UPPSALA
UNIVERSITET

UPTEC X 19039
Examensarbete 30 hp
September 2019

Detection of artefacts in FFPE-sample sequence data

Hugo Swenson

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Detection of artefacts in FFPE-sample sequence data

Hugo Swenson

Next generation sequencing is increasingly used as a diagnostic tool in the clinical setting. This is driven by the vast increase in molecular targeted therapy, which requires detailed information on what genetic variants are present in patient samples. In the hospital setting, most cancer diagnostics are based on Formalin Fixed Paraffin Embedded (FFPE) samples. The FFPE routine is very beneficial for logistical purposes and for some histopathological analyses, but creates problems for molecular diagnostics based on DNA. These problems derive from sample immersion in formalin, which results in DNA fragmentation, interstrand DNA cross-linking and sequence artefacts due to hydrolytic deamination. Distinguishing such artefacts from true somatic variants can be challenging, thus affecting both research and clinical analyses.

In order to identify FFPE-artefacts from true variants in next generation sequencing data from FFPE samples, I developed the novel program FUSAC (FFPE tissue UMI based Sequence Artefact Classifier) for the facility Clinical Genomics in Uppsala. FUSAC utilizes Unique Molecular Identifiers (UMI's) to identify and group sequencing reads based on their molecule of origin. By using UMI's to collapse duplicate paired reads into consensus reads, FFPE-artefacts are classified through comparative analysis of the positive and negative strand sequences. My findings indicate that FUSAC can succesfully classify UMI-tagged next generation sequencing reads with FFPE-artefacts, from sequencing reads with true variants. FUSAC thus presents a novel approach in bioinformatic pipelines for studying FFPE-artefacts.

Handledare: Patrik Smeds, Claes Ladenvall
Ämnesgranskare: Adam Ameur
Examinator: Jan Andersson
ISSN: 1401-2138, UPTEC X 19039
Tryckt av: Uppsala

Sammanfattning

Det biotekniska fältet har under det senaste decenniet ändrats dramatiskt. Analys av DNA sker numer via alltmer sofistikerade digitala program. Likt hur binär kod via ettor och nollor kan bygga upp komplexa program, kan DNA representeras via ett set med bokstäver. Dessa bokstäver, de så kallade nukleotiderna adenin, tymin, cytosin och guanin (A, T, C, G) är grundstenarna i vårt DNA, och parar ihop med varandra vilket bildar en dubbelsträngad helix. Mer specifikt binder A till T, C till G [1]. För att veta hur en DNA-sekvens är uppbygd nyttjas så kallad sekvensering, vilket är en teknik där en maskin läser ordningen av nukleotiderna på en DNA-sträng. [2]. Denna data kan sedan användas till att hitta mätbara biologiska förteelser, så kallade biomarkörer, som är associerade med sjukdomar och sedan skapa läkemedel som specifikt interagerar med just dem. Denna typ av läkemedel kallas för ***Molecular Targeted Therapies*** (MTT) och används aktivt för att behandla cancer. Dessa läkemedel nyttjas som ett alternativ eller komplement till andra cancerbehandlingar, och kan bidra till effektivare behandling, mindre allvarliga biverkningar samt lägre kostnader inom sjukvården [3].

För att kunna sekvensera krävs DNA som startmaterial, vilket extraheras från prover av den organism som skall studeras. Ett prov som tas från en levande organism börjar dock snabbt att degradera då det lämnat värdkroppen. Detta är ett problem om man önskar studera dess inre struktur, eller i fallet med sekvensering vill studera DNA som inte är skadat. Att lagra prover i en frys kostar pegnar över tid, och det kan vara svårt att lagra i korrekt temperatur samt undvika köldskador. Att kontinuerligt ta färska prov är för många inte heller en möjlighet då det är dyrt, tidskrävande, och det dessutom finns etiska aspekter att ta hänsyn till. Om provet tas från patienter med allvarliga sjukdomar riskerar de dessutom att avlida. Lyckligtvis finns det en teknik kallad Formalin Fixated Parafin Embedded (FFPE) Tissue som eliminerar några av dessa problematiska aspekter. Vid skapandet av ett FFPE-lagrat prov sänks vävnadsprovet ned i en lösning kallad formalin vilket fixerar provet, en process som effektivt bevarar den inre strukturen. Vidare inkapslas provet sedan i ett paraffinblock, vilket gör att slutprodukten kan lagras i åratal i rumstemperatur utan att förstöras [4]. Med hänsyn till de förhållandesvis låga kostnaderna, dess aplikationer inom histologi, samt hur väletablerad metoden är används FFPE aktivt även idag. Majoriteten av världens lagrade genetiska material i så kallade ***biobankar***, är FFPE prov.

Fixering av vävnad via formalin leder dock till skador på DNA. Denna typ av skada är inte ett problem om provet skall studeras under mikroskåp likt vid histologi, men leder till större svårigheter då det skall sekvenseras. Det finns flera olika typer av skador som kan orsakas av formalin och lagring i FFPE block, såsom fragmentering av DNA, tvärbindning mellan de två strängarna samt hydrolytisk deaminering av cytosin. Mer specifikt deamineras cytosin till uracil eller tymin, vilket leder till felmatchningar, så kallade ***artefakter*** i form av T-C eller G-A bindingar. Artefakter likt dessa korrigeras normalt i kroppen av specifika enzym, men sådana saknas

helt i FFPE blocket där de istället kvarstår, och dessutom ökar ju längre blocket lagras. Problemet med artefakter kvarstår även vid amplifiering av DNA. Vid amplifiering delas dubbelsträngat DNA upp i två strängar. Dessa två strängar nyttjas sedan som mall för att bygga en matchande sträng, och således ökar mängden DNA-fragment med en faktor 2^n varje gång processen utförs. Detta innebär att för hydrolytiskt deaminerat DNA bildas det två DNA-molekyler som skiljer sig åt på en punkt, istället för två identiska DNA-molekyler. Då en DNA-molekyl under provberedning ofta amplifieras miljontals gånger är det således mycket svårt att skilja dessa artefakter från riktiga varianter [5]. Detta kan i sin tur, om datan ej hanteras med försiktighet, leda till felbedömningar rörande om en position är en verklig variant eller inte. Något som i värsta fall kan leda till felaktig diagnosticering av en position som potentiell biomarkör, och därmed även felaktig behandling av en patient. Alltså kan den omfattande mängd genetiskt material som finns tillgängligt i FFPE format, inte användas som grundmaterial för utveckling av MTT's utan att riskera felaktig klassifiering.

För att hantera dessa problem, skapades via detta examensarbete programmet FUSAC. FUSAC använder sig utav Unique Molecular Identifiers (UMI's), en sorts id-kod uppbyggd av korta nukleotidsekvenser. Dessa id-taggar fäster på ändarna hos ett DNA-fragment innan amplifiering sker, vilket alltså innebär att alla kopior av molekylen kommer ha samma UMI [6]. Programmet använder sig även utav en VCF-fil som guide, vilken listar genetiska positioner som behöver kontrolleras för förekomst av mutationer. För varje position i filen finner programmet alla fragment som överlappar med just den positionen, och dessa grupperas sedan baserat på sina UMI's. Med respekt till antalet möjliga UMI's och vart i genomet ett fragment passar in, kan man med stor sannolikhet anta att fragment som uppfyller båda kraven härstammar från samma molekyl. Programmet finner sedan de vanligaste nukleotiderna tillhörandes molekylens positiva och negativa sträng, och dessa kan sedan jämföras med varandra för att korrekt klassifiera positionen. Om konsensusnukleotiderna är identiska, men skiljer sig från ursprungsgenomet kan de antas vara en riktig variant, men om de ej är identiska och en utav dem är i form av en hydrolytisk deaminering kan de istället antas vara en FFPE artefakt [5].

