                              foreach (@unc_samples)
                              {
                                        #get the type of variant (SNP/InDel) for the sample.
                                        $variant_type = $_;
                                        $variant_type =~ s/s.{1}unzipped.+//i;
                                        $variant_type =~ s/.+\///g;

                                        #the sample ID, just the number (eg. 042031_02136_s_6).
                                        $sample = $_;
                                        $sample =~ s/.+\///g;
                                        $sample =~ s/\..+//;

                                        #counts how many samples there are from North Carolina.
                                        if ($variant_type eq "snp"){$sample_no++;}

                                        print "Sample $sample_no: $sample\tVariants:
$variant_type\n";

                                        #opens the VCF file to read in variants.
                                        open VCF, "$_" or die "$_ file could not be opened\n";

                                        while ($vcf_line = <VCF>)
                                        {
                                                  #ignores header lines.
                                                  next if $vcf_line =~ /^\#/;

                                                  @vcf_line = split (/\s/, $vcf_line);
                                                  $line_size = scalar @vcf_line;

                                                  $vcf_chr_pos = $vcf_line[0]."\_".$vcf_line[1];

                                                  #checks whether the variant is hom ref, het or hom
mut. since UNC have been awkward and made it so
                                                  #their indel VCF files do not contain the normal AC=\d
format, use the GT 0/1 at the end of the line to get the allele count.
                                                  if ($vcf_line[$line_size-1] =~ /^\d\/\d/)
                                                  {
                                                            $vcf_AC = $&;

                                                            @ac = split (/\//, $vcf_AC);

                                                            #set the genotype for the variant detected by
UNC.
                                                            #if variant is hom ref, ref and alt alleles are
both the reference allele.
                                                            if ($ac[0] == 0)
                                                            {
                                                                      $vcf_ref = $vcf_line[4];
                                                                      $vcf_alt = $vcf_line[4];
                                                            }

                                                            #if variant is het, ref and alt alleles are ref and
alt.
                                                            elsif ($ac[0] == 1 && $ac[1] == 1)
                                                            {
                                                                      $vcf_ref = $vcf_line[3];
                                                                      $vcf_alt = $vcf_line[4];
                                                            }
```

```
					#if variant is hom mut, ref and alt alleles are
both the alternate allele.
					else
					{
						$vcf_ref = $vcf_line[3];
						$vcf_alt = $vcf_line[3];
					}
				}

				$vcf_ref = $vcf_line[3];
				$vcf_alt = $vcf_line[4];

				$their_rsID = $vcf_line[2];

				#used for getting the gene name from the CCDS file.
Not needed unless looking only at particular genes.
				# for $coordinates (sort keys
%{$gene_coordinates{$vcf_line[0]}})
				# {
					# @coordinates = split (/\_/, $coordinates);
					# if ($vcf_line[1] >= $coordinates[0] &
$vcf_line[1] <= $coordinates[1])
					# {
						# $gene_name =
${$gene_coordinates{$vcf_line[0]}{$coordinates}};
					# }
				# }

				#use this line to only look at particular genes (specified
at top of script).
				#next unless grep {$gene_name eq $_} @genes;

				#use this line to only look at novel variants (i.e. have no
rsID).
				#next if $their_rsID =~ /^rs\d+/i;

				#push the reference and mutated alleles into the hash
of hashes of hashes of hashes of arrays.
				push
(@{$their_calls{$centre}{$sample}{$variant_type}{$vcf_chr_pos}}, $vcf_ref, $vcf_alt,
$their_rsID);
			}#end of file loop.

			#$variant_type .= "s";

		}#end of samples.
	}#end of UNC.

	#count the total number of samples stored in the hashes.
	$total_samples += scalar (keys %{$their_calls{$centre}});

}#end of all sequencing centres

%yoDirs = (	BGIx10 => "/media/Triceratops/Exomes/BGI
output/19_GATK_variantannotator",
		BGIx13 => "/media/Triceratops/Exomes/BGIx13
output/19_GATK_variantannotator",
		BGIx18 => "/media/Triceratops/Exomes/BGIx18
output/19_GATK_variantannotator",
                              UNC => "/media/Triceratops/Exomes/UNC
output/19_GATK_variantannotator");

          for $my_dir (keys %yoDirs)
          {
                    opendir YoDirs, $yoDirs{$my_dir} or die "$yoDirs{$my_dir} cannot be found, did
you mount the drive?\n";

                    @my_files = grep {-f "$yoDirs{$my_dir}/$_" && /variants\_annotated\.vcf$/}
readdir YoDirs;

                    foreach $my_file (@my_files)
                    {
                              $my_sample_no++;

                              #open VCF must be before changes to $my_file.
                              open VCF, "$yoDirs{$my_dir}/$my_file" or die
"$yoDirs{$my_dir}/$my_file cannot be opened, did you mount the drive?";

                              $my_file =~ s/\_variants\_annotated\.vcf$//;

                              #if you have a bgix10 sample, change the $my_file value into the
sample ID fom the barcode.
                              %rev_bgi = reverse %bgi_exomes;
                              if (grep {$my_file eq $_} keys %rev_bgi){$my_file = $rev_bgi{$my_file};}

                              print "Gathering data on my sample $my_sample_no: $my_file\n";

                              while ($my_line = <VCF>)
                              {
                                        #ignore header lines.
                                        next if $my_line =~ /^\#/;

                                        @my_line = split (/\s/, $my_line);

                                        #gets a key for the hash (chromosome_position) and reference
and alternate alleles (as well as length of these alleles, so we know if its a SNP or InDel).
                                        $my_chr_pos = $my_line[0]."\_".$my_line[1];

                                        #checks whether the variant is hom ref, het or hom mut.
                                        $my_AC = $my_line[7];
                                        $my_AC =~ s/\;.+$//;

                                        #set the genotype for the variant I called.
                                        #if variant is hom ref, ref and alt alleles are both the reference
allele.
                                        if ($my_AC =~ /\=0/)
                                        {
                                                  $my_ref = $my_line[3]; $my_ref_length = length
$my_ref;
                                                  $my_alt = $my_line[3]; $my_alt_length = length
$my_alt;
                                        }

                                        #if variant is het, ref and alt alleles are ref and alt.
                                        if ($my_AC =~ /\=1/)
                                        {
                                                  $my_ref = $my_line[3]; $my_ref_length = length
$my_ref;
```

```
					$my_alt = $my_line[4]; $my_alt_length = length 
$my_alt;
				}

				#if variant is hom mut, ref and alt alleles are both the alternate 
allele.
				if ($my_AC =~ /\=2/)
				{
					$my_ref = $my_line[4]; $my_ref_length = length 
$my_ref;
					$my_alt = $my_line[4]; $my_alt_length = length 
$my_alt;
				}

				$my_ref = $my_line[3]; $my_ref_length = length $my_ref;
				$my_alt = $my_line[4]; $my_alt_length = length $my_alt;

				$my_rsID = $my_line[2];


				#get gene name from line (not needed unless only comparing 
certain genes).
				if ($my_line[7] =~ /SNPEFF_GENE_NAME=[A-Z0-9\_\-
\.]+\;/i){$gene_name = $&;}
				$gene_name =~ s/SNPEFF_GENE_NAME\=//i;
				$gene_name =~ s/\;//i;

				#use this line to only look at particular genes (specified at top 
of script).
				#next unless grep {$gene_name eq $_} @genes;

				#use this line to only look at novel variants (i.e. have no rsID).
				#next if $my_rsID =~ /^rs\d+/i;

				if ($my_line =~ /snpeff\_effect\=[a-z0-9\_]+\;/i)
				{
					$my_var_effect = $&;
					$my_var_effect =~ s/snpeff\_effect\=//i;
					$my_var_effect =~ s/\;//i;
				}

				#checks whether the variant is a SNP or indel, and pushes it 
into a hash of hashes of hashes of arrays.
				#Also pushes in the rsID and effect the variant has (e.g. 
synonymous, frameshift, stopgain, etc.).
				if ($my_ref_length < 2 && $my_alt_length < 2)
				{
					$my_variant_type = "snp";
					push 
(@{$my_calls{$my_file}{$my_variant_type}{$my_chr_pos}}, $my_ref, $my_alt, $my_rsID, 
$my_var_effect);
				}
				else #variant is an indel
				{
					$my_variant_type = "indel";
					push 
(@{$my_calls{$my_file}{$my_variant_type}{$my_chr_pos}}, $my_ref, $my_alt, $my_rsID, 
$my_var_effect);
				}
			}
```

```
					close VCF;
				}

				closedir YoDirs;
			}

			#calculate the total number of samples analysed (actually counts the number of
sequencing read files input).
			$total_samples += scalar (keys %my_calls);
			print "Total samples: $total_samples\n\n";

}
###########################################################################
###########################################################################
############
#end of reader subroutine
###########################################################################
###########################################################################
############

###########################################################################
###########################################################################
############
#sub routine to check concordance between datasets
#loop over your hash, count totals for all samples and gather totals of the effects of each
variant.
#Do the same for their hash, and compare each of their samples with yours to calculate
concordance.
###########################################################################
###########################################################################
############
sub concordance
{
			%sequencing_centres = (BGIx10 => "Beijing Genomics Institute", UNC => "University
of North Carolina");

			print "Calculating concordance between Marc's pipeline and the sequencing centres.\n";

			#count of how many samples have been analysed.
			$my_sample_no = 0;
			$their_sample_no = 0;

			#my hash layout (hash of hashes of hashes of arrays / hash of hashes of arrays.
Inception eat your heart out).
			# %my_calls = { %sample 1 .. n	= (%snp/indel		= (@chr_pos		= [ref
allele, alt allele, rsID, effect type]))
			#				= (%my_effect_types	= (@effect of variant	=
[count]))
			#				= (@sample n_stats	= [my_total_variants,
my_novel_total, my_novel_snp, my_total_snp, my_total_indel, my_novel_indel])
			#		}

			#loop over the samples in Marc's hash.
			for $sample (sort keys %my_calls)
			{
				#ignore the my_effect_types hash, only look at the samples.
				next if $sample =~ /my\_effect\_types/;

				#create name of sample stats array
```

```
        $sample_stats = $sample."\_stats";

        #for the 2 variant kinds, SNP and InDel.
        for $variant_type (sort keys %{$my_calls{$sample}})
        {
            #look only at the snp and indel hashes.
            next if $variant_type eq "$sample\_stats";

            #count how many SNPs/InDels/total variants I have for this sample (ie.
counts the number of arrays [the chr-pos combos] for each variant types, but not the array
values)
            ${$my_calls{$sample}{$sample_stats}}[0]+= scalar (keys
%{$my_calls{$sample}{$variant_type}}); #$my_total_variants = ...

            #count how many SNPs i found for this sample.
            if ($variant_type eq "snp")
            {
                ${$my_calls{$sample}{$sample_stats}}[2] = scalar (keys
%{$my_calls{$sample}{$variant_type}}); #$my_total_snp = ...
            }

            #count how many indels I found for this sample.
            else
            {
                ${$my_calls{$sample}{$sample_stats}}[4] = scalar (keys
%{$my_calls{$sample}{$variant_type}}); #$my_total_indel = ...
            }

            #for each variant (ie. every SNP or indel in the sample).
            for $variant (sort keys %{$my_calls{$sample}{$variant_type}})
            {
                #if the variant doesn't have an rsID (ie. isn't in the version of
dbSNP I used for the pipeline).
                unless (${$my_calls{$sample}{$variant_type}{$variant}}[2] =~
m/rs\d+/i)
                {
                    #count how many novel variants I have.
                    ${$my_calls{$sample}{$sample_stats}}[1]++;
#$my_novel++;

                    #count how many novel SNPs I have.
                    if ($variant_type eq
"snp"){${$my_calls{$sample}{$sample_stats}}[3]++;} #$my_novel_snp++;

                    #count how many novel InDels I have.
                    else {${$my_calls{$sample}{$sample_stats}}[5]++;}
#$my_novel_indel++;
                }

                #Find out what kind of variant it is ('SNPEFF_EFFECT='
annotations) in my data,
                $my_effect_type =
${$my_calls{$sample}{$variant_type}{$variant}}[3];

                #Count how many times this kind of variant appears in my
data.
                #each sample could have only a range of effect types, so some
samples wont contain effect types seen in others.
                #therefore storing my effect types in the $my_calls{$sample}
hash means all types are acounted for.
```

```
	${$my_calls{$sample}{'my_effect_types'}{$my_effect_type}}[0]++;
				}
			}



			#push my effect types into an array, which can be looped over later for file
printing.
			for $my_effect_type (sort keys %{$my_calls{$sample}{'my_effect_types'}})
			{
				unless (grep {$my_effect_type eq $_} @all_effect_types)
				{
					push (@all_effect_types, $my_effect_type);
				}
			}
		}

		#their hash layout (hash of hashes of hashes of hashes of arrays / hash of hashes of
hashes of arrays. Inception eat your heart out!!).
		# %their_calls = {%seq centre = (%sample 1 .. n = (%snp/indel		= (@chr_pos
			= [ref allele, alt allele, rsID, effect type]))
		#							= (%their_effect_types	= (@effect of
variant	= [count]))
		#							= (@sample n_stats	=
[their_total_variants, their_total_snp, their_total_indel, their_novel_total, their_novel_snp,
their_novel_indel])
		#							= (@sample concordance		=
[percent of mine they found, % snps of mine they found, % indels of mine they found, percent of
theirs I found, % snps of theirs I found, % indels of theirs I found])
		#					)}

		#loop over the sequencing centres in their hash (BGIx10, BGIx18, UNC).
		for $centre (sort keys %their_calls)
		{
			print "Calculating concordance between $centre and exomeExpress.pl.\n";

			#loop over each sample they analysed.
			for $sample (sort keys %{$their_calls{$centre}})
			{
				$total_match = 0;
				$snp_match = 0;
				$indel_match = 0;
				$novel_in_theirs_match_total = 0;
				$novel_in_theirs_match_snp = 0;
				$novel_in_theirs_match_indel = 0;
				$novel_in_mine_match_total = 0;
				$novel_in_mine_match_snp = 0;
				$novel_in_mine_match_indel = 0;

				#create the name of the sample stats/concordance array.
				$sample_stats = $sample."\_stats";
				$sample_concordance = $sample."\_concordance";

				#count how many samples they have analysed.
				$their_sample_no++;

				print "Sample $sample\n";

				#for every variant (ie. every single SNP and InDel).
				for $variant_type (sort keys %{$their_calls{$centre}{$sample}})
				{
					#ignore the effect types hash, only look at samples.
					next if $sample =~ /their_effect_types/;

					#look only at the snp and indel hashes.
					next if ($variant_type eq "$sample\_stats" || $variant_type eq
"$sample\_concordance");

					#count how many SNPs/InDels/total variants I have for this
sample (ie. counts the number of arrays [the chr-pos combos] for each variant types, but not the
array values)
					${$their_calls{$centre}{$sample}{$sample_stats}}[0]+= scalar
(keys %{$their_calls{$centre}{$sample}{$variant_type}}); #$their_total_variants = ...

					#count how many SNPs/InDels they have for this sample (ie.
counts the number of arrays for each variant types, but not the array values)
					if ($variant_type eq "snp")
					{
						${$their_calls{$centre}{$sample}{$sample_stats}}[2] =
scalar (keys %{$their_calls{$centre}{$sample}{$variant_type}}); #$their_total_snp = ...
					}

					else
					{
						${$their_calls{$centre}{$sample}{$sample_stats}}[4] =
scalar (keys %{$their_calls{$centre}{$sample}{$variant_type}}); #$their_total_indel = ...
					}

					#Loop over their variants for matching. It doesn't matter
whether you loop over their variants or Marc's, you'll be making a direct comparison to the other
group so will effectively
					#be looking at both anyway (i.e. all matches will be found).
					for $variant (sort keys
%{$their_calls{$centre}{$sample}{$variant_type}})
					{
						#counts how many variants are shared between their
data and Marc's. Also counts how many of these are SNPs/InDels.
						$my_variant =
${$my_calls{$sample}{$variant_type}{$variant}}[0]."\_".${$my_calls{$sample}{$variant_type}{$v
ariant}}[1];
						$their_variant =
${$their_calls{$centre}{$sample}{$variant_type}{$variant}}[0]."\_".${$their_calls{$centre}{$sampl
e}{$variant_type}{$variant}}[1];
						if ($my_variant eq $their_variant)
						{
							# $sample_variant_type_count =
$sample.$variant_type;
							# $$sample_variant_type_count++;
							# print "matching $centre $sample
$variant_type $$sample_variant_type_count\n$my_variant\n$their_variant\n";

							$total_match++;
							unless
(${$my_calls{$sample}{$variant_type}{$variant}}[2] =~
/^rs\d+/i){$novel_in_mine_match_total++;}
							unless
(${$their_calls{$centre}{$sample}{$variant_type}{$variant}}[2] =~ /dbSNP\=1/i ||
${$their_calls{$centre}{$sample}{$variant_type}{$variant}}[2] =~ /rs\d+/i)
							{
										$novel_in_theirs_match_total++;
										print "$centre $sample variant:
$their_variant ($my_variant)\nrsID:
${$their_calls{$centre}{$sample}{$variant_type}{$variant}}[2]\tmy rsID:
${$my_calls{$sample}{$variant_type}{$variant}}[2]\n\n";

									}

									if ($variant_type eq "snp")
									{
										$snp_match++;
										unless
(${$my_calls{$sample}{$variant_type}{$variant}}[2] =~
/^rs\d+/i){$novel_in_mine_match_snp++;}
										unless
(${$their_calls{$centre}{$sample}{$variant_type}{$variant}}[2] =~ /dbSNP\=1/i ||
${$their_calls{$centre}{$sample}{$variant_type}{$variant}}[2] =~ /rs\d+/i)
										{

	$novel_in_theirs_match_snp++;
										}
									}

									else
									{
										$indel_match++;
										unless
(${$my_calls{$sample}{$variant_type}{$variant}}[2] =~
/^rs\d+/i){$novel_in_mine_match_indel++;}
										unless
(${$their_calls{$centre}{$sample}{$variant_type}{$variant}}[2] =~
/^rs\d+/i){$novel_in_theirs_match_indel++;}
									}
								}

								unless
(${$their_calls{$centre}{$sample}{$variant_type}{$variant}}[2] =~ /dbSNP\=1/i ||
${$their_calls{$centre}{$sample}{$variant_type}{$variant}}[2] =~ /^rs\d+/i)
								{

	${$their_calls{$centre}{$sample}{$sample_stats}}[1]++; #$their_novel++;

									if ($variant_type eq
"snp"){${$their_calls{$centre}{$sample}{$sample_stats}}[3]++;} #$their_novel_snp++;

									else
{${$their_calls{$centre}{$sample}{$sample_stats}}[5]++;} #$their_novel_indel++;
								}

								#Find out what kind of variant it is (CSV file or
'SNPEFF_EFFECT=' annotations in VCF) in their data,
								$their_effect_type =
${$their_calls{$centre}{$sample}{$variant_type}{$variant}}[3];
								if ($their_effect_type eq ""){$their_effect_type = "Effect
unclassified in $centre";}

								#Count how many times this kind of variant appears in
their's data.
```

```
		${$their_calls{$centre}{$sample}{'their_effect_types'}{$their_effect_type}}[0]++;
#$their_effect_type++
					}
				}

				#push their effect types into an array, which can be looped over later
for file printing.
				for $their_effect_type (sort keys
%{$their_calls{$centre}{$sample}{'their_effect_types'}})
				{
					unless (grep {$their_effect_type eq $_} @all_effect_types)
					{
						push (@all_effect_types, $their_effect_type);
					}
				}

				#the percent of my calls that they have called.
				${$their_calls{$centre}{$sample}{$sample_concordance}}[0] =
$total_match / ${$my_calls{$sample}{$sample_stats}}[0] * 100;
				${$their_calls{$centre}{$sample}{$sample_concordance}}[2] =
$snp_match / ${$my_calls{$sample}{$sample_stats}}[2] * 100;
				${$their_calls{$centre}{$sample}{$sample_concordance}}[4] =
$indel_match / ${$my_calls{$sample}{$sample_stats}}[4] * 100;

				#the percent my novel calls that they found.
				${$their_calls{$centre}{$sample}{$sample_concordance}}[1] =
$novel_in_mine_match_total / ${$my_calls{$sample}{$sample_stats}}[1] * 100;
				${$their_calls{$centre}{$sample}{$sample_concordance}}[3] =
$novel_in_mine_match_snp / ${$my_calls{$sample}{$sample_stats}}[3] * 100;
				${$their_calls{$centre}{$sample}{$sample_concordance}}[5] =
$novel_in_mine_match_indel / ${$my_calls{$sample}{$sample_stats}}[5] * 100;

				#the percent of their calls that I found.
				${$their_calls{$centre}{$sample}{$sample_concordance}}[6] =
$total_match / ${$their_calls{$centre}{$sample}{$sample_stats}}[0] * 100;
				${$their_calls{$centre}{$sample}{$sample_concordance}}[8] =
$snp_match / ${$their_calls{$centre}{$sample}{$sample_stats}}[2] * 100;
				${$their_calls{$centre}{$sample}{$sample_concordance}}[10] =
$indel_match / ${$their_calls{$centre}{$sample}{$sample_stats}}[4] * 100;

				#the percent their novel calls that I found.
				${$their_calls{$centre}{$sample}{$sample_concordance}}[7] =
$novel_in_theirs_match_total / ${$their_calls{$centre}{$sample}{$sample_stats}}[1] * 100;
				${$their_calls{$centre}{$sample}{$sample_concordance}}[9] =
$novel_in_theirs_match_snp / ${$their_calls{$centre}{$sample}{$sample_stats}}[3] * 100;
				${$their_calls{$centre}{$sample}{$sample_concordance}}[11] =
$novel_in_theirs_match_indel / ${$their_calls{$centre}{$sample}{$sample_stats}}[5] * 100;
			}
		}
}
################################################################################
################################################################################
############
#end of concordance subroutine
################################################################################
################################################################################
############
```

```
###########################################################################
###########################################################################
############
#printer
#print header/column titles: sample, my tot var, % in other pipeline, novel var, my SNPs, % in
other pipeline, novel SNPs, my InDels, % in other pipeline, novel InDels, their var, % in mine,
novel var, their SNPs... their InDels...
#also for headers, loop over the array of effect types and print each type.
#
###########################################################################
###########################################################################
############
sub printer
{
	#name of file to print to.
	$stats_file = "/home/marc/Documents/work/exome
analysis/Output/Pipeline_comparisons_updated.csv";

	print "Now printing to $stats_file\n";

	open STATS, ">$stats_file" or die "output file $stats_file cannot be opened";

	#print the header line to the file (my totals, their totals, concordance and effect types).
	print STATS "Sample,Marc's total variants,Novel variants,Marc's SNPs,Novel
SNPs,Marc's InDels,Novel InDels,BGI/UNC total variants,Novel variants,BGI/UNC's
SNPs,Novel SNPs,BGI/UNC's InDels,Novel InDels,";
	print STATS "Percent (all) they found in Marc's pipeline,Novel,Percentage (SNP) they
found in Marc's pipeline,Novel,Percent (InDel) they found in Marc's pipeline,Novel,Percent (all)
Marc found in their pipeline,Novel,Percentage (SNP) Marc found in their pipeline,Novel,Percent
(InDel) Marc found in their pipeline,Novel";
	foreach (@all_effect_types){print STATS ",$_";}

	#Loop over my hash and print the totals. If the sample is also in their hash, print their
totals and the percents. then print the effect types.
	for $sample (sort keys %my_calls)
	{
		#Used to decide whether or not one of the sequencing centres has also done
some analysis on the sample.
		$sample_match = 0;

		$print_sample = "House\_$sample";

		#Builds a sample name, consisting of the sequencing centre and the sample.
This is used to keep the samples together by centre when printing to a file (alphabetically).
		for $centre (sort keys %their_calls)
		{
			if (grep {$sample eq $_} sort keys %{$their_calls{$centre}})
			{
				$print_sample = "$centre\_$sample";
				last;
			}
		}

		#start building the line to print to file. Start with a new line character and the
centre/sample name combo.
		$stats = "\n$print_sample";

		#foreach sample, gather stats on the total number of variants, SNPs, etc that I
called and each of the seq centres called where applicable.
		for $sample_stats (sort keys %{$my_calls{$sample}})
```

```
        {
            #looping over the sample_stats array in my hash, ignore the other keys
at this level.
            next unless $sample_stats =~ /stats$/i;

            #Add each of the statistics array values to the printing line.
            foreach $total (@{$my_calls{$sample}{$sample_stats}}){$stats .=
",$total";}

            #find the corresponding sample in the seq centres hash and add their
statistics to the printing line as well.
            for $centre_loop (sort keys %their_calls)
            {
                $centre = $centre_loop;
                if (grep {$sample eq $_} sort keys %{$their_calls{$centre}})
                {
                    $sample_match = 1;

                    foreach $total
(@{$their_calls{$centre}{$sample}{$sample_stats}}){$stats .= ",$total";}
                }

                #saves time by not searching rest of their hash if you have
found the sample.
                last if $sample_match == 1;
            }
        }

        #find the corresponding sample in their hash and gather concordance data for
printing.
        for $sample_concordance (sort keys %{$their_calls{$centre}{$sample}})
        {
            next unless $sample_concordance =~ /concordance$/i;

            if (grep {$sample eq $_} sort keys %{$their_calls{$centre}})
            {
                foreach $concordance
(@{$their_calls{$centre}{$sample}{$sample_concordance}}){$stats .= ",$concordance";}
            }
        }

        #if the sample isnt in their calls, then there will be 6 empty fields where their
totals and percents should be.
        #printing 12 commas will keep the rest of the line aligned.
        unless ($sample_match == 1){$stats .= ",,,,,,,,,,,,,,,,,,";}

        #print effect types
        foreach $effect_type (@all_effect_types)
        {
            $stats .= ",";

            #if the sample has a variant which causes this effect type
            if (grep {$effect_type eq $_} sort keys
%{$my_calls{$sample}{'my_effect_types'}})
            {
                #print the number of times this effect types appears in this
sample.
                $stats .=
"${$my_calls{$sample}{'my_effect_types'}{$effect_type}}[0]";
            }
```

```
                              #if the sample has a variant which causes this effect type
                              if (grep {$effect_type eq $_} sort keys
%{$their_calls{$centre}{$sample}{'their_effect_types'}})
                              {
                                        #print the number of times this effect types appears in this
sample.
                                        $stats .=
"${$their_calls{$centre}{$sample}{'their_effect_types'}{$effect_type}}[0]";
                              }
                    }

                    #push the final assembled printing line to an array.
                    push (@stats, $stats);
          }

          #loop over each of the printing lines in the array and print each line alphabetically.
          #You will print each sample on a new line, grouped alphabetically into their sequencing
centres.
          foreach (sort @stats){print STATS "$_";}

          close STATS;
}
##############################################################################
##############################################################################
############
#end of printer
##############################################################################
##############################################################################
############
```

Custom Script 5

```
(note: There are proabably more comment lines than actual lines of code...)
#This script is designed to compare genotypes emitted from GATK (determined by reading in
from the final VCF files) with genotyping data from Illumina's GoldenGate assay.
#The script has to convert GATK GT's to Illumina's top/bottom stranded convention, which is
where things get very tricky indeed.