Genom att utföra denna process för varje position i VCF-filen, kan sedan positioner identifieras som är (1) fria från både FFPE och somatiska varianter, (2) fria från FFPE men med stöd för somatiska varianter eller (3) fria från somatiska varianter men med stöd för FFPE artefakter. Således kan FUSAC bidra till att ge mer information till dem som studerar denna typ av data, vilket förväntas öka pålitligheten i genetisk diagnosticering utgående från FFPE behandlade vävnadsprov.

Contents

Glossary

BAI	BAM-file index.
BAM	Binary Alignment Map file.
CSV	Comma Separated Value.
Deque	Double ended queue.
DNA	Deoxyribonucleic Acid.
FF	Fresh Frozen tissue.
FFPE	Formalin Fixed Paraffin Embedded tissue.
FUSAC	FFPE-tissue UMI-based Sequence Artefact Classifier.
ICL	Interstrand DNA crosslinks.
M5C	5-methylcytosine.
MTT	Molecular Targeted Therapies.
NGS	Next Generation Sequencing.
PCR	Polymerase Chain Reaction.
RX	Raw Chromium barcode sequence.
SAM	Sequence Alignment Map file.
SNP	Single Nucleotide Polymorphism.
TDG	Thymine-DNA Glycosylase.
UDG	Uracil-DNA Glycosylase.
UMI	Unique Molecular Identifier .
VAF	Variant Allele Frequency.
VCF	Variant Call Format file.

Introduction

Cancer is a prevalent and serious health problem, recognised as the worlds third leading cause of death [7]. Many established treatment methods for cancer exist such as radiation therapy, surgery, chemotherapy, immunotherapy and more recently molecular targeted therapies (MTT) [8]. Based on the type of cancer, it's location, tumour stage and overall patient health. One, or a combination of treatment methods are traditionally selected with the goal of maximizing therapeutic efficacy [9]. However, information on genetic variation in the tumour is needed to guide treatment decisions for some of the more powerful modern treatment options. Despite their inherent advantages, the treatment methods may give rise to side effects which may result in serious adverse effects on the physical health and emotional well-being of the patient [10][11]. Even for patients with identical or similar cancer-types, the effects of drug therapy need not manifest in the same manner, but rather tends to show variability in it's effectiveness between individuals [12]. Such inconsistencies can derive from genetic variation between patients and between tumours. As an example, alterations in tumor suppressing genes such as BRCA1 and BRCA2 [13] are known to increase the rate of cancer formation (carcinogenesis) and spread of the disease in some but not all breast cancer tumors. Thus drugs targeting BRCA-mutated tumors are only likely to have an effect on tumors with genetic alterations in that gene. Unfortunately, consecutive use of a treatment often results in a portion of the tumor cells acquiring resistance towards the drug being used, a process which can lead to the drug having little to no therapeutic effect on the patient, whilst side effects are still present. Finally, such resistance can further lead to multi-drug resistance (MDR) of functionally unrelated anticancer drugs [14], severely complicating further treatment. Understanding the type of genetic changes and what effect they are likely to have on the tumor for the patient of interest, is therefore important when choosing an effective treatment.

Due to the rapid recent advancements in biotechnology, genomics and bioinformatics, we now have a wider understanding of how genetic variability may influence the effects of therapeutic treatment and drug-response. Precision medicine is an emerging approach for the prevention and treatment of disease. Based on the idea of tailoring the medical treatment for a patient based on their characteristics, precision medicine has the goal of generating a specific diagnosis and subsequently an effective individualized treatment plan [12]. Through identifying therapeutic or preventive treatment for the patient, the side-effects and costs of drugs unlikely to benefit the patient can be avoided. In addition to improving patient treatment response and overall quality of life [15], precision medicine also brings with it the hope of reducing overall health-care costs and time spent for the practitioner by avoiding unnecessary treatment and testing [3]. Increased understanding of cancer biology has allowed researchers to create novel molecular targeted therapies (MTT) specifically developed to act on molecular targets associated with carcinogenesis. These drugs typically bind to vital targets such as signaling molecules, growth factors or cell death (adoptosis) modulators [17]. As such, they interfere with specific regions with high

selectivity rather then act on all rapidly dividing cells as is the case in tradition chemotherapy [16]. While side effects from MTT's are of course still prevalent, their overall toxicity profile is more favorable then that of chemotherapy [17]. Furthermore, due to their high selectivity, they offer patients suffering from MDR new alternative treatment methods.

Formalin-fixed paraffin-embedded (FFPE) tissue is a major source for genetic material used in pathology, immunochemistry, histology and oncology. Due to it's convenience and long-term storage cost-effectiveness when compared to ultra low temperature storage of fresh-frozen (FF) tissue, it makes up a majority of diagnostic samples and cancer-tissue samples in bio-banks worldwide [18]. However, fixing tissue with formalin induces serious damage to DNA such as fragmentation, inter-strand cross-links and sequence artefacts in the form of the hydrolytic deamination of cytosine to uracil, or methylated cytosine to thymine [5]. These sequence artefacts, when not treated, will give rise to a mismatch on the deaminated position. As such, FFPE tissue sequence artefacts presents a challenge for researchers trying to identify true somatic variants to be used as target regions in MTT.

1.1 Motivation

FFPE samples present a possibly vast and varied source of genetic material to be used in precision medicine. However, false positive classification of FFPE artefacts as true variants in FFPE sample sequence data can lead to erroneous diagnosis and treatment of a patient. Despite advances in the biotechnical field, FFPE artefacts remain as a limiting factor in the viability of FFPE samples as starting material for bioinformatic analysis in precision medicine. Prior sequence libraries have attempted to reduce the impact of FFPE artefacts [19] [20] [21]. However, few methods have incorporated the use of UMI's to improve sensitivity and allow for lower frequency variants to be correctly classified.

The purpose of this thesis was the creation of a novel program FUSAC (FFPE tissue UMI-based Sequence Artefact Classifier) for detecting and classifying FFPE artefacts in FFPE sample UMI-tagged sequence data, as well as the evaluation of its performance. FUSAC was developed by me, in cooperation with the facility Clinical Genomics in Uppsala.

Background

2.1 Somatic variants

A somatic mutation is defined as a genetic alteration in the somatic cells of an individual, meaning it is not inherited from the parents, nor will it be passed down to eventual offspring [22]. Such mutations instead arise over time in an individual as a consequence of erroneous replication or damage to DNA through external factors [22]. Most such damage is typically repaired, however, a small fraction goes uncorrected and may be fixed as mutations [22]. The majority of somatic mutations do not have a noticeable effect on the health of the individual harboring them, however, some do bring with them alterations to important cellular functions.

While cancer is typically characterised by multiple accumulated genomic variants, certain mutations may induce a selective advantage for the cancer-cells. These so called ***driver mutations*** have been positively selected during the evolution of the cancer, and thus reside in regions highly correlated with carcinogenesis. Other genetic changes are called ***passenger mutations***. They are biologically inert somatic mutations that have been selected alongside the driver-mutation. [22]. Due to recent advancements in technology, the widespread use of NGS-methods has brought with them a major increase in our understanding of cancer-genomics. Through large-scale analysis of DNA originating from tumor samples, multiple genes affected by mutations present in different tumor types have been identified [23]. As most somatic mutations are biologically inert, these typically outnumber driver-mutations by a significant margin. The identification of somatic driver mutations from somatic passenger-mutations is therefore a major challenge in the fields of precision medicine and oncology [23].