#Illumination - comparison of GATK genotype calls with Illumina genotype calls.
#Illumina's strand designation system make this very difficult and complex (from a
bioinformatician's viewpoint, anyway).
#This script functions based on the assumptions that:
#	A GATK genotype is a dinucleotide, which depending on the allele count (AC=n),
consists of either:
#		2 reference bases (AC=0),			e.g. AA
#		1 reference base and 1 variant base (AC=1),	e.g. AG
#		2 variant bases (AC=2),				e.g. GG.
#	The Illumina genotype is a dinucleotide which has been converted to Illumina's
strand/allele designation convention.
#		Top stranded heterozygote genotypes are: AC, AG, AT and CG. Every other
heterozygote genotype is bottom stranded.
#		For homozygote genotypes, strand is determined by seeing whether the ref/alt
dinucleotide is top or bottom stranded.
#			If top stranded then the GATK genotype does not need converting.
#			If bottom stranded and the ref/alt dinucleotide is not top stranded, use
the (reverse) complement of the homozygote genotype.
#			e.g. if GATK calls hom variant AA (with ref = G), ref/alt = GA, which
reversed is AG (top stranded) so the genotype remains AA.
#			e.g. but if GATK hom call = TT (with ref = C), ref/alt = CT, which is the
rev. complement of AG, so the genotype is complemented to AA.
#		If homozygous and the ref/alt nucleotides are homozygous, you're going to
need a bigger boat.

#There is a known flaw in the comparisons, but fortunately the situation in which it arises never
occurs when comparing my data with Chris's 153 SNPs.
#If dealing with a multi-allelic SNP, and the call from the VCF file is homozygous reference, then
you cannot know which SNP Illumina detected if their call is also homozygous.
#To find the alt allele, you would have to look at the strand in which SNP occurs, which is
possible to do by looking up the strand from %strands if the ref and alt bases are not
complementary.
#If the ref and alt are complementary, then you would have to look at the ref genome to
determine strand.
#If Illumina call a het multiallelic SNP, then you can look up their call in the %strands keys and
then look for a value beginning with the ref allele in the values for that keys to find the alt allele.


#Hash of BGI exomes, where sample name (key) is associated with sequencing reads name
(value)
%bgi_exomes = (		x => "110926_I174_FCC0343ACXX_L3_HUMvejXAAAAAPEI-9", x =>
"110930_I810_FCC02KFACXX_L6_HUMvejXABAAAPEI-12",
		x => "110926_I174_FCC0343ACXX_L3_HUMvejXACAAAPEI-1", x =>
"110926_I174_FCC0343ACXX_L3_HUMvejXADAAAPEI-2",
		x => "110930_I810_FCC0335ACXX_L5_HUMvejXAEAAAPEI-3", x =>
"110930_I810_FCC02KFACXX_L6_HUMvejXAFAAAPEI-4",
		x => "110930_I810_FCC02KFACXX_L6_HUMvejXAGAAAPEI-5", x =>
"110930_I810_FCC0335ACXX_L5_HUMvejXAHAAAPEI-7",
			x => "110930_I810_FCC0335ACXX_L5_HUMvejXAIAAAPEI-8", x =>
"110926_I174_FCC0343ACXX_L4_HUMvejXAJAAAPEI-9");


#Input from the command line lets the user choose the depth to filter out genotype calls from the
GATK data.
print "Please enter the minimum depth threshold for filtering out genotype calls.\n";
while ($dp_filter !~ /^\d+$/){chomp ($dp_filter = <>);}

#first sub routine, get the illumina data into a hash.
illuminate();

#second sub routine, get my data into a hash
enlighten();


##############################################################################
########################################
#print output to a table
##############################################################################
########################################
open TABLE, ">/home/marc/Documents/work/exome analysis/Output/illumina
genotype_comparisons/genotype_comparisons_table_".$dp_filter."\.csv";

print TABLE "Illumina vs Marc's genotype for sample:";
for $sample (sort keys %percent_matches){next if ($sample == 71021);print TABLE
",,$sample";}
print TABLE "\n";

for $sample (sort keys %percent_matches){next if ($sample == 71021);print TABLE
",Illumina,Marc";}
print TABLE "\n";

for $rsID (sort keys %table)
{
	print TABLE "$rsID";

	for $sample (sort keys %percent_matches)
	{
		if (grep {$sample eq $_} sort keys %{$table{$rsID}})
		{
			print TABLE
",${$table{$rsID}{$sample}}[0]\,${$table{$rsID}{$sample}}[1]";
		}

		else {print TABLE ",,";}
	}

	print TABLE "\n";
}

#prints the percent of match for each sample
for $sample (sort keys %percent_matches)
{
	print TABLE ",,${$percent_matches{$sample}}[0]";
	$average_concordance += ${$percent_matches{$sample}}[0];
}

print TABLE "\n";
```

```
for $sample (sort keys %percent_matches)
{
			print TABLE
",${$percent_matches{$sample}}[2],${$percent_matches{$sample}}[1]";
}

print TABLE "\n\n";

#print a table of x/n calls, y% concordance for each sample vertically.
for $sample (sort keys %percent_matches)
{
			print TABLE
"$sample,${$percent_matches{$sample}}[2]\/${$percent_matches{$sample}}[1],${$percent_mat
ches{$sample}}[0]\n";
}


$average_concordance /= scalar keys %percent_matches;
$total_percent = $total_matches / $total_genotypes * 100;
print TABLE "Total:matches:percent,$total_genotypes\/$total_matches,$total_percent";
################################################################################
########################################

################################################################################
##################################
#sub routine illumination.
#Loads the data in the illumina genotypes file into a hash of hashes of hashes of arrays,
#%genotypes = (%samples = (%genes => {@snps => [$genotype]}))
#Necessary information from the file:
#first line contain sample names, so use these to figure out what columns the actual genotypes
are in
#(split into array, note the element number of samples - they're the only purely numerical
values).
#From other lines, gather gene name, chromosome, rsID and position (from dbSNP file?)
################################################################################
##################################
sub illuminate
{
		#The output file that the chromosome positions for each rsID will be written.
		#The region file will be used to parse to GATK UnifiedGenotyper. Each line contains the
chromosome:position-position of a SNP.
		#The interval file is a copy of the region file with rsIDs and ref. and alt. alleles from
dbSNP at the end of each line
		$interval_list = "/home/marc/Documents/work/exome
analysis/Output/Illumina_genotype_regions_rsID.interval_list";
		$region_file = "/home/marc/Documents/work/exome
analysis/Output/illumina_genotype_regions.interval_list";

		#Users decides whether or not to call GATK's unified genotyper to produce the
necessary VCF files.
		#Cannot use VCF files from Marc's pipeline as you need to call reference sites as well,
which haven't been called in the pipeline.
		while ($call_gatk ne "y" && $call_gatk ne "n")
		{
			print "Would you like to call GATK's Unified Genotyper to produce VCF files for
each of the variants in the Illumina data?\n";
			print "Please enter 'y' or 'n'.\n";

			#remove any new line characters from the input.
			chomp ($call_gatk = <STDIN>);
			#ensure the input is lower case.
			$call_gatk =~ tr/[A-Z]/[a-z]/;

			#ensure input is either 'y' or 'n'.
			if ($call_gatk ne "y" && $call_gatk ne "n")
			{
				$call_gatk = "";
				print "Please enter 'y' or 'n'\n";
			}
		}

		#Unless an interval list has already been created, read in data from dbSNP to a hash
(rsID = chr:pos-pos). Some entries in this will be printed later.
		unless (-f $interval_list){get_regions();}

		#load the file created by get_regions subroutine and create the hash (rsID = chr:pos-
pos).
		#the if statement is necessary as if the file hasn't been created then &get_regions will
have already loaded the necessary data (plus more!) into memory.
		#Could perhaps modify this so that the file is printed by &get_regions, and the file will
be loaded into memory then.
		if (-f $interval_list)
		{
			#open the file.
			open INTERVAL, $interval_list or die "$interval_list exists but cannot be
opened. You can't explain that!\n";

			#load data in from the intervals list and store the data in a hash.
			#line format: chr:pos-pos rsID alleles
			while ($int_line = <INTERVAL>)
			{
				chomp $int_line;

				#split the line into an array
				@int_line = split (/\s/, $int_line);

				$chr_pos = $int_line[0];
				$rsID = $int_line[1];
				$alleles = $int_line[2];

				if ($chr_pos =~ /^[XY\d]{1,2}/i){$chr = $&;}
				if ($chr_pos =~ /\d+$/){$pos = $&;}
				$position_key = $chr."\_".$pos;

				#skips if there is more than 1 allele for the SNP. Multiple alleles are
fine, it is possible to 'reverse engineer' the Illumina genotype to find which allele they detected.
				#therefore these lines should be ignored (but left in just in case!).
				#$multi_alleles = length ($alleles);
				#next if $multi_alleles > 2;

				#this hash contains the chromosomeal position for each rsID.
				$dbSNP{$rsID} = $chr_pos;
				$rsIDs{$position_key} = $rsID;

				#this hash contains the alt alleles for each SNP.
				$alt_alleles{$rsID} = $alleles;
			}

			close INTERVAL;
			#print the data in the dbSNP hash out to a new file for parsing to the GATK
genotyper. This file is identical to the interval file but without the rsIDs.
			unless (-f $region_file)
			{
				open REGION, ">$region_file" or die "$region_file could not be
created.\n";
				for $rsID (sort keys %dbSNP){print REGION "$dbSNP{$rsID}\n";}
				close REGION;
			}
		}

		#open the illumina file containing the Illumina genotyping data.
		$illumina_file = "/home/marc/Documents/work/exome analysis/Miscellaneous/illumina
genotypes 153.csv";
		open illumina, "$illumina_file" or die "the illumina file ($illumina_file) could not be
opened.\n";

		#read in the lines of the illumina file and capture the relevant information.
		while ($illumina_line = <illumina>)
		{
			#the counter of columns in the first line, set to 0. The first line contains sample
names, etc.
			$column_count = 0;

			#split the line into an array.
			@illumina_line = split (/\,/, $illumina_line);

			#if the line is the first line of the file.
			if ($illumina_line =~ /^Gene\_Name/i)
			{
				#loop over the line array.
				foreach $column (@illumina_line)
				{
					#if the element is only numerical (or is, eg. 103006B,
103010B...) (i.e. a sample name).
					if ($column =~ /^\d+B?$/i)
					{
						#tie the sample name to its column number. the
column number will be necessary to tell which genotypes belong to which sample later.
						$column_sample = $column_count."\_".$column;

						#push the column number/sample into an array.
						push (@column_samples, $column_sample);
					}

					#count what column the current element is in. Must be done
after pushing samples into array or the column number they contain will be +1 of their actual
column.
					$column_count++;
				}

				#NECESSARY! move past the first line of the file. Necessary!
				next;
			}

			#capture some useful (~ essential) data from the illumina line. Note that rsID
won't always be an rsID but some other thing which is meaningless.
			#the chromosome positions will almost certainly not match dbSNP, my data or
anyone else's data (Illumina data produced using an older ref genome??).
```

```
        $gene = $illumina_line[0];
        #$chr = $illumina_line[1];
        $rsID = $illumina_line[3];

        #the chromosome and position are set by looking in the dbSNP hash, using the
rsID as a reference.
        #Therefore only genotypes with a valid rsID can be processed.
        #chromsome will actually be the chr:pos-pos, but will be modified soon.
        $chromosome = $dbSNP{$rsID};

        #get the position of the SNP.
        #gets the '-pos' from the chromosome, and removes '-'.
        if ($chromosome =~ /\-[0-9]+/){$position = $&;$position =~ s/\-//;}
        else {$position = "unknown";}

        #set the chromosome number (the first numbers/letter from chr:pos-pos).
        if ($chromosome =~ /^[0-9a-z]+/i){$chromosome = $&;}
        else {$chromosome = "unknown";}

        #the key used for a hash later. Key contains the chr, pos and rsID.
        $hash_key = "$chromosome\_$position\_$rsID";

        #push each rsID into an array, if the rsID is in dbSNP. Will ensure only valid
rsIDs are stored in the array.
        if ($dbSNP{$rsID})
        {
            #count the number of variants/rsIDs in the Illumina file.
            $illumina_rsID_count++;

            #also ensures no duplicate rsIDs are stored. probably not necessary.
            unless (grep {$rsID eq $_} sort @rsID){push (@rsID, $rsID);}
            print "$illumina_rsID_count $rsID\n";
        }

        #look in Illumina's genotype file and store the genotype for each SNP in each
sample into a hash.
        foreach $column_sample (@column_samples)
        {
            #split the columnNumber_sample into an array (@sample = [number,
sample])
            @sample = split (/\_/, $column_sample);

            #set the column and sample.
            $column = $sample[0];
            $sample = $sample[1];

            #get the genotype for the sample from the illumina line.
            #Because column number was attached to sample name earlier,
$column points to the correct column number of the @illumina_line array.
            $genotype = $illumina_line[$column];

            #load the hash of hashes of hashes of arrays with all the (useful, i.e.
only actual GT calls) genotype data with a valid rsID.
            #genotype is a dinucleotide (e.g. AT, CC, GA).
            if ($genotype =~ /^[a-z]{2}$/i)
            {
                if ($dbSNP{$rsID})
                {
                    #remember hash key = chr_pos_rsID.
```

```
	${$illumina_genotypes{$sample}{$gene}{$hash_key}}[0] = $genotype;
				}
			}
		}
	}

	#Prints the interval list here if it hasn't been already. I.e. will only be printed if reading in
from the actual dbSNP file for the first time.
	unless (-f $interval_list)
	{
		#open the output file for printing the region for each rsID in Illumina's list.
		open INTERVAL, ">$interval_list" or die "the interval list ($interval_list) could
not be opened.\n";
		open REGION, ">$region_file" or die "$region_file could not be created.\n";

		#print each of the rsID regions to the output files.
		foreach $rsID (sort {$a <=> $b} @rsID)
		{
			#count how many rsIDs are in the array
			$illumina_rsIDs++;

			#set the region and interval (both will = chr:pos-pos rsID alleles for
now).
			$region = $dbSNP{$rsID};
			$interval = $dbSNP{$rsID};

			chomp $region;
			chomp $interval;

			#remove the space, rsID and alleles from the end of the region line.
			#What's left is chr:pos-pos. This will be printed to a file used by Unified
Genotyper.
			$region =~ s/\s.+//i;

			#Store the rsID for each SNP in %alt_alleled. The alt_alleles hash will
be used in &enlighten to get the alternate allele.
			$alt_alleles{$rsID} = $dbSNP{$rsID};

			print "printing out $illumina_rsIDs\t$illumina_rsIDs\t$rsID
$alt_alleles{$rsID}\n";

			#print the full line (chr:pos-pos rsID alleles) to the interval file.
			print INTERVAL "$interval\n";

			#print the data in the dbSNP hash out to a new file for parsing to the
GATK genotyper. This file is identical to the interval file but without the rsIDs or SNP bases.
			print REGION "$region\n";
		}

		#close INTERVAL and REGION.
		close INTERVAL;
		close REGION;
	}

	#clear the dbSNP hash and rsID array to save memory.
	%dbSNP = ();
	@rsID = ();

}
```

```
##########################################################################
##################################
#End of illumination sub routine
##########################################################################
##################################

##########################################################################
##################################
#sub routine get_regions
#reads in the variants in a downloaded dbSNP file, and extracts the chromosome ID and
position for each
#rsID in the Illumina genotype's file. Will take an hour or more to run since the dbSNP file is
large (>4.6 GB for v132, subsequent versions are a lot bigger!!)
#will use a lot of memory, so maybe find a way to improve this?
##########################################################################
##################################
sub get_regions
{
	#Open up the dbSNP VCF file so that the chromosomal positions of all rsIDs/SNPs can
be obtained.
	#alt. alleles are necessary when specifying regions to GATK's UnifiedGenotyper tool,
which will be used to call genotypes in Marc's data at sites that have no variants detected.
	#Searching dbSNP for these regions will take up to an hour. The output is a file
containing the regions, and this step will only have to be done once,
	#unless the output file is lost.
	$dbSNP_file = "/media/Brachiosaur/Exomes/database resources/dbsnp_135.b37.vcf";

	#In all fairness I could make this part a lot less computationally demanding. I'm not sure
why it was so expensive in the first place. I'll put it on my 'to do... eventually' list, but for now if it
ain't broke, don't fix it!
	print "Getting regions from dbSNP database. This could take over an hour and will use
a LOT of memory (>12 GB). This is going to hurt in the morning.\n";

	#loop over the dbSNP file.
	open dbSNP, "$dbSNP_file" or die "the dbSNP file ($dbSNP_file) could not be
opened.\n";
	while ($dbSNP = <dbSNP>)
	{
		#ignore header lines.
		next if $dbSNP =~ /^\#/;

		#split the line into an array.
		@dbSNP = split (/\s/, $dbSNP);

		#under construction - use to build a string of variants for each chromosome, will
save memory when reading in from dbSNP.
		#will also need to heavily edit &illuminate subroutine to account for the
changes.
		# #When a new chromsome is encountered...
		# unless ($dbSNP[0] == $chr)
		# {
			# Push the last chromosome's string of variants into %dbsnp.
			# if ($chr){$dbsnp{$chr} =  $chromosome_variants;}
			#
			# Clear the last chromosome's string, ready to build up for the new
chromosome.
			# $chromosome_variants = "";
		# }

		#the rsID from the dbSNP line.
```

```
			$rsID = $dbSNP[2];

			#get the data for the 3 missing SNPs from Chris' list of 153 SNPs. 3 SNPs were
merged to new rsIDs, so search dbSNP using these updated rsIDs.
			#next unless ($rsID eq "rs1130222" || $rsID eq "rs10511" || $rsID eq
"rs2228612");

			#the chromosome and position of the variant.
			$chr = $dbSNP[0];
			$pos = $dbSNP[1];

			#the reference and alternate (variant) allele.
			$ref = $dbSNP[3];
			$alt = $dbSNP[4];

			#Store the region in a hash, using the rsID as the key.
			$dbSNP{$rsID} = "$chr:$pos-$pos\t$rsID\t$ref$alt\n";

			#$chromosome_variants .= "$chr:$pos-$pos\t$rsID\t$ref$alt\n";
		}

		close dbSNP;
}
##########################################################################
##################################
#End of get_regions sub routine
##########################################################################
##################################

##########################################################################
##################################
#sub routine elighten.
#Loads the data in my variant call format (VCF) files into a hash.
#information useful (~necessary) for capture includes the chromosome, position, rsID, ref and
alt alleles, and the 'AC=n;' field.
#illumina have a strangely awkward (also ingenious, but mainly awkward) way of expressing
their genotypes based on a 'strand' basis;
#they always express the genotype on the 'top' strand. This means to compare my GTs against
theirs, I have to convert mine to the top
#strand format as well. Tool up boys, we're going shopping...
##########################################################################
##################################
sub enlighten
{
		#a list of the sequencing centres, useful for finding vcf directories.
		@sequencing_centres = qw(BGI BGIx13 BGIx18 UNC);

		#The Illumina strand system. If you have a change between an A and a G (A-G) for
example, then you would use AG (for hets) or AA/GG (for homs, depending on which base is
now homozygous).
		#If the change is between the complementary base pairs of these (i.e. between T-C)
then you would still use AG (for hets).
		#The hash table below contains all the 'top' strand bases as values and the 'bottom'
strand bases as keys, so you can search with my data and convert the GTs to the Illumina style
		#top strand format.
		%strands = (AG => ["AG", "GA", "CT", "TC"], AC => ["AC", "CA", "GT", "TG"], AT =>
["AT", "TA"], CG => ["CG", "GC"]);
		%reverse_strands = reverse %strands;

		#This hash contains the bases of DNA and their complementary bases.
```

```
	%complementary_bases = (A => 'T', T => 'A', C => 'G', G => 'C');

	#Get the reference sequence for each chromosome and store it in a hash. The
sequences will be stored line by line (not base by base), and prefixed with the line number and
"_".
	open REF, "/media/Brachiosaur/Exomes/reference genome/human_g1k_v37.fasta";
	while ($ref_line = <REF>)
	{
		chomp $ref_line;

		#The header line for each chromosome (>1 ...other info).
		if ($ref_line =~ /^\>[a-z0-9]+/i)
		{


			#Print the length of each chromosome (will actually be printed at the
start of the next chromosome, therefore the last chromosome length will not be printed.
			unless ($ref_chromosome eq "")
			{
				print "Chromosome: $ref_chromosome\tLength: $chr_length\n";

				#Store the line and number (lineNumber_lineSequence) in the
hash, with the chromosome as the key.
				push (@{$reference_genome{$ref_chromosome}}, $chr_seq);
			}

			$chr_seq = "";

			#Reset the number of lines and length for each chromosome.
			$line_number = 0;
			$chr_length = 0;

			#Set the chromosome ID (eg. 1, 5, 19, X, Y, MT)
			$ref_chromosome = $&;
			$ref_chromosome =~ s/\>//;
		}

		#the line contains nucleotide sequence.
		elsif ($ref_line =~ /^[ACGTN]/i)
		{
			#add the length of the line to the total chromosome length. The b37
build uses FASTA format (so 60 bases per line), although the final line may be shorter.
			$chr_length += length $ref_line;

			#Store the chromosome sequence as a string, with lines starting with
the line number and ending with a colon.
			$chr_seq .= "$line_number\_$ref_line\n";

			#Increase line number. Since this is done after storage in the hash, the
first line stored for each chromosome is line 0.
			$line_number++;
		}
	}

	close REF;

	#Prepare to read in data from Marc's VCF files.
	foreach $centre (@sequencing_centres)
	{
		#set the vcf directory.
		$recal_dir = "/media/Triceratops/Exomes/$centre
output/12_GATK_tablerecalibration";

		#The directory the VCF files from the GATK are found.
		$vcf_dir = "/home/marc/Documents/work/exome analysis/Output/illumina
genotype_comparisons/unified_genotyper MBQ 10";

		#open the VCF directory and read in the VCF files if they are
'variants_annotated.vcf' files. Also reads in recalibrated .bam files in case the user wanted to
call the GATK.
		opendir VCF_DIR, $vcf_dir or die "could not open $vcf_dir. Did you mount the
drive?\n";
		opendir RECAL_DIR, $recal_dir or die "could not open $recal_dir. Did you
mount the drive?\n";

		#store the names of the .bam files in an array. These will be the input for the
Unified Genotyper call.
		@recal_files = grep {-f "$recal_dir/$_" && $_ =~ /recalibrated\.bam$/i} readdir
RECAL_DIR;

		#If the user specified yes, prepare the call to the GATK UG and store them in
an array.
		if ($call_gatk eq "y")
		{
			#for every recalibrated BAM file.
			foreach $recal_file (@recal_files)
			{
				#set the sample name.
				$sample = $recal_file;
				$sample =~ s/\_recalibrated\.bam$//i;

				#set the sample name if it is a BGIx10 exome.
				for $exomes (sort keys %bgi_exomes)
				{
					#gets the sample name from the barcode thing that
BGI insisted on calling some of their FASTQ files.
					if ($bgi_exomes{$exomes} eq $sample){$sample =
$exomes;}
				}

				#the command line string used to call GATK.
				#delete/add these switches -baq
CALCULATE_AS_NECESSARY -MBQ 17 -stand_call_conf 30 -stand_emit_conf 10
				$gatk_call = "java -jar \"/home/marc/Documents/bioinformatics
software/GenomeAnalysisTK/GenomeAnalysisTK.jar\" -T UnifiedGenotyper -R
\"/media/Brachiosaur/Exomes/reference genome/human_g1k_v37.fasta\" -L \"$region_file\" -glm
both -mbq 10 -stand_call_conf 30 -stand_emit_conf 10 -D
\"/media/Brachiosaur/Exomes/database resources/dbsnp_135.b37.vcf\" -I
\"$recal_dir\/$recal_file\" -o \"/home/marc/Documents/work/exome analysis/Output/illumina
genotype_comparisons/unified_genotyper MBQ 10/$centre\-$sample\.vcf\" -out_mode
EMIT_ALL_SITES -metrics \"/home/marc/Documents/work/exome analysis/Output/illumina
genotype_comparisons/unified_genotyper MBQ 10/$sample\_metrics\.txt\"";