2.2 Formalin Fixation

When fresh tissue is removed from it's host, it immediately begins to degrade. This presents a problem as the tissue needs to be in a lifelike state for adequate clinical diagnostics. Fixation is carried out to slow down or prevent this degenerative process from taking place, as well as preserving the cellular morphology of the tissue.

Formalin (10% neutral buffered formaldehyde solution) is the most common fixative in the world [24], and is made through dissolving formaldehyde gas in water. When a tissue-sample of interest is immersed in formalin, the formaldehyde penetrates the tissue quickly through diffusion to a cellular level. This gives rise to a hardening process, which effectively preserves the tissue in a lifelike state, allowing for storage and analysis of the tissue without the risk of immediate degradation [4]. However, formalin fixation also damages DNA in multiple ways. Formalin fixation causes fragmentation of DNA, resulting in shorter DNA templates which can prove problematic for certain sequencing techniques [25]. The fragmentation arises as a result of the low pH of formalin, which over time may lead to an increase in the rate of apurinic and apyrimidinic site formation which in turn leads to decomposition and fragmentation [26]. Furthermore, fixation by formalin may give rise to DNA-DNA inter-strand cross-links (ICL's). These lesions are highly cytotoxic [27] and have severe effects on translation and replication, thus reducing the amount of available template even further [28]. Even after fixation, long-term storage of FFPE blocks may induce DNA fragmentation due to environmental factors [29]. Finally, different labs may have different protocols for FFPE tissue sample processing, resulting in varying sample quality [30].

2.3 FFPE Artefacts

Immersion in formalin may also give rise to sequence artefacts as a result of hydrolytic deamination. In living organisms, spontaneous hydrolytic deamination of cytosine may occur resulting in a nucleotide change to uracil (figure 1 on the following page, A1). The sequence mismatch is then identified and corrected by the enzyme uracil-DNA glycosylase (UDG) (figure 1 on the next page, A2), which removes the uracil resulting in an abasic site [5]. The enzyme AP-endonuclease recognises this newly formed abasic site, and breaks the DNA phosphodiester bond at the site to allow for repair by DNA polymerase (figure 1 on the following page, A3) in the form of a new cytosine molecule. To complete the repair, DNA-ligase forms a new phosphodiester bond as to seal the site [31]. For 5-methyl-cytosine (m5c), the hydrolytic deamination will instead result in thymine (figure 1 on the next page, B1). The mismatch is then identified by thymine-DNA glycosylase (TDG) which removes the thymine (figure 1 on the following page, B2), allowing for repair as in the case with UDG (figure 1 on the next page, B3) [32]. These artefacts however, are more rare as the highest methylated regions of DNA only contain only around 1% of m5c [33]. This mechanism of DNA repair is obviously absent in FFPE samples,

and as a consequence any spontaneous hydrolytic deamination of DNA in the sample is left uncorrected.

Because these kind of artefacts arise on one of the two DNA-strands after fixation, identification of FFPE artefacts is made possible through comparison with the opposite strand. When performing NGS experiments, relatively large amounts of DNA are required and the preparatory steps most often involve PCR amplification to obtain sufficient material for sequencing. During amplification the heat-stable polymerase employed will elongate both strands separately [34], yielding two nearly identical DNA molecules differing only on the sites where hydrolytic deamination took place. Therefore, a cytosine that has been deaminated into a uracil will have one correct sequence from one strand with a C-G pairing at the deaminated site, but also an otherwise identical incorrect sequence with a U-A pairing at the deaminated site (figure 1, A4). The U-A pairing arises as a consequence of DNA-polymerase recognising uracil as thymine [35] and this pairing will then in subsequent rounds of the PCR replication give rise to a artefact as the adenine after denaturation in the next cycle will pair with a thymine during replication (figure 1, A5) [36].

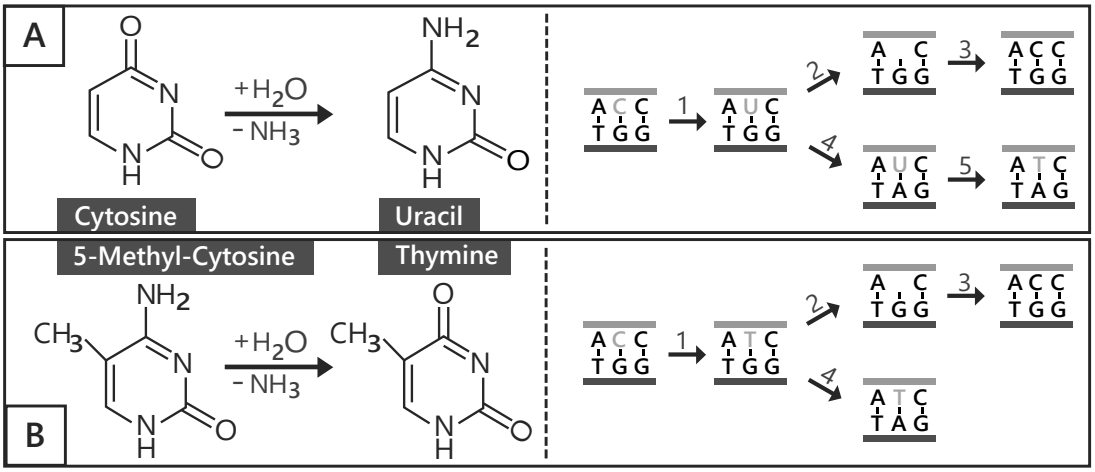


Figure 1: *Hydrolytic deamination of cytosine to uracil (A) and hydrolytic deamination of m5c to thymine (B) illustrated in step 1,2. The process seen in step 3 illustrates deamination and its subsequent repair in-vivo. Step 4 (and for A 5) illustrates the creation of a new fragment differing from the reference fragment by amplification, as a consequence of sequence artefacts.*

These two artefact types will from this point onward be referred to as C->T:G->A, where C->T (C to T) is representative of the artefact generated by a hydrolytic deamination on the positive strand, and G->A (G to A) is representative of the hydrolytic deamination on the negative strand. As such, FFPE artefacts may appear to be true somatic-variants in sequence data from FFPE samples, presenting a difficult diagnostic challenge when trying to identify true somatic variants that are key when selecting the appropriate treatment.

2.4 Unique Molecular Identifiers

Many modern sequencing-based techniques in genomics are dependent on amplification to increase the number of reads in the initial sequencing library, as inadequate levels of DNA will yield insufficient sequence data. However, bias in the amplification process often results in the increased prevalence of some sequences over others, and by extension a false representation of the original data set. Unique Molecular Identifiers or UMI's for short, are random oligonucleotide sequences which are attached onto fragmented DNA-molecules before the amplification process. By attaching these sequences to the end of a molecule of interest, each molecule gets an easily recognizable identity (figure 2). Meaning one can with high confidence identify subsequent copies of the source molecule, as these all share the same molecular identifier and align to the same position in the reference genome. This allows for the identification of the source molecule and it's subsequent copies, which yields a comprehensive view of the amplification-process efficiency as well as the identification of possible artefacts, given enough amplified reads to study [6]. Furthermore, by collapsing all reads with the same alignment-coordinates and UMI into one representative read, a more accurate representation of sample allele frequencies can be obtained [37].