				#VQSR on such a small amount of sites is apparently not
feasible. Was only added to see if it enhanced concordance. Looking at the pipeline's VQSR
files, it makes no difference.
				# $var_recal_call = "java -jar
\"/home/marc/Documents/bioinformatics software/GenomeAnalysisTK/GenomeAnalysisTK.jar\"
-T VariantRecalibrator -R \"/media/Brachiosaur/Exomes/reference
genome/human_g1k_v37.fasta\" -input \"/home/marc/Documents/work/exome
analysis/Output/illumina genotype_comparisons/unified_genotyper/$centre\-$sample\.vcf\" -
recalFile \"/home/marc/Documents/work/exome analysis/Output/illumina
genotype_comparisons/unified_genotyper/$centre\-$sample\_recal\.csv\" -tranchesFile
\"/home/marc/Documents/work/exome analysis/Output/illumina
genotype_comparisons/unified_genotyper/$centre\-$sample\_tranches\.tranches\" -mode SNP -
an HaplotypeScore -an MQRankSum -an ReadPosRankSum -an MQ -an DP -
resource:hapmap,VCF,known=false,training=true,truth=true,prior=15.0
\"/media/Brachiosaur/Exomes/database resources/hapmap_3.3.b37.sites.vcf\" -
resource:omni,known=false,training=true,truth=false,prior=12.0
\"/media/Brachiosaur/Exomes/database resources/1000G_omni2.5.b37.sites.vcf\" -
resource:dbsnp,VCF,known=true,training=false,truth=false,prior=8.0
\"/media/Brachiosaur/Exomes/database resources/dbsnp_135.b37.vcf\" -mG 4 -percentBad
0.05";
                                # $apply_recal_call = "java -jar
\"/home/marc/Documents/bioinformatics software/GenomeAnalysisTK/GenomeAnalysisTK.jar\"
-T ApplyRecalibration -R \"/media/Brachiosaur/Exomes/reference
genome/human_g1k_v37.fasta\" -input \"/home/marc/Documents/work/exome
analysis/Output/illumina genotype_comparisons/unified_genotyper/$centre\-$sample\.vcf\" -
recalFile \"/home/marc/Documents/work/exome analysis/Output/illumina
genotype_comparisons/unified_genotyper/$centre\-$sample\_recal\.csv\" -tranchesFile
\"/home/marc/Documents/work/exome analysis/Output/illumina
genotype_comparisons/unified_genotyper/$centre\-$sample\_tranches\.tranches\" -o
\"/home/marc/Documents/work/exome analysis/Output/illumina
genotype_comparisons/unified_genotyper/$centre\-$sample\.vcf\"";

                                #push the GATK call into an array (providing it is not already in,
which it shouldn't be).
                                unless (grep {$gatk_call eq $_} @gatk_calls)
                                {
                                        push (@gatk_calls, $gatk_call);

                                        #not needed unless using VQSR.
                                        #push @{$system_calls{$sample}}, $gatk_call,
$var_recal_call, $apply_recal_call;
                                }
                        }
                }
        }

        #if the user specified yes, call the GATK UG for each sample.
        unless ($call_gatk eq "n")
        {
                #Call the GATK.
                foreach $call (@gatk_calls)
                {
                        $call_number++;
                        print "\nNow calling Unified Genotyper on sample number
$call_number\n";
                        system ($call);
                }

                #Again, only used for VQSR.
                # foreach $call (@{$system_calls{$sample}})
                # {
                        # $call_number++;
                        # print "\nNow calling GATK on sample number $call_number\n";
                        # print "\n$call\n";
                        # system ($call);
                # }
        }
```

```
    #get the names of the VCF output files from the GATK UG step.
    @vcf_files = grep {-f "$vcf_dir/$_" && $_ =~ /\.vcf$/i} readdir VCF_DIR;

    #for each of the VCF files in the directory (i.e. for each sample);
    foreach $vcf_file (@vcf_files)
    {
        #resets the number of (published and novel) variants in the VCF file to 0, along
with the number of Illumina and my genotypes per sample, and the number of matches.
        $variant_count = 0;
        $illumina_genotypes_total = 0;
        $my_genotypes_total = 0;
        $my_count = 0;
        $genotype_matches = 0;

        $depth_filtered = 0;
        $q_filtered = 0;
        $no_GT_filtered = 0;
        $no_comparison_possible = 0;
        $info = 0;

        #This array is used to skip chromosomes when searching the VCF files to
decrease run time.
        @chromosomes = ();

        #set the centre and sample name. More work is executed below for bgix10
samples.
        $centre = $vcf_file;
        $centre =~ s/\-.+$//;

        $sample = $vcf_file;
        $sample =~ s/\.vcf$//i;
        $sample =~ s/$centre\-//;
        $sample =~ s/\_pe$//i;

        #ignores the duplicate samples from the BGIx18, in which BGI sent 2 copies of
2 samples (the second copy had 'extra' sequencing information in to reach a depth threshold).
        #this script will ignore the second copy and only use the first copy, as the
second copies are unlikely to contain new genotype data.
        if ($sample =~ /B\d$/i)
        {
            if ($sample =~ /2$/){next;}
            else {$sample =~ s/\d$//;}
        }

        #This sample was accidentally mistaken somewhere for 71012. As a result,
71012 was sequenced again instead of 71021.
        #Therefore 71021 contains the same genotypes as 71012. Not useful for this
script, but it means 71012 has twice as much sequencing coverage.
        next if $sample =~ /71021/;

        #If the sample is an UNC sample, only use the first bit of the sample name, and
remove the zeros (eg. 042031_02136_s_6 becomes 42031).
        #This is to match the illumina sample names found in the illumina file.
        if ($centre eq "UNC" && $sample =~ /\_.+/i){$sample =~ s/$&//;}
        if ($centre eq "UNC" && $sample =~ /^0+/){$sample =~ s/$&//;}

        #set the sample name if it is a BGIx10 exome - convert from BGI's barcode
style name.
        for $exomes (sort keys %bgi_exomes)
        {
            #gets the sample name from the barcode thing that BGI insisted on
calling their FASTQ files.
            if ($bgi_exomes{$exomes} eq $sample){$sample = $exomes;}
        }

        print "Now reading in information from $sample\n";

        #open the VCF file.
        open VCF, "$vcf_dir/$vcf_file" or die "$vcf_file could not be opened. Why not?
Your guess is as good as mine.\n";

        #Loop over Marc's VCF file.
        while ($vcf_line = <VCF>)
        {
            #ignore the header lines and chromosomes that aren't 1-22, X or Y.
            next unless $vcf_line =~ /^[0-9xy]+/i;

            #used for printing if the variant is a complementary homozygote.
            $hom_comp = 0;

            #reset the strand for every variant in the file. Ensures all SNPs are
labelled correctly.
            $strand_designation = "";

            #Counts the number of variants per sample.
            $variant_count++;

            #split the line into an array.
            @vcf_line = split (/\s/, $vcf_line);

            #capture the useful (~essential) info from each line.
            $chromosome = $vcf_line[0];
            $position = $vcf_line[1];
            $rsID = $vcf_line[2];
            $ref_allele = $vcf_line[3];
            $alt_allele = $vcf_line[4];
            $filter = $vcf_line[6];
            $info = $vcf_line[7];
            $depth = $vcf_line[7];
            $chr_pos = $chromosome."\_".$position;

            #filter out low quality calls or sites with low sequencing coverage.
            if ($filter eq "LowQual")
            {
                $q_filtered++;

                print "FILTER: LowQual
$q_filtered\t$chromosome\t$position\t$rsID\t$ref_allele $alt_allele\n";

                #next is essential or else the SNP won't be filtered.
                next;
            }

            #Used to determine whether the call is wild type homozygous (AC=0),
heterozygous (AC=1) or mutant homozygous (AC=2).
            if ($info =~ /^AC\=[0-9]\;/i){$info = $&;}

            #the info field could be empty, due to the fact there is no sequencing
coverage at this site.
```

```
			else{$no_GT_filtered++;print "FILTER: no_GT 
$no_GT_filtered\t$chromosome\t$position\t$rsID\t$ref_allele $alt_allele\n";next;}

			#check the sequencing depth and filter out if lower than a user 
specified threshold.
			#Defined is defined in the VCF file by 'DP=n' - is the depth for that 
specific SNP.
			if ($depth =~ /DP\=\d+/i)
			{
				$depth = $&;
				$depth =~ s/DP\=//i;

				#Filter out if lower than the threshold.
				if ($depth < $dp_filter)
				{
					$depth_filtered++;
					print "FILTER: low depth 
$depth_filtered\t$chromosome\t$position\t$rsID\t$ref_allele $alt_allele\n";

					#Again, the next is necessary to filter out the SNP.
					next;
				}
			}

			#create a variable used to check for entries in the illumina data.
			#rsIDs can't be used as novel SNPs or reference sites won't be found.
			$chr_pos = $chromosome."\_".$position;

			#The lines in the VCF files contain only rsIDs of SNPs containing non-
reference alleles, so if the rsID in Marc's VCF file is '.' then the genotype must be a homozygous 
wild type.
			#Look in the Illumina hash (which has position info and the rsID 
together), and using the VCF's position retrieve Illumina's rsID.
			#Not all of Illumina's genotype have a valid rsID, and so proabably 
won't give an rsXXXXX number. Also, there are some SNPs which failed genotyping, these will 
not have
			#an entry in Marc's VCF, or in the genotypes comparison file for the 
output (counted as 'missing').
			if (grep {$chr_pos eq $_} sort keys %rsIDs)
			{
				$rsID = $rsIDs{$chr_pos};
			}
				#Loop over the genes for illumina's data for the current sample.
				# for $gene (sort keys (%{$illumina_genotypes{$sample}}))
				# {
					# #look at each SNP in the illumina data.
					# #the hash key = chr_pos_rsID.
					# for $hash_key (sort keys 
(%{$illumina_genotypes{$sample}{$gene}}))
					# {
						# #set the rsID by removing the chr and pos 
text.
						# if ($hash_key =~ /^$chr_pos\_/i)
						# {
							# $rsID = $hash_key;
							# $rsID =~ s/$&//i;
							#
							# #once the rsID has been found for 
the SNP, stop searching the Illumina data (stop looping over SNPs).
```

```
                            # last;

                        # }
                        #
                    # }
                    #
                    # #once the rsID has been found for the SNP, stop
searching the Illumina data (stop looping over genes).
                    # #unless ($rsID eq "\."){last;}
                # }

            #if the rsID is not present in the VCF file or Illumina data, then no
comparison is possible.
            if ($rsID eq "\.")
            {
                $no_comparison_possible++;
                print "FILTER: no_comparison
$no_comparison_possible\t$chromosome\t$position\t$rsID\t$ref_allele $alt_allele\n";
                #skip this SNP if no comparison is possible.
                next;
            }


    ######################################################################
########################################
            #THE BIG BAD ILLUMINA CONVERSION - ABANDON HOPE ALL YE
WHO ENTER HERE
            #Even if you can't figure out how this works, it works. Just trust me on
this!

    ######################################################################
########################################
            #There are 2 critical assumptions to be made:
            #       Assume that Illumina always call all their genotypes by the top
stranded conversion.
            #       Assume that Illumina detected the same genotype as the
GATK before converting it.
            #               At multi-alleleic sites in which the GATK called hom
reference, if Illumina don't call these sites
            #               homozygous reference then it is not possible to choose
an alternate allele or strand with complete confidence,
            #               therefore the resulting converted genotype may not
match the Illumina call.

            #When dealing with heterozygous genotypes, Illumina gave definitions
to define top stranded SNPs (where a SNP is in ref/alt base format).
            #Top stranded SNPs can be used as keys in a hash, with values
including the SNP itself and the bottom stranded combinations of reverse,
            #complementary and reverse complementary SNPs:
            #               - AC -> AC, CA, GT, TG
            #               - AG -> AG, GA, CT, TC
            #               - AT -> AT, TA
            #               - CG -> CG, GC
            #Using this table it is possible to decide that an A/C heterozygous SNP
at any site is top stranded, and that C/A, G/T and T/G are bottom stranded.
            #Although A/C is also a value in the hash, it is top stranded since it is
also a key. Under the assumption Illumina emit only top stranded genotypes,
            #you simply convert the heterozygous bottom stranded SNP to its top
stranded key (so GATK genotypes CA, GT and TG would be converted to genotype AC).
```

```
			#For designating strands when dealing with homozygotes, more
information is needed to make some logical rules.
			#Looking at the calls at sites which have known alternate alleles, I
conclude that:
			#	- If the SNP is top stranded:
			#		- homozygous reference genotypes will be a
dinucleotide of the reference allele.
			#		- homozygous alternate genotypes will be a
dinucleotide of the alternate allele.
			#	- If the SNP is bottom stranded:
			#		- homozygous reference genotypes will be a
dinucleotide of the complementary reference allele.
			#		- homozygous alternate genotypes will be a
dinucleotide of the complementary alternate allele.
			#	- In cases where the reference and alternate allele are
complementary,then the strand must be determined by looking
			#	  at the sequence surrounding the SNP site in the reference
genome.
			#		- this relies on comparing the sites immediately
upstream (site n+1) and downstream (site n-1) of the SNP (site n).
			#		- If n+1 is an A or T and n-1 is a C or G, then the strand
is designated top.
			#		- If n-1 is an A or T and n+1 is a C or G, then the strand
is designated bottom.
			#		- If both n+1 and n-1 are a A/T or both are a C/G, then
look at the sites next to those (n+2 and n-2). Keep going until a comparison is possible
			#		  between an A/T and a G/C.
			#From this we can say that if a site containing an A/C SNP is
homozygous variant, the genotype emitted would be CC. If a C/A SNP site was homozygous
variant,
			#the genotype would be TT.

			#If GATK calls a homozygous reference SNP, then there is an issue
which requires logic to resolve. Since the GATK will not emit an alternate allele for homozygous
reference
			#(normally it will not emit these sites, but is an option which has been
turned on for use in this script), you must look in dbSNP to find the alternate allele (GATK will
not emit rsIDs
			#either at these sites so query dbSNP using the chromosome and
position of the SNP). Once you have the alternate allele, you know the SNP (i.e. ref/alt) and can
designate the strand as above.
			#The problem arises when dbSNP contains multiple alleles at a site:
			#	- It is not always possible to say which allele Illumina detected
if the Illumina call is homozygous reference (you don't even know that their call IS hom ref, you
must assume it is because the
			#	  GATK call is). All you can do in this case, is designate a
strand based on the Illumina call.
			#		- If their call contains the reference allele, then you
know the SNP is top stranded. If their call is complementary to the reference allele, then you
know the SNP is bottom stranded.
			#	- Once the strand has been designated, you can follow the
rules for homozygous reference SNPs (above) to convert the GATK genotype to suit Illumina's
convention!


			#if the GATK call is homozygous reference, then look up the alternate
SNP.
			if ($info =~ /0/)
			{
```

```
                #get the alternate nucleotides from the hash of SNPs from
dbSNP (read in from the file created earlier).
                $alt_allele = $alt_alleles{$rsID};

                #remove the reference base from the alleles so that just the alt
alleles at this site are left.
                if ($alt_allele =~ /[a-z\,]+$/i){$alt_allele = $&;$alt_allele =~
s/^.//;}

                #if the SNP has multiple alleles (eg. A -> C,G,T), then split the
allele into an array and try to figure out from the illumina genotype which allele to use.
                if ($alt_allele =~ /\,/)
                {
                    #splits the alleles into an array (eg. so C,G,T would
become @multi_allelic = (C, G, T)
                    @multi_allelic = split (/\,/, $alt_allele);

                    #Look through the genes in Illumina's genotypes to find
the SNP identified by the rsID.
                    for $gene (sort keys
(%{$illumina_genotypes{$sample}}))
                    {
                        #Loop over Illumina's rsIDs - each genotype
call should have one.
                        for $illumina_rsID (sort keys
%{$illumina_genotypes{$sample}{$gene}})
                        {
                            #look at the entry in the Illumina hash
for the specific SNP in Marc's VCF file.
                            if ($illumina_rsID =~ /$rsID$/i)
                            {

                                #foreach of the dbSNP alleles
at this SNP.
                                foreach $allele (@multi_allelic)
                                {
                                    #set the (temporary)
mutation dinucleotide.
                                    $mutation =
$ref_allele.$allele;
                                    $rev_mutation =
reverse $mutation;

                                    #If there are multiple
alleles which form a top stranded genotype with the reference allele, (eg. ref = A, alleles = G, C,
T, top GTs = AG, AC, AT)
                                    #then you do not have
to detect which specific allele Illumina found, all we need to know is that it is top stranded.
Assuming the call is reference
                                    #and the genotype is
top stranded, then the genotype formed by converting to Illumina's convention will be a dinuc of
the ref allele.
                                    #eg. whether Illumina
detected G,C or T does not matter, all will result in an AA genotype.
                                    if
(${$illumina_genotypes{$sample}{$gene}{$illumina_rsID}}[0] =~ /^$ref_allele/i)
                                    {
                                        #if the
mutation is on the top strand (ie. a key in the %strands hash).
```

```
                                                                      if ((grep
{$mutation eq $_} sort keys %strands) || (grep {$rev_mutation eq $_} sort keys %strands))
                                                                      {

      #assign the alt allele and stop looping over the possible alleles.

      $alt_allele = $allele;


      $strand_designation = "top";

      $strand_designations{$chr_pos} = $strand_designation;

                                                                            last;
                                                                      }
                                                                }

                                                                #Illumina's genotype is
the dinucleotide of the complementary ref allele.
                                                                elsif
(${$illumina_genotypes{$sample}{$gene}{$illumina_rsID}}[0] =~
/^$complementary_bases{$ref_allele}/i)
                                                                {
                                                                      #loop over the
keys in the strands hash.
                                                                      for $strand
(sort keys %strands)
                                                                      {
                                                                            #if the
mutation is a bottom stranded dinucleotide, set the allele and stop looping over the other
possible alleles.
                                                                            if
(grep {$mutation eq $_} sort @{$strands{$strand}})
                                                                            {

      $alt_allele = $allele;



      $strand_designation = "bottom";

      $strand_designations{$chr_pos} = $strand_designation;



      last;
                                                                            }
                                                                      }
                                                                }

                                                                #assign the strand
here now so that if the ref and alt alleles are complementary the strand for this SNP is not
reassigned later.

      $strand_designations{$chr_pos} = $strand_designation;
                                                          }
                                                    }
                                              }
                                        }
                                  }
```

```
			}

			#The ref and alt alleles, used later to determine which strand should be
used when converting to Illumina format.
			$mutation = $ref_allele.$alt_allele;
			$rev_mutation = reverse $mutation;

			#sample is homozygous at this position.
			if ($info =~ /2/ || $info =~ /0/)
			{
				#if the ref and alt alleles are not complementary (eg. are not
A/T or G/C).
				if ($alt_allele ne $complementary_bases{$ref_allele})
				{
					#if the mutation (ref/alt) or reverse mutation (alt/ref) is a
key in the strands hash (i.e. is on the top strand)
					if (grep {$mutation eq $_} sort keys %strands || grep
{$rev_mutation eq $_} sort keys %strands)
					{
						#if the genotype is a reference homozygote
then it will consist of a dinucleotide of the ref allele.
						if ($info =~ /0/){$genotype =
$ref_allele.$ref_allele;}#print "$ref_allele is not complementary to $alt_allele and is on the top
strand\t$genotype\n";}
						else{$genotype = $alt_allele.$alt_allele;}
					}

					#the mutation is on the bottom strand, so convert to the
top stranded genotype by complementing the observed genotype.
					else
					{
						for $strand (sort keys %strands)
						{
							#if the mutation is a value in the
strands hash array
							if (grep {$mutation eq $_} sort
@{$strands{$strand}})
							{
								#if the VCF call is
homozygous reference, the Illumina genotype is the dinucleotide of the base complementary to
the reference allele
								if ($info =~ /0/){$genotype =
$complementary_bases{$ref_allele}.$complementary_bases{$ref_allele};}#print "$ref_allele is
not complementary to $alt_allele and is on the bottom strand\t$genotype\n";}

								#if the VCF call is
homozygous variant, then the genotype is a dinucleotide of the base complementary to the alt
allele.
								else{$genotype =
$complementary_bases{$alt_allele}.$complementary_bases{$alt_allele};}
							}
						}
					}
				}

				#the mutation is homozygous and complementary.
				else
				{
					#this is used to store strands for checked SNPS -
strands determined by looking at the reference genome
```

```
					#are stored in here. Doing this speeds up the
computation as it does not have to check the reference genome for each comparison.
					if
($strand_designations{$chr_pos}){$strand_designation = $strand_designations{$chr_pos};}

					#Since Illumina's strand rules cannot be applied to
homozygous complementary SNPs, look in the reference genome to determine the strand
					#by comparing the bases immediately upstream and
downstream of the SNP site.
					else
					{
						#in this case, you need to determine whether
the genotype you called is on the 'top' or 'bottom' strand. These are not the actual top or bottom
strands,
						#but those according to Illumina. A top strand
is determined if the base 5' to the variant is an A or T and the base 3' to the variant is a C or G.
If it is vice versa then
						#the strand is designated bottom strand. If both
neighbour bases are A/T or C/G, then compare the bases next to those (i.e. variant pos +2 vs.
pos -2). Repeat this process
						#until the strand has been designated.

						#reset the necessary counters and strings to
the starting values.
						$neighbourhood = "";
						$strand_designation = "";
						$n = 1;
						$hom_comp = 1;

						#for each chromosome encountered in the ref
file, reset the number of lines it takes up
						#and the sum of the length of the lines to the
variant position to 0.
						#This, along with the next 'for loop' will ensure
that when encountering a variant on the same chromosome as the prev. variant,
						#you do not have to search from the
begginning of the chromosome.
						unless (grep {$chromosome eq $_} sort
(@chromosomes))
						{
							$i = 0;
							$i_plus = 0;
							$len_to_pos = 0;
							push (@chromosomes,
$chromosome);
						}

						@chr_seq = split (/\n/,
${$reference_genome{$chromosome}}[0]);

						#This loop counts initially from the first line of
the ref chromosome (line 0),
						#and then for each variant only from the line
the prev. variant was on (for each chromosome).
						for $i ($i_plus .. scalar (@chr_seq))
						{
							#line count increments with i, and
i_plus is always +1 more than i so that when looped, the last ref line will not be counted again.
							$line_count = $i;
							$i_plus = $i + 1;
```

```
                                #The DNA sequence of the reference
line.
                                $line = $chr_seq[$line_count];

                                #remove line number from the ref line
(which I added when storing in the hash).
                                if ($line =~ /^\d+\_/){$line =~ s/$&//;}

                                #sum of the length of the lines upto
and including the line which the variant is on.
                                $len_to_pos += length ($line);

                                #stop when the length of the lines is
bigger than the variant's position
                                #eg. pos = 12, len_to_pos = 60, so
stop (line_count would == 0)
                                if ($len_to_pos >= $position){last;}
                        }

                        #the line number of the prev and next
reference lines.
                        $line_minus = $line_count - 1;
                        $line_plus = $line_count + 1;

                        #Build a string containing the ref line that the
variant is on, along with the prev and next lines.
                        #This will be used for 'walking' when
determining the strand. It ensures that there is always a line of sequence on either side
                        #of the variant (unless it is very close to the
start or end of the chromosome). The length of this 'neighbourhood' can be increased if needed.
                        for $j ($line_minus .. $line_plus)
                        {
                                #the reference line (i.e. the DNA
sequence). Remove the line numbers from the start.
                                $ref_line = $chr_seq[$j];
                                if ($ref_line =~ /\d+\_/){$ref_line =~
s/$&//;}

                                #add the line to the neighbourhood.
The neighbourhood will eventually contain the prev line, variant line and next line in a continual
DNA sequence.
                                $neighbourhood .= $ref_line;

                                #Count the position in the
neighbourhood that the variant is at.
                                #equation 1: length (prev line) + lengh
(var line) - (length_to_pos - var pos) - 1.
                                if ($j ==
$line_minus){$neighbourhood_pos = length ($ref_line);}
                                elsif ($j ==
$line_count){$neighbourhood_pos += length ($ref_line);}
                                else {$neighbourhood_pos -=
($len_to_pos - $position) +1;}#notice the +1 at the end, not a -1 as it would be a double -ve.
                        }

                        #split the neighbourhood into an array for
looping.
                        @neighbourhood = split (//, $neighbourhood);
```

```
						while (($neighbourhood_pos - $n) >= 0 &&
($neighbourhood_pos + $n) <= scalar (@neighbourhood))
						{
							if
($neighbourhood[$neighbourhood_pos - $n] =~ /[AT]/i &&
$neighbourhood[$neighbourhood_pos + $n] =~ /[CG]/i){if ($strand_designation eq
""){$strand_designation = "top";}}
							if
($neighbourhood[$neighbourhood_pos - $n] =~ /[CG]/i &&
$neighbourhood[$neighbourhood_pos + $n] =~ /[AT]/i){if ($strand_designation eq
""){$strand_designation = "bottom";}}

							last unless $strand_designation eq "";
							$n++;
						}

						$strand_designations{$chr_pos} =
$strand_designation;
					}

					if ($strand_designation eq "top")
					{
						if ($info =~ /0/){$genotype =
$ref_allele.$ref_allele;}#print "$rsID $ref_allele is complementary to $alt_allele and is on the top
strand\t$genotype\n"}
						else{$genotype = $alt_allele.$alt_allele;}
					}

					if ($strand_designation eq "bottom")
					{
						if ($info =~ /0/){$genotype =
$alt_allele.$alt_allele;}#print "$rsID $ref_allele is complementary to $alt_allele and is on the
bottom strand\t$genotype\n"}
						else{$genotype = $ref_allele.$ref_allele;}
					}