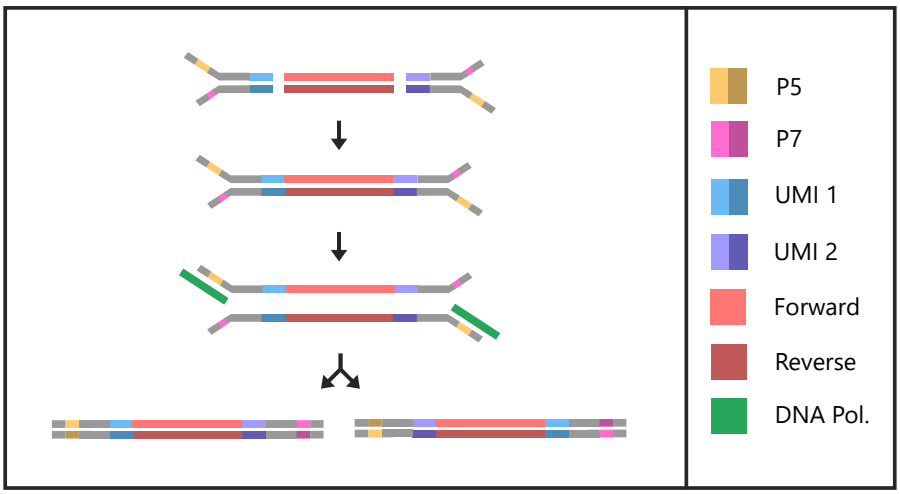


Figure 2: *Ligation of UMI-adapters to double-stranded DNA, followed by amplification. The resulting two molecules differ seen to their UMI-orientation, and can thus be identified*

2.5 Using UMI-tagged reads to identify FFPE artefacts

Artefacts introduced during PCR amplification can present a problem when attempting to separate true variant sequence reads from sequence reads with artefacts. One cannot easily determine the origin of the reads, nor if biased amplification has led to a misrepresentation of the original fragment distribution. For this purpose, UMI-tagged reads present a solution to this issue, as one can easily identify reads based on their UMI's in combination with their aligned position. Furthermore, through identifying if a read in a read pair is read 1 or read 2, and combining this information with the directionality that the read aligns with against the reference genome, one can determine it's strand of origin before amplification figure 3.

Figure 3: *To determine the strand of origin, each read is checked whether it is read 1 (light grey) or read 2 (dark grey), as well as if the read aligns to the reverse strand (left white arrow) or forward strand (right white arrow). Based on these two facts, the UMI is then rearranged to obtain the correct UMI sequence-tag order, and the read is classified as positive strand (Strand 1) or negative strand (Strand 2)*

In cases where no PCR-artefacts occur on the position where hydrolytic deamination has taken place, it should follow that because FFPE artefacts introduce a mismatch before amplification, all subsequent amplified reads from that strand should also carry this mismatch. Therefore, through collapsing all amplified reads belonging to the positive and negative strand separately, one would end up with two consensus reads representing the original strands used for amplification [38].

Once consensus sequences have been obtained for the positive and negative strand belonging to the UMI, one can inspect the sequence from each strand to identify mismatches and true variants. A site where hydrolytic deamination has taken place will typically show a C->T:G->A nucleotide mismatch, whereas true somatic variants display matching variant nucleotides. Thus, through comparing UMI-tagged reads, it is possible to identify FFPE artefacts in NGS fragments whenever information is available from both strands, regardless of amplification bias (figure 4 on the following page).

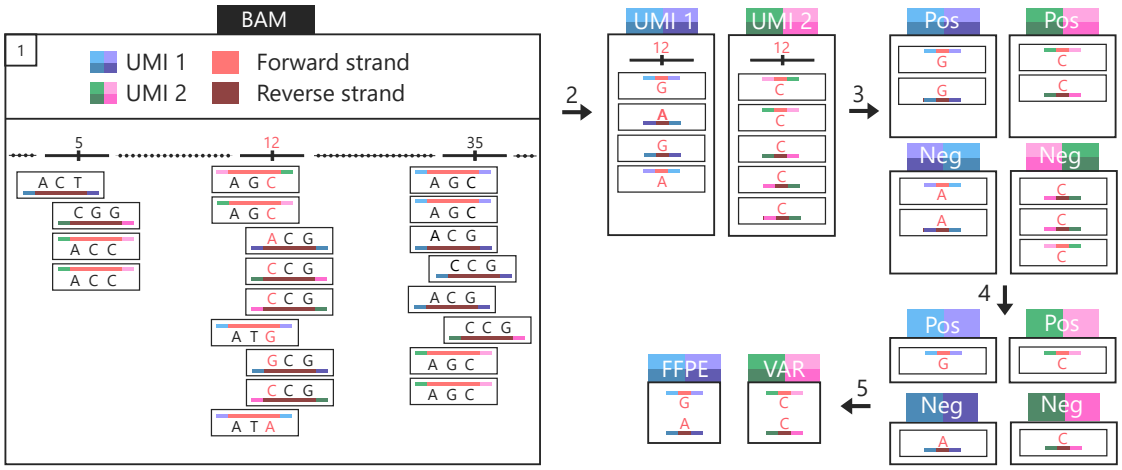


Figure 4: *Illustration showing classification of FFPE artefacts using collapsed UMI-reads. All nucleotoide bases shown are shown aligning to the forward strand, thus reverse reads are reverse complemented. (1) Using an identified variant-call (here shown in the color red), all reads aligning to the variant position are extracted from the Binary Alignment Map file (BAM) (2) All extracted reads are grouped by their UMI (3) The newly grouped reads are further split based on which strand they are duplicates of (Positive, Negative) (4) The reads are collapsed into consensus reads representative of all duplicates (5) The consensus nucleotide (red) on the variant-call position are compared with the consensus nucleotide on the same position on the opposite strand, and the variant-call is then classified*

Methods

3.1 Programming language and hardware

FUSAC was written in Python 3.7.2, using the additional module Pysam [39] and the library Pandas [40]. All programming and testing were performed on a laptop with a 4-core 2GHZ i7 processor and 8GB RAM. To generate BAM and Variant Call Format (VCF) files, an 8-node cluster with 128 cores was used. The output from FUSAC was further analyzed and visualized using custom R-scripts [41].

3.2 Data

3.2.1 Unfiltered data set

A data set originating from FFPE-stored human tissue data, sequenced with the NGS assay Illumina TSO-500 protocol was used to generate input for FUSAC. Each read in the data set had been primed using a 7nt long degenerate sequence (UMI) on each end to ensure identification of amplified molecules would be possible. The sequenced raw data was used as input for the pipeline tool bcbio [42] to generate a new BAM-file as well as a corresponding VCF-file. The toolkit fgbio [43] was then used to modify the query-names found in the newly generated BAM-file as per the instructions in section 3.5 on page 19. The unfiltered BAM-file was sorted and indexed using SAMtools [44], generating a corresponding BAI-file. Once this setup had been completed. For sake of clarity the generated BAM and VCF will from this point onward be referred to as the ***unfiltered data set***.

3.2.2 UMI-filtered data set

To minimize the impact of PCR sequencing artefacts, another data set was generated based on reads with unique UMI's in the unfiltered BAM-file that were observed at least three times. If a sequencing error is present in a read, it is impossible to know which nucleotide is correct using only 1 or 2 reads. However, if 3 or more reads are available, a decision can be made based on majority voting as to which nucleotide is correct. Based on this assumption, the unfiltered BAM-file was filtered using fgbio based on the condition that any unique UMI must have at least 3 corresponding reads, followed by the addition of UMI-tags to the query-name to fulfill the criteria listed in section 3.5 on page 19. Any UMI fitting this criteria then had their reads collapsed into a consensus read, whereas all UMI's not eligible were removed from the data set. The BAM-file was then sorted and indexed using samtools. Finally, the pipeline tool bcbio was used with the newly generated BAM-file as input to generate a new VCF-file. For sake of clarity the generated BAM and VCF will from this point onward be referred to as the ***UMI-filtered data set***.

3.2.3 Non-FFPE treated data set

To further evaluate the classifying capabilities of FUSAC, a non-FFPE-treated data set originating from a fresh human bone-marrow sample, sequenced with a library from Twist Biosciences and with UMI-adapters from IDT was used. Known hematologic variants in the sample measured to have a variant allele frequency (VAF) of 5-50%. The pipeline tool bcbio was used to generate another BAM-file as well as a corresponding VCF-file. The toolkit fgbio was then used to modify the query-names found in the newly generated BAM-file as per the instructions in section 3.5 on page 19. This data set will from this point onward for sake of clarity be referred to as the ***non-FFPE-treated data set***.

3.2.4 R-plots and CSV-files

FUSAC was used to generate Comma Separated Value (CSV) files for the output of the unfiltered UMI-filtered and non-FFPE-treated data set (see section 8 on page 48 for details). These files were then used in conjunction with custom R-scripts to generate data-plots. Furthermore, the output from BCFtools [44] was used for each of the three data sets in combination with custom R-scripts to generate data-plots.

3.3 FUSAC requirements and input

FUSAC is a novel python-based program for the statistical analysis of FFPE artefacts in UMI-tagged sequence data. It is largely based on the python module Pysam which is required to run the program along with the library pandas. FUSAC is written in Python 3.7.2, and thus it is recommended that it is run in a Python 3.x-environment. The required input-data to run an analysis is as follows

1. A Binary Alignment Map (BAM) file
2. An indexed BAM-file (BAI)
3. A Variant Call Format file (VCF)

Furthermore, the input BAM-file must be adjusted to fulfill three assumptions made by FUSAC. Ignoring this will result in erroneous or no output, and it is thus strongly advised to make sure the input-data fulfills these three conditions.

1. The input sequence data is UMI-tagged
2. The complete UMI is located either in the query-name or the RX (Raw Chromium barcode sequence) tag field of a read
3. If the UMI is located in the query-name, it is located as the last entry and separated from the rest of the query name through a symbol
   (a) Example_Name_With_Tag_At_End_UMI_ACTACTA+ACTACTA

The first assumption is necessary due to FUSAC's algorithm working in a classifying manner. To identify all reads stemming from a source molecule, a common identifier in the form of the UMI is vital for collapsing reads into a consensus sequence. The second and third assumption are both necessary to ensure that UMI-tagged data can be properly extracted from each read.

3.3.1 Singletons

When dealing with paired sequence data, one may encounter sequences occurring only once. These reads, also known as singletons, lack a mapped mate-strand (ie: only read 1 or read 2), and often stem from sequencing errors. As such, singletons are often subject to low-frequency variants caused by read-errors, and as consequence cannot typically be used for analysis. Furthermore, due to the frequency of rare variants in such reads, the removal of singletons have been shown to improve accuracy [46]. The FUSAC algorithm is able to identify singletons for every UMI-read aligning to a variant record (data line in a VCF containing information about a position in the genome) variant position in the input VCF-file, based on the assumption that such reads will lack a mate. Furthermore, as the inclusion of singletons into the variant-classification algorithm would likely decrease accuracy, FUSAC instead filters such reads out and stores their variant position data in the custom FORMAT-field ***SUMI*** (section 3.4) for interpretation by the user. In most cases, this will result in a string of values close to or equal to zero.

3.3.2 Single-strand reads

During the read-extraction process of FUSAC, some UMI's may only have reads amplified from one of the strands extracted. These reads, unlike singletons, have a mapped mate-strand and thus occur more then once in the data set. For sake of clarity, such reads will from this point onward be referred to as ***single-strand reads***. As FUSAC's algorithm requires strand-wise comparison to classify UMI-tagged read-pairs, single-strand reads cannot currently be used for classification. Instead, FUSAC records and writes the unique molecular counts for the variant and reference nucleotide to the custom field UMI to enable users to identify strand-bias based on the assumption that if strand bias were to occur for a given variant record, the unique molecular counts for either the positive or negative strand would be noticeably higher [47].

3.4 FUSAC output

The output from FUSAC is generated in the form of a modified VCF-file. The modified VCF is identical to the input VCF with the exception of a custom FILTER tag (***FFPE***) for classified FFPE variants, as well as two custom FORMAT fields called ***UMI*** (Unique Molecular Identifier counts) and SUMI (Singleton Unique Molecular Identifier counts). Both these fields contain the unique counts for the five variant-types (table 2 on the following page), the unique counts for the reference and variant nucleotides split by strand for read pairs where fragments from both

strands have been detected, and the unique counts for the reference and variant nucleotides on single strand reads belonging to the positive and negative strand.

Table 2: Variant-type classifications in the UMI and SUMI fields.

Variant classification	Definition
Reference nucleotide	Reference nucleotide
True variant	Variant nucleotide
FFPE artefact	C->T:G->A mismatch
Unknown	N
Deletion	-

3.4.1 Example Unmodified VCF-record

Example of a unmodified variant record in a VCF-file (table 3). The field INFO have been cut out for sake of clarity and are marked as "**...**". The ***FORMAT*** field values are displayed separately in table 4 for clarity.

Table 3: Example unmodified variant record.

Field	Value
CHROM	chr1
POS	4,367,323.00
ID	rs1490413
REF	G
ALT	A
QUAL	.
FILTER	FFPE
INFO	...
FORMAT	GT:AD:AF:DP

Table 4: Example unmodified VCF FORMAT-fields.

FORMAT field	Value
GT:AD:AF:DP:	0/1:571,632:0.527:1203:

3.4.2 Example modified VCF-record

Example of the FUSAC output (table 5 on the next page) based on the variant record as shown in table 3. The field ***INFO*** have been cut out for sake of clarity and is marked as **...**. The ***FORMAT***, ***UMI*** and ***SUMI*** fields are displayed in a separate table for clarity (table 6 on the next page). As can be seen, the ***FILTER*** field has been modified to display ***FFPE*** instead of

pass. This tells us that that the algorithm detected at least one read pair with a unique UMI and a C->T:G->A mismatch at the variant record variant-call position.

Table 5: Example FFPE classed variant record generated by FUSAC.

Field	Value
CHROM	chr1
POS	4,367,323.00
ID	rs1490413
REF	G
ALT	A
QUAL	.
FILTER	FFPE
INFO	...
FORMAT	GT:AD:AF:DP:UMI:SUMI

Table 6: Example FFPE classed FORMAT-fields, generated by FUSAC.

FORMAT field	Value
GT:AD:AF:DP:	0/1:571,632:0.527:1203:
UMI:	235;313;15;0;0;313;328;250;235;272;322;306;256:
SUMI:	0;0;0;0;0;0;0;0;0;0;0;0;0

Further investigation in the ***FORMAT*** field, more specifically the two fields UMI (table 7 on the following page) and ***SUMI*** generated by FUSAC, shows us that the unique molecular counts for reference nucleotide is 235, the counts for a true variant is 313 and the counts for FFPE artefacts is 15. The counts for a deletion or an unknown is deemed to be 0, the counts for the reference nucleotide (G) on the positive strand for paired reads as 313, and on the negative strand 328. Finally, the counts for the variant nucleotide (A) on the positive strand for paired reads is deemed to be 250, and on the negative strand 235. We can quickly see that this score is consistent with the 15 FFPE artefact classifications that were found by FUSAC. For single strand reads, the unique molecular counts for the reference nucleotide is 272 on the positive strand and 322 on the negative strand. Whereas the counts for the variant nucleotide is 306 on the positive strand, and 256 on the negative strand. Finally, the field ***SUMI*** indicates that no unique molecular counts for singletons were found at the given variant record.