					#if ($info =~ /0/){$genotype = $ref_allele.$ref_allele;}
					if ($vcf_line[4] eq "\."){$strand_designation .= "\_WT";}

					#print "$sample $chromosome $position $rsID
$ref_allele \-\> $alt_allele $info $strand_designation $genotype\n";
				}
			}

			else #sample is heterozygous at this position.
			{
				#loop through the top stranded genotypes in %strands.
				for $top_genotype (sort keys %strands)
				{
					#loop through all the possible heterozygous genotypes
for the ref and alt allele dinucleotide (inc. reverse and rev. complement)
					#and if a match is found to the observed heterozygous
genotype then the Illumina formatted genotype will be the top stranded %strands key.
					if (grep {$mutation eq $_} sort
@{$strands{$top_genotype}}){$genotype = $top_genotype;}
				}
			}

			#push data into hash.
			${$marc_genotypes{$sample}{$rsID}}[0] = $genotype;
```

```
                        ${$marc_genotypes{$sample}{$rsID}}[1] = $position;

                        #unless ($hom_comp == 1){print "$sample $chromosome $position
$rsID $ref_allele \-\> $alt_allele $info $genotype\n\n";}
                }

                $sample_genotypes = 0;

                deer_in_the_headlights();
        }

}
##############################################################################
##################################
#End of enlighten sub routine
##############################################################################
##################################

##############################################################################
##################################
#sub routine deer_in_the_headlights.
#compares the data in the illumina hash with the data in Marc's hash
#calculate the total number of genotypes Illumina called, and see what percentage of them are
shared in Marc's data.
#
##############################################################################
##################################
sub deer_in_the_headlights
{
        $print_file = "/home/marc/Documents/work/exome analysis/Output/illumina
genotype_comparisons/$sample Illumina genotypes comparison.txt";
        open OUT, ">$print_file" or die "Could not open $print_file. Don't ask me why, nobody
tells me nothing.\n";

        for $rsID (sort keys %{$marc_genotypes{$sample}})
        {
                #signal used later to stop searching once a hash key is found.
                $end = 0;

                $print_position = ${$marc_genotypes{$sample}{$rsID}}[1];

                for $gene (sort keys %{$illumina_genotypes{$sample}})
                {
                        for $hash_key (sort keys %{$illumina_genotypes{$sample}{$gene}})
                        {
                                #cut the chr and pos from $rs.
                                $rs = $rsID;
                                $rs =~ s/.+\_//gi;

                                if ($hash_key =~ /$rsID$/)
                                {
                                        push (@{$table{$rsID}{$sample}},
${$illumina_genotypes{$sample}{$gene}{$hash_key}}[0],
${$marc_genotypes{$sample}{$rsID}}[0]);

                                        if
(${$illumina_genotypes{$sample}{$gene}{$hash_key}}[0] =~ /^[a-
z]{2}/i){$my_count++;$my_genotypes_total = scalar keys %{$marc_genotypes{$sample}};}

                                        #Not sure where this should go...
                                        #$illumina_genotypes_total += scalar keys
%{$illumina_genotypes{$sample}{$gene}};

                                        if
(${$illumina_genotypes{$sample}{$gene}{$hash_key}}[0] eq
${$marc_genotypes{$sample}{$rsID}}[0])
                                        {
                                                #the total number of genotypes and total
number of matches for all samples.
                                                $total_genotypes++;
                                                $total_matches++;

                                                #The number of genotypes and matches for
the sample
                                                $sample_genotypes++;
                                                $genotype_matches++;

                                                print OUT
"${$illumina_genotypes{$sample}{$gene}{$hash_key}}[0]\t${$marc_genotypes{$sample}{$rsID}}
[0]\t$strand_designations{$chr_pos}\tMatch\t\t$sample\t$gene\t$rsID\t$print_position\n";
                                        }

                                        else
                                        {
                                                $total_genotypes++;
                                                $sample_genotypes++;
                                                print OUT
"${$illumina_genotypes{$sample}{$gene}{$hash_key}}[0]\t${$marc_genotypes{$sample}{$rsID}}
[0]\t$strand_designations{$chr_pos}\tNot a match\t$sample\t$gene\t$rsID\n";

                                                print
"$sample\t$gene\t$rsID\t$strand_designation${$illumina_genotypes{$sample}{$gene}{$hash_k
ey}}[0]\t${$marc_genotypes{$sample}{$rsID}}[0]\n";
                                        }

                                        $end = 1;
                                        last;
                                }
                        }

                        if ($end == 1){last;}
                }
        }

        %{$marc_genotypes{$sample}} = ();

        if ($my_genotypes_total == 0){$percent_match = 0;}
        else{$percent_match = $genotype_matches / $my_count * 100;}

        push (@{$percent_matches{$sample}}, $percent_match, $genotype_matches,
$sample_genotypes);

        print "total Illumina genotypes matched: $genotype_matches ($percent_match %)\nMy
total = $my_count\n$depth_filtered with a depth lower than $dp_filter\n$q_filtered which are
classed as low quality\n$no_GT_filtered which had no genotype data called by the
GATK\n$no_comparison_possible which could not be compared\n\n";
        print OUT "total Illumina genotypes matched: $genotype_matches ($percent_match
%)\nMy total = $my_count\nIllumina total = $illumina_genotypes_total\n";

        close OUT;
```

```
}
################################################################################
####################################
#End of deer_in_the_headlights sub routine
################################################################################
####################################
```

Custom Script 6

```
#Given a list of amplicon regions within a targeted resequencing panel, extract reads from a
BAM file that map within the outer coordinates of an amplicon.
#Reads that are contaminating reads from another panel (whose amplicons don't overlap with
the exact amplicon regions from the first panel) that map outside
#of the amplicons will be excluded.
#Misaligned reads (even those that aren't contaminating) will also be excluded.
#Input requirements: A CSV file of amplicon coordinates, sorted by chr and left-most position,
with at least the following information per line:
#	chromosome ID, amplicon start and amplicon end coordinates.

#Subroutines:
#Read in sample BAM files.
#Read in amplicon coord's
#Use SAMtools view to isolate reads mapping within targeted amplicon regions. Print these
reads to a new SAM file and convert back to BAM (may need to resort based on chr coord).

use Term::ANSIColor;

#CSV files containing sequencing target coordinates for a particular NGS panel.
$amplicon_coords_file = "/home/marc/Documents/work/reseq analysis/non responder
project/Miscellaneous/DesignStudio amplicons.csv";
# $amplicon_coords_file = "/home/marc/Documents/work/reseq analysis/pilot
project/Miscellaneous/DesignStudio amplicons.csv";
$contig_bed_file = "/home/marc/Documents/work/reseq analysis/non responder
project/Miscellaneous/NR project amplicon contigs.bed";
# $contig_bed_file = "/home/marc/Documents/work/reseq analysis/pilot
project/Miscellaneous/pilot project amplicon contigs.bed";

$sam_dir = "/media/marc/Triceratops/Exomes/Pilot project output/WT (HD)
output/09_GATK_indelrealigner/isolated amplicons";
$alien_dir = "/media/marc/Triceratops/Exomes/Non responder
output/09_GATK_indelrealigner/alien amplicons";
$bam_dir = "/media/marc/Triceratops/Exomes/Pilot project output/WT (HD)
output/09_GATK_indelrealigner";
$pilot_isolated_dir = "/media/marc/Triceratops/Exomes/Non responder
output/09_GATK_indelrealigner/pilot isolated amplicons";
$pattern = "realigned\\.bam\$";

#Make necessary directories for output files.
unless (-d $sam_dir){
	print "mkdir \"$sam_dir\"\n";

	`mkdir "$sam_dir"`;
}

unless (-d $alien_dir){
	print "mkdir \"$alien_dir\"\n";

	`mkdir "$alien_dir"`;
}

unless (-d $pilot_isolated_dir){
	print "mkdir \"$pilot_isolated_dir\"\n";

	`mkdir "$pilot_isolated_dir"`;
}

#Read in CSV file containing sequencing target coordinates.
$coords = get_coords($amplicon_coords_file, $contig_bed_file);

#Get an array containing BAM filepaths. $pattern is used in a regex to search for particular files.
($filepaths, $bam_files) = read_dir($bam_dir, $pattern);

#Using target coord's, isolate target amplicons and print them to a new BAM file.
#This subroutine can also isolate off-target ('alien') reads and print those to a separate BAM.
isolate_amplicons($bam_files, $contig_bed_file, $pilot_isolated_dir, $coords);

#Read dir to get BAM files.
sub read_dir{
        #get the directory.
        my ($dir, $pattern, $files) = @_;

        my @files = @$files;

        #open the directory.
        opendir DIR, "$dir" or die "cannot open $dir!\n";

        #Read in all files from the directory (excluding the "." and ".." files which may be
present), and store the full filepaths in an array.
        my @filepaths = map{$dir."/".$_} grep {"$dir/$_" && !/^\.{1,2}$/} readdir DIR;

        closedir DIR;

        #foreach file and sub directory within the current directory.
        foreach my $filepath (sort {$a <=> $b} @filepaths){
                #If the current file matches a certain criteria ('-f' specifies a file).
                if (-f "$filepath" && $filepath =~ /$pattern/){
                        push @files, $filepath;
                }

                #if the directory contains sub directories, store them in the array and call this
sub routine on them.
                #the result is each sub directory within a directory will be searched before
moving onto the next directory;
                #e.g. a/1/i and a/1/ii will be searched before a/2/i, etc.
                if (-d $filepath){
                        my ($returned_array, $files) = read_dir($filepath, $pattern, \@files);

                        @files = @$files;

                        my @returned_array = @$returned_array;

                        push @filepaths, @returned_array;
                }

        }

        return (\@filepaths, \@files);
}

#Read in a CSV of target coordinates and store them in a hash table. Each target is stored in a
F and R direction, to allow for rapid searching downstream:
#Briefly, the start coord of a sequencing read (the left-most or right most coord for reads on the
F and R strand, respectively) will be used to search the target coordinate hash keys.
```

```
#If the read start coord is a key in the hash, then the read maps to a target amplicon. End
coords cannot be used for searching, as read length is variable between runs and so would not
always match targeted amplicon length.
#Therefore end coord's may not be present in the hash even though the start coord is present.
#This subroutine also prints target coord's to a BED file for use with samtools. The BED file
contains contigs of target amplicons, where overlapping targets are merged to avoid
redundancy when calling samtools.
sub get_coords{
	my ($amplicon_coords_file, $contig_bed_file) = @_;

	my %coords;

	open BED, ">$contig_bed_file" or die $!;

	open AMP, "$amplicon_coords_file" or die $!;

	while (my $line = <AMP>){
		chomp $line;

		next if $line =~ /^#/;

		my @fields = split (/,/, $line);

		my ($chr, $start_coord, $end_coord) = @fields;

		#Store the target coordinates in a hash (...of hashes of arrays...). For each left-
most coordinate, push the right-most coordinate into an array and vice versa.
		#The sequencing read's strand will determine which of these hash keys is used
for searching. e.g. for searchin
		unless (grep {$end_coord eq $_} sort {"\L$a" cmp "\L$b"}
@{$coords{"amplicons F"}{$chr}{$start_coord}}){push @{$coords{"amplicons
F"}{$chr}{$start_coord}}, $end_coord;}
		unless (grep {$start_coord eq $_} sort {"\L$a" cmp "\L$b"}
@{$coords{"amplicons R"}{$chr}{$end_coord}}){push @{$coords{"amplicons
R"}{$chr}{$end_coord}}, $start_coord;}
	}

	close AMP;

	#Create a hash of contigs read for printing to the BED file.
	for my $chr (sort {"\L$a" cmp "\L$b"} keys %{$coords{"amplicons F"}}){
		my @amp_starts = sort {$a <=> $b} keys %{$coords{"amplicons F"}{$chr}};
		my $contig_start = $amp_starts[0];
		my $contig_end = $contig_start;

		for my $amp_start (sort {$a <=> $b} keys %{$coords{"amplicons F"}{$chr}}){
			my @amp_ends = sort {$a <=> $b} @{$coords{"amplicons
F"}{$chr}{$amp_start}};

			#The biggest entry in @amp_ends.
			my $amp_end = $amp_ends[-1];

			#If the amplicon start is overlapping or immediately next to the current
end of the contig, merge the amplicon into the contig.
			if ($amp_start <= ($contig_end + 1)){
				if ($amp_end > $contig_end){$contig_end = $amp_end;}
				$coords{"contigs"}{$chr}{$contig_start} = $contig_end;
			}
```

```
                        #The amplicon has at least a 1 nucleotide gap between itself and the
contig end, so start a new contig.
                        else{
                                $contig_start = $amp_start;