Table 7: Example UMI-field values generated by FUSAC.

Value	Meaning
235;	Unique molecular counts for reference nucleotide
313;	Unique molecular counts for true variant
15;	Unique molecular counts for FFPE artefact
0;	Unique molecular counts for unknown
0;	Unique molecular counts for deletion
313;	Unique molecular counts for reference nucleotide on positive strand for paired reads
328;	Unique molecular counts for reference nucleotide on negative strand for paired reads
250;	Unique molecular counts for variant nucleotide on positive strand for paired reads
235;	Unique molecular counts for variant nucleotide on negative strand for paired reads
272;	Unique molecular counts for reference nucleotide on positive strand for single strand reads
322;	Unique molecular counts for reference nucleotide on negative strand for single strand reads
306;	Unique molecular counts for variant nucleotide on positive strand for single strand reads
256;	Unique molecular counts for variant nucleotide on negative strand for single strand reads

3.5 The FUSAC algorithm

The algorithm of FUSAC is based on six fundamental steps which are repeated for every variant record entry in the input VCF input file. The description provided here will cover the main points of the FUSAC algorithm. For a visual representation, please see figure 6 on page 21, or for a more detailed description, please refer to section 8 on page 48 and section 9 on page 73.

1. Using a VCF, BAM and BAI file as input, FUSAC calls a function which iterates through the variant records in the input VCF-file. From the variant record, the variant position is extracted and used to identify all reads in the BAM file aligning to it. FUSAC then copies the current variant record and proceeds to step 2.

2. In step 2, FUSAC extracts the UMI for every read identified as aligning to the variant position, and splits it into two half's (left and right). A function is then called which determines the string of origin for the read based on the directionality (forward, reverse) and the mate-type (mate 1, mate 2).

3. All reads are then grouped based on UMI, strand of origin (positive, negative) and read name. The newly generated list is then used as input for step 4.

4. The algorithm selects all reads belonging to unique UMI's for the positive and negative strand, and then iterates through all reads belonging to a UMI where it selects all reads with identical query-names. A function is then called which identifies the nucleotide on the variant position in the read. When all reads sharing the same UMI has been iterated through, the most prominent nucleotide is then chosen to represent the UMI. In other words collapsing all reads belonging to a UMI to generate a consensus nucleotide. If the reads were already collapsed before this step, the nucleotide for the variant position will simply be selected as it is the only entry.

5. Step 5 is carried out differently for fragments that have consensus nucleotides for the positive and negative strand, and for fragments that only have one (single-strand reads). For fragments that have been sequenced on both strands, the newly generated consensus nucleotide lists for the positive and negative strand are used as input. The algorithm then iterates through the positive and negative strand consensus nucleotide lists separately, selecting only entries with UMI's that appear in both. The molecules which such entries represent, are then classified as one of the five variant-types described in table 2 on page 16, based on comparative analysis of the positive and negative strand consensus nucleotides (figure 5 on the next page).

   (a) The ***Reference nucleotide*** classification will be selected if the consensus nucleotide for both the positive and negative strand is equal to the nucleotide in the reference genome for the variant position.

(b) The ***True variant*** classification is instead chosen if the consensus nucleotide for both the positive and negative strand is equal to the called variant in the variant record.

(c) The ***FFPE artefact*** classification is chosen, based on user input, as the positive and negative strand having a T-C or G-A mismatch, or if requested by the user as any form of mismatch.

(d) The classifications ***Unknown*** and ***Deletion*** are chosen only if either the consensus nucleotide of the positive and negative strand are equal to a unknown (N) or a deletion (-) respectively.

Each classification is counted and stored under it's UMI, complete with the consensus nucleotides used to make the classification. For single-strand reads, the consensus nucleotide is identified along with its strand of origin (positive, negative).

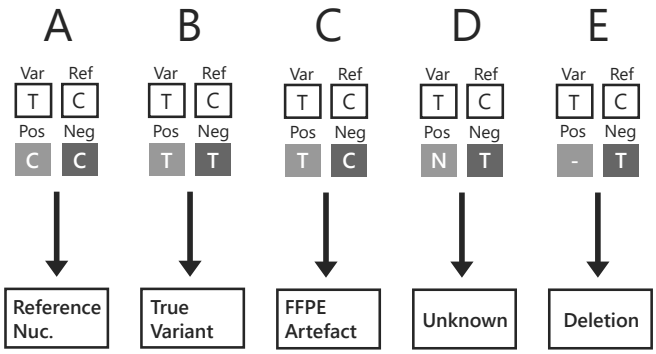


Figure 5: *Illustration for the variant type classification for unique consensus read-pairs. In scenario A, the consensus nucleotides for the positive and negative strand are identical, and equal to that of the reference genome nucleotide. Thus it is classified as* **Reference Nucleotide**. *In scenario B, the consensus nucleotides for the positive and negative strand are identical, and equal to that of the variant-call nucleotide. Thus it is classified as* **True Variant**. *In scenario C, the consensus nucleotides for the positive and negative strand are not identical. Furthermore, one of the consensus nucleotides is equal to the variant call nucleotide, and finally, the mismatch is of either the C-T or G-A type. Thus it is classified as a* **FFPE artefact**. *For the two remaining scenarios D and E, one of the consensus nucleotides is shown to be unknown or a deletion, and are therefore classified as such*

6. A function is called using the generated list and the copied variant record as input, generating a new modified variant record. All variant records showing counts for FFPE artefacts are flagged as ***FFPE*** in the ***FILTER*** field (section 3.4 on page 15). Including both variant records showing counts for only FFPE artefacts, and variant records showing counts for true variants and FFPE artefacts. After variant record classification has been carried out, the UMI and SUMI-fields are added (frefsec:Output). The algorithm then continues iterating through the VCF until the last variant record has been processed.

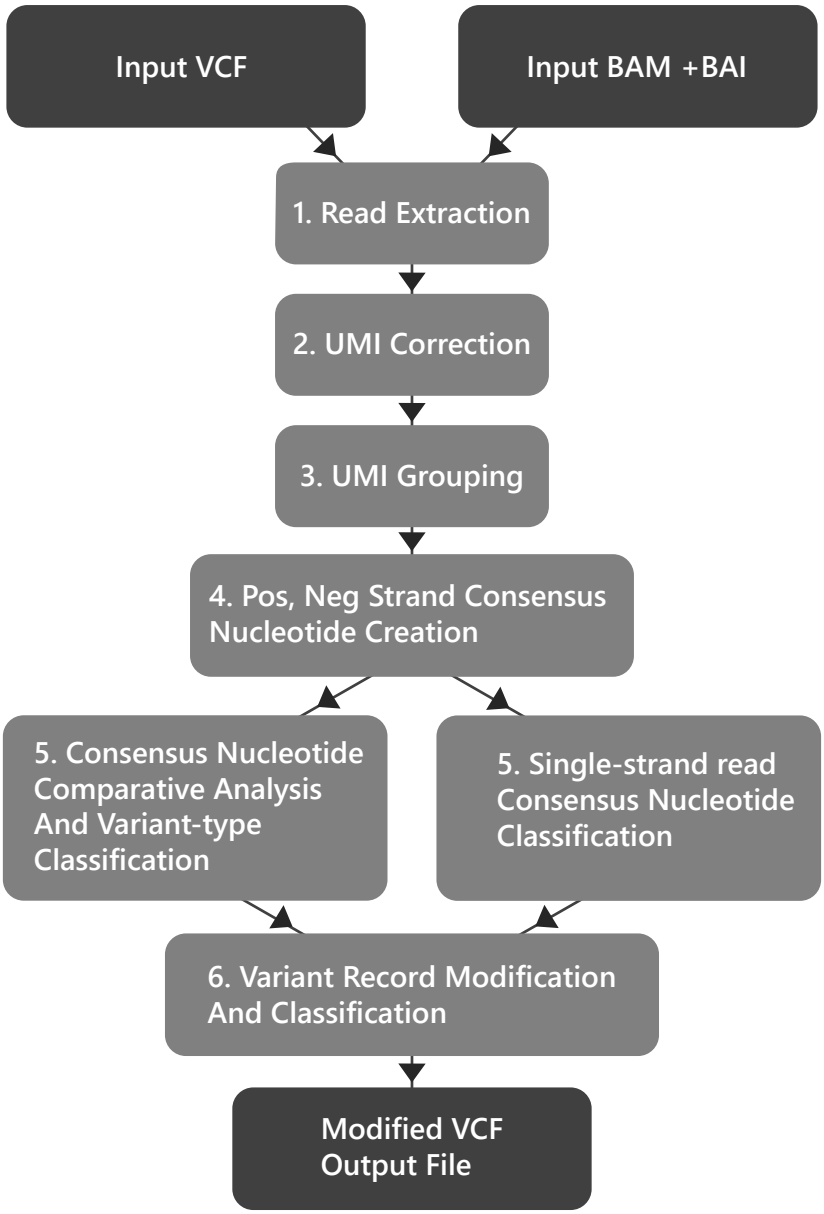


Figure 6: *Simplified flowchart for the six step FUSAC algorithm described in section 3.5 on page 19. (1) Using a VCF, BAM and BAI-file as input, all reads in the BAM-file aligning to a variant record variant position in the VCF are extracted (2) All extracted reads have their UMI's corrected (3) Reads are then grouped based on UMI, strand of origin and read-name (4) For both strands belonging to a UMI, all reads are collapsed to generate a consensus nucleotide for the variant position nucleotide (5) Based on if the UMI contains consensus nucleotides for both the positive and negative strand. Variant type classification based on comparative analysis of the consensus nucleotides, or identification of the consensus nucleotide (single strand reads) is carried out (6) The variant record used to extract the reads in step 1 is classified as* **FFPE** *or* **Pass** *based on the total data from step 4, and the fields UMI and SUMI are added to the* **Format** *field of the variant record. Once all variant records have been processed, the program stops and outputs a modified VCF-file*

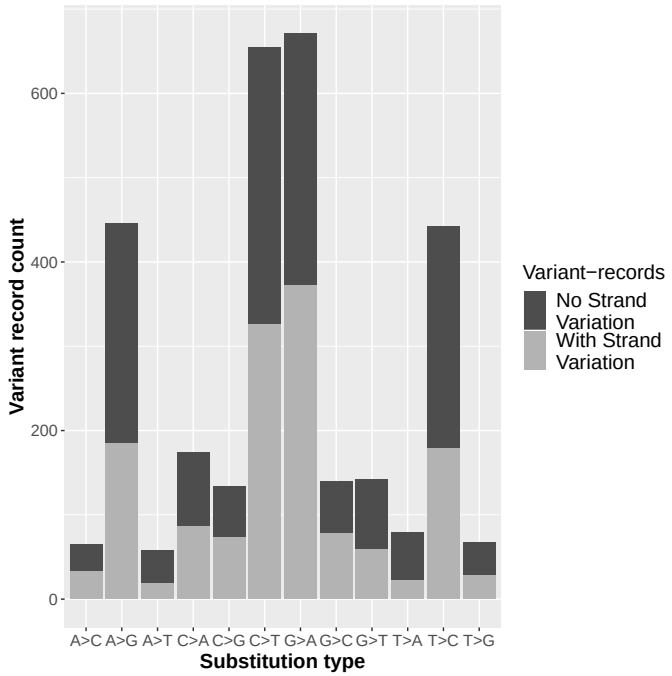
Results

4.1 Substitution type and counts

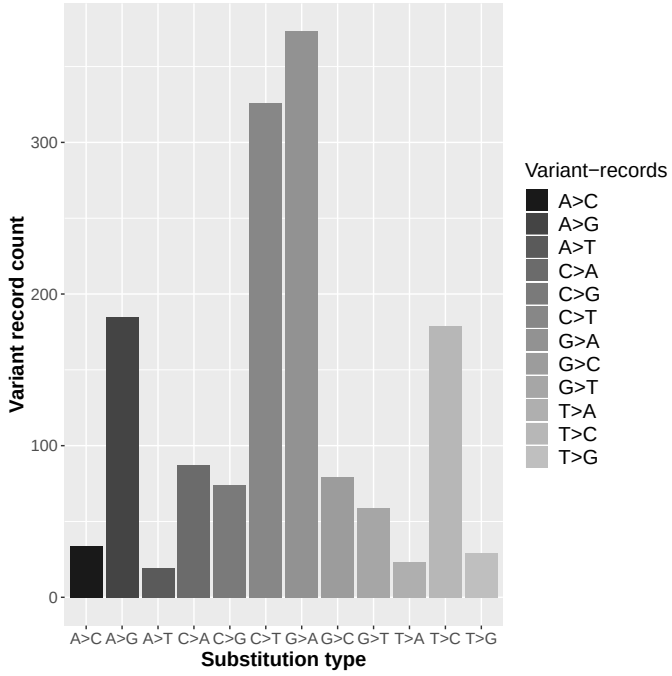
To evaluate the classifying algorithm of FUSAC, the unfiltered, UMI-filtered and non-FFPE-treated data sets were used as input for FUSAC with the setting ***all*** which flags any variant record with unique molecular counts for a substitution-type change, as well as the total number or variant records and their listed substitution type change. The output was used to generate bar-plots showing the counts of variant records for each respective substitution-type. As reference, bar plots were generated for the three data sets using output from BCFtools (section 3.2 on page 13) based on the ***Substitution Type*** field.

4.1.1 Unfiltered data set

For the unfiltered data set, the plots yielded by the output (figure 7 on the next page) of FUSAC and BCFtools showing the counts of variant records seen to listed substitution type, are identical (figure 7 on the following page). In total, 3,077.00 out of the total 3,779.00 variant records were identified by both FUSAC and BCFtools as having listed substitution-type changes. In both plots, G->A substitutions are consistently the most prevalent, followed closely by C->T and then A->G, T->C. The ratio of variant records with listed C->T, G->A substitutions to T->C, A->G substitutions being equal to 1.492. These results are consistent with previous findings for the substitution profile of FFPE treated data sets [48], suggesting that the binning process of FUSAC is accurate. Furthermore, the counts (1,467.00) for variant records showing signs of variation between the strands (figure 7 on the next page, left figure, blue), indicates that a majority (1,610.00) of the variant records reported in the VCF file have no variation between the strands, and are thus not affected by FFPE artefacts (figure 7 on the following page, left figure, red).