                                #This should aways be true.
                                if ($amp_end > $contig_end){$contig_end = $amp_end;}

                                $coords{"contigs"}{$chr}{$contig_start} = $contig_end;
                        }
                }
        }

        for my $chr (sort {"\L$a" cmp "\L$b"} keys %{$coords{"contigs"}}){
                for my $contig_start (sort {$a <=> $b} keys %{$coords{"contigs"}{$chr}}){
                        my $contig_end = $coords{"contigs"}{$chr}{$contig_start};

                        print BED "$chr\t$contig_start\t$contig_end\n";
                }
        }

        close BED;

        return \%coords;
}


#For each BAM file, use samtools to extract reads to a new SAM file. Each mapped read is then
assessed and determined whether or not its start coordinate (left-most or right-most coord for F
and R strand reads, respectively) matches an amplicon start/end in the %coords hash.
#Reads mapping to target amplicons are isolated and printed in a new sample-specific SAM file.
A SAM file can also be created for printing off-target ('alien') reads if desired.
sub isolate_amplicons{
        my ($bam_files, $contig_bed_file, $sam_dir, $coords) = @_;

        my @bam_files = @$bam_files;

        my %coords = %{$coords};

        my %flag_data;

        foreach my $bam_file (sort {"\L$b" cmp "\L$a"} @bam_files){
                my %mapped_reads;

                #Create filenames for new SAM/BAM files to be printed.
                my $sam_file = $bam_file;
                $sam_file =~ s/indelrealigner\//indelrealigner\/isolated amplicons\//i;
                $sam_file =~ s/realigned\.bam$/isolated_amplicons\.sam/i;

                my $alien_file = $bam_file;
                $alien_file =~ s/indelrealigner\//indelrealigner\/alien amplicons\//i;
                $alien_file =~ s/realigned\.bam$/alien_amplicons\.sam/i;

                my $new_bam_file = $sam_file;
                $new_bam_file =~ s/\.sam//i;

                my $new_bam_file_sorted = $new_bam_file;
                $new_bam_file_sorted .= "\_sorted";

                my $new_alien_file = $alien_file;
                $new_alien_file =~ s/\.sam//i;

                my $new_alien_file_sorted = $new_alien_file;
                $new_alien_file_sorted .= "\_sorted";

                my $sample = $bam_file;

                $sample =~ s/^.+\///g;

                my @sample = split (/-/, $sample);

                my ($run, $coin, $tissue) = @sample;

                $tissue =~ s/\_.+$//;

                # next unless $tissue eq "e";
                # next unless $run =~ /NR3/i;
                # next unless $coin =~ /26004/;

                print color ("blue"), "Current sample: $run-$coin-$tissue.\n";

                #Get the unmapped reads and print to the new SAM file. Can also print these to
the alien file.
                print color ("green"), "Printing header and unmapped reads for ";
                print color ("blue"), "$run-$coin-$tissue\n";
                print color ("red"), "samtools view ";
                print color 'reset';
                print "-h -f4 \"$bam_file\" > \"$sam_file\"\n";
                `samtools view -h -f4 "$bam_file" > "$sam_file"`;
                # `samtools view -h -f4 "$bam_file" > "$alien_file"`;

                print color ("yellow"), "Complete!\n";
                print color 'reset';

                #Get the mapped reads that overlap with targeted coordinates (can get all
mapped reads instead, which is necessary for getting alien reads, but this takes longer).
                print color ("green"), "Gathering mapped reads overlapping amplicon regions in
";
                print color ("blue"), "$run-$coin-$tissue\n";
                print color ("red"), "samtools view ";
                print color 'reset';
                print "-h \"$bam_file\" -L \"$contig_bed_file\"\n";
                my @mapped_reads = `samtools view -F4 "$bam_file" -L "$contig_bed_file"`;
#This script is capable of looking at a given set of targets (specified in a BED file)...
                # my @mapped_reads = `samtools view -F4 "$bam_file"`;
                 #or the whole genome.

                print color ("yellow"), "Complete!\n";
                print color ("red"), "Isolating reads mapping to the designed amplicons.\n";

                #Loop over mapped reads and store each in a hash.
                foreach my $line (@mapped_reads){
                        chomp $line;
                        my @fields = split (/\s+/, $line);

                        my ($read_name, $flag, $chr, $read_start) = @fields;

                        my $sequence = $fields[9];

                        my $read_len = length $sequence;
```

```
            my $read_end = $read_start + ($read_len - 1);

            unless (exists $flag_data{$flag}){
                $flag_data = flagstat($flag, \%flag_data);

                %flag_data = %{$flag_data};
            }

            #If the read is on the reverse strand.
            if ($flag_data{$flag}{"SEQ is reverse complemented"} == 0){
                push @{$mapped_reads{"F strand
reads"}{$chr}{$read_start}{$read_end}}, $line;
            }

            #The read is on the F strand.
            else{
                push @{$mapped_reads{"R strand
reads"}{$chr}{$read_end}{$read_start}}, $line;
            }
        }

        print color ("yellow"), "Complete!\n";
        print color 'reset';

        open SAM, ">>$sam_file" or die $!;
        open ALIEN, ">>$alien_file" or die $!;

        #For each targeted amplicon starting coordinate, check if there are any mapped
reads that map to the same coordinate.
        #Such reads will be isolated by printing to a new SAM file.
        #By searching in a strand specific manner, you guarantee that matching reads
map within the target amplicon and not an adjacent one (plus it saves time).
        for my $chr (sort {"\L$a" cmp "\L$b"} keys %{$coords{"amplicons F"}}){
            for my $amplicon_start (sort {$a <=> $b} keys %{$coords{"amplicons
F"}{$chr}}){
                if (exists $mapped_reads{"F strand reads"}{$chr} && exists
$mapped_reads{"F strand reads"}{$chr}{$amplicon_start}){
                    for my $read_end (sort {$a <=> $b} keys
%{$mapped_reads{"F strand reads"}{$chr}{$amplicon_start}}){
                        foreach my $read (sort {"\L$a" cmp "\L$b"}
@{$mapped_reads{"F strand reads"}{$chr}{$amplicon_start}{$read_end}}){
                            print SAM "$read\n";
                        }
                    }
                }

                #This is not strictly necessary, but deletes on-target reads from
the hash. They need to be removed if you want to print only off-target reads later.
                delete $mapped_reads{"F strand
reads"}{$chr}{$amplicon_start};
            }
        }

        #Once reads mapping to target amplicons have been isolated, anything
remaining must be off-target.
        #Admittedly this method will include any reads that are misaligned, but should
definitely include amplicons that don't match target amplicons.
        for my $chr (sort {"\L$a" cmp "\L$b"} keys %{$mapped_reads{"F strand
reads"}}){
				for my $off_target_start (sort {$a <=> $b} keys %{$mapped_reads{"F
strand reads"}{$chr}}){
					for my $read_end (sort {$a <=> $b} keys %{$mapped_reads{"F
strand reads"}{$chr}{$off_target_start}}){
						foreach my $read (sort {"\L$a" cmp "\L$b"}
@{$mapped_reads{"F strand reads"}{$chr}{$off_target_start}{$read_end}}){
							print ALIEN "$read\n";
						}
					}
				}
			}

			#Do the same as above, but for the reverse strand.
			for my $chr (sort {"\L$a" cmp "\L$b"} keys %{$coords{"amplicons R"}}){
				for my $amplicon_end (sort {$a <=> $b} keys %{$coords{"amplicons
R"}{$chr}}){
					if (exists $mapped_reads{"R strand reads"}{$chr} && exists
$mapped_reads{"R strand reads"}{$chr}{$amplicon_end}){
						for my $read_start (sort {$a <=> $b} keys
%{$mapped_reads{"R strand reads"}{$chr}{$amplicon_end}}){
							foreach my $read (sort {"\L$a" cmp "\L$b"}
@{$mapped_reads{"R strand reads"}{$chr}{$amplicon_end}{$read_start}}){
								print SAM "$read\n";
							}
						}
					}

					delete $mapped_reads{"R strand
reads"}{$chr}{$amplicon_end};
				}
			}

			for my $chr (sort {"\L$a" cmp "\L$b"} keys %{$mapped_reads{"R strand
reads"}}){
				for my $off_target_end (sort {$a <=> $b} keys %{$mapped_reads{"R
strand reads"}{$chr}}){
					for my $read_start (sort {$a <=> $b} keys
%{$mapped_reads{"R strand reads"}{$chr}{$off_target_end}}){
						foreach my $read (sort {"\L$a" cmp "\L$b"}
@{$mapped_reads{"R strand reads"}{$chr}{$off_target_end}{$read_start}}){
							print ALIEN "$read\n";
						}
					}
				}
			}

			close SAM;
			close ALIEN;

			#Convert the new SAM file(s) to BAM file(s).
			print color ("green"), "Converting SAM file to BAM for sample ";
			print color ("blue"), "$run-$coin-$tissue\n";
			print color ("red"), "samtools view ";
			print color 'reset';
			print "-bSh \"$sam_file\" > \"$new_bam_file\"\n";
			`samtools view -bSh "$sam_file" > "$new_bam_file\.bam"`;
			# `samtools view -bSh "$sam_file" > "$new_alien_file\.bam"`;

			print color ("yellow"), "Conversion complete!\n";
			#Sort the BAM files.
			print color ("green"), "Sorting new SAM file for sample ";
			print color ("blue"), "$run-$coin-$tissue\n";
			print color ("red"), "samtools sort ";
			print color 'reset';
			print "\"$new_bam_file.bam\" \"$new_bam_file_sorted\"\n";
			`samtools sort "$new_bam_file.bam" "$new_bam_file_sorted"`;
			# `samtools sort "$new_bam_file.bam" "$new_alien_file_sorted"`;

			print color ("yellow"), "Sorting complete!\n\n";

			#Index the new sorted BAM file(s)
			print color ("green"), "Indexing BAM file for sample ";
			print color ("blue"), "$run-$coin-$tissue\n";
			print color ("red"), "samtools index ";
			print color 'reset';
			print "\"$new_bam_file_sorted.bam\"\n";
			`samtools index "$new_bam_file_sorted.bam"`;
			# `samtools index "$new_alien_file_sorted.bam"`;

			print color ("yellow"), "Complete!\n";

			#Remove the SAM file(s) and unsorted BAM file(s) as these are no longer
needed.
			print color ("green"), "Removing SAM file and unsorted BAM file for sample ";
			print color ("blue"), "$run-$coin-$tissue\n";
			print color ("red"), "rm";
			print color 'reset';
			print "\"$sam_file\"\n";

			`rm "$sam_file"`;
			`rm "$alien_file"`;

			print color ("red"), "rm";
			print color 'reset';
			print "\"$new_bam_file\"\n";

			`rm "$new_bam_file.bam"`;
			`rm "$new_alien_file.bam"`;

			print color ("bright_yellow"), "Isolation of amplicons is complete for sample ";
			print color ("blue"), "$run-$coin-$tissue";
			print color ("bright_yellow"), ".\n";
			print color 'reset';

			undef %mapped_reads;
		}
}

#Use the flag field to determine sequencing data about the read. See the flag description in the
SAMtools documentation for further info.
#In brief, uses a decimal integer to represent a binary string. Each digit in this string represents
a different aspect of the read (e.g. is mapped or not, is mapped to the F or R strand, is a PCR
duplicate, etc).
#This subroutine converts the decimal integer to its binary string (there is only 1 possible string
for each decimal) and stores the inferred information in a hash.
#If the particular flag has been encountered before, then the hash key should already exist and
this subroutine does not have to be performed again.
sub flagstat{
	my ($flag, $flag_data) = @_;
	my %flag_data = %{$flag_data};

	my $power_sum = 0;

	my $binary;

	for (my $i = 11; $i >= 0; $i--){
		my $power = 2 ** $i;

		if (($power_sum + $power) <= $flag){
			$power_sum += $power;
			$binary .= "1";
		}

		else {$binary .= "0";}
	}

	$binary = reverse $binary;

	my @binary = split (//, $binary);

	$flag_data{$flag}{"binary string"} = $binary;

	$flag_data{$flag}{"multiple template segments"} = $binary[0];
	$flag_data{$flag}{"segment properly aligned"} = $binary[1];
	$flag_data{$flag}{"segment unmapped"} = $binary[2];
	$flag_data{$flag}{"next segment unmapped"} = $binary[3];
	$flag_data{$flag}{"SEQ is reverse complemented"} = $binary[4];
	$flag_data{$flag}{"SEQ of next segment is reversed"} = $binary[5];
	$flag_data{$flag}{"first segment in template"} = $binary[6];
	$flag_data{$flag}{"last segment in template"} = $binary[7];
	$flag_data{$flag}{"secondary alignment"} = $binary[8];
	$flag_data{$flag}{"failed quality controls"} = $binary[9];
	$flag_data{$flag}{"PCR duplicate"} = $binary[10];
	$flag_data{$flag}{"supplementary alignment"} = $binary[11];

	return \%flag_data;
}
```