(a) Unfiltered - FUSAC



(b) Unfiltered - BCFtools

Figure 7: ***a:*** *Substitution-type and counts for variant records in the VCF (dark grey). Counts and substitution type for variant records showing signs of variation between the strands, suggesting some influence from FFPE artefacts (light grey), generated from the output of FUSAC* ***b:*** *Substitution-type and counts for variant records in the VCF, generated from the output of BCFtools*

4.1.2 UMI-filtered data set

For the UMI-filtered data set, the plots yielded by the output of FUSAC (figure 8) and BCFtools (Appendix C, figure 14 on page 74) showing the counts of variant records seen to listed substitution type, are identical (figure 8). The collapsing process resulted in a small reduction (231) of the total number of variant records listed with substitution type changes in the unfiltered data set. As well as a large reduction of counts (544) for variant records showing signs of variation between the strands. Suggesting that the unfiltered data set had a large number of low-frequency paired-reads with strand-variation.

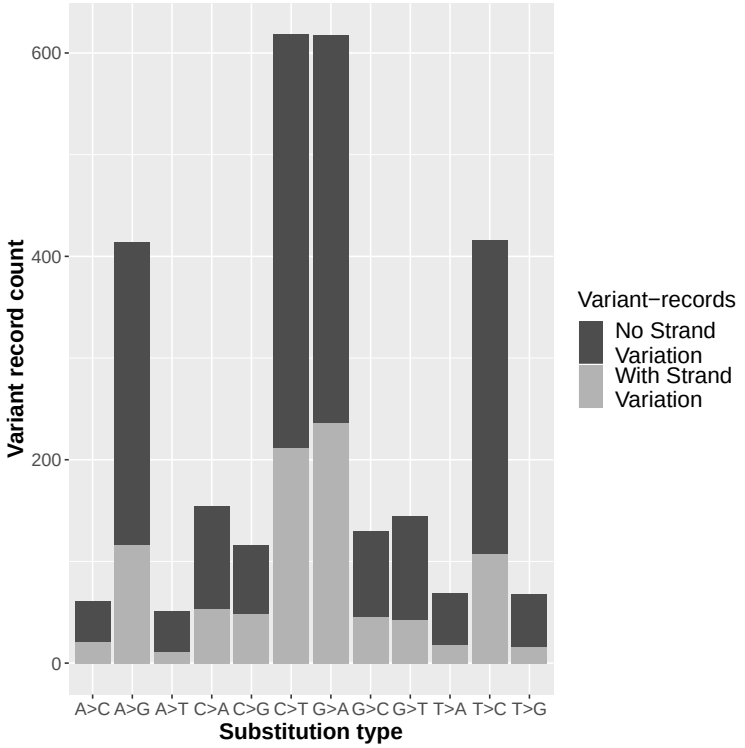


Figure 8: Substitution-type and counts for variant records in the VCF (dark-grey). Counts and substitution type for variant records showing signs of variation between the strands (light-grey), generated from the output of FUSAC

4.1.3 Non-FFPE-treated data set

The output from FUSAC (figure 9 on the following page and BCFtools (Appendix C, figure 15 on page 75) was identical, seen to the counts for variant record substitution type in the VCF-file from the Non-FFPE-treated data set. Transition-type substitutions show consistently higher an for that of transversion-type substitutions for both data sets, whilst showing similar counts to one another. The transition to transversion ratio (Ti/Tv) (equation (1) on the following page) for the data set was equal to 2.15, a ratio consistent with literature [49]. Furthermore, when comparing the G->A, C->T to A->G, T->C variant record counts ratio of the non-FFPE-treated data set (1.062), to that of the FFPE treated data sets (1.492 and 1.488 respectively), a clear increase in

the ratio of variant records with A->G, T->C substitutions could be observed.

$$Ti/Tv = \frac{C_{C->T} + C_{G->A} + C_{T->C} + C_{A->G}}{C_{A->C} + C_{A->T} + C_{C->A} + C_{C->G} + C_{G->C} + C_{G->T} + C_{T->A} + C_{T->G}} \tag{1}$$

Out of 1,598.00 variant records, 1,419.00 were identified by BCFtools and FUSAC as having a listed substitution type change. The counts (1,228.00) for variant records showing signs of variation between the strands (figure 9, left figure, blue), indicate that only a a small portion (199) of the variant records in this data set do not contain strand-variation. (figure 9, left figure, red).

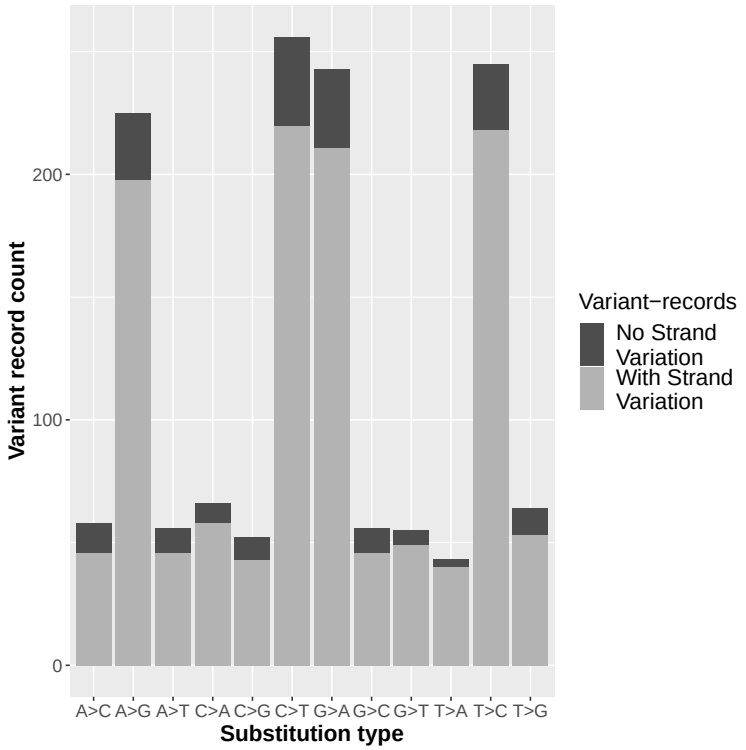


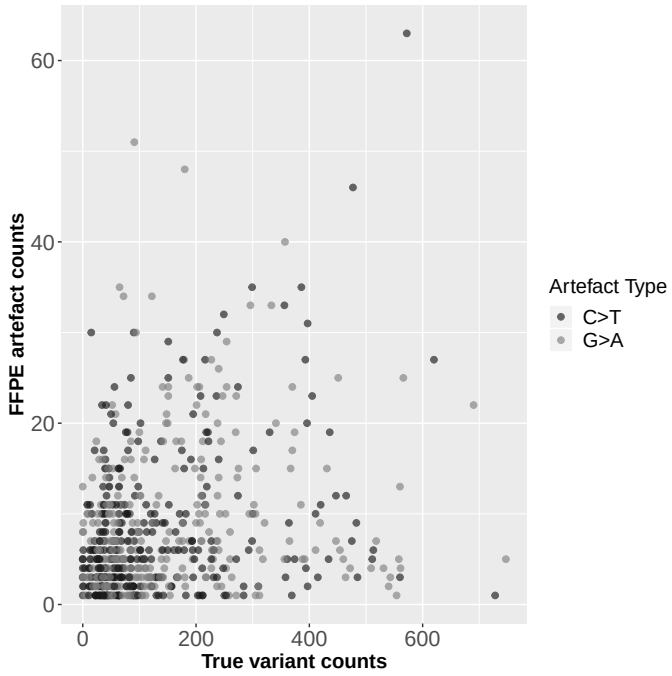
Figure 9: Substitution-type and counts for variant records in the VCF (dark-grey). Counts and substitution type for variant records showing signs of variation between the strands, suggesting some influence from FFPE artefacts (light-grey), generated from the output of FUSAC

4.2 Distribution of true variant and FFPE support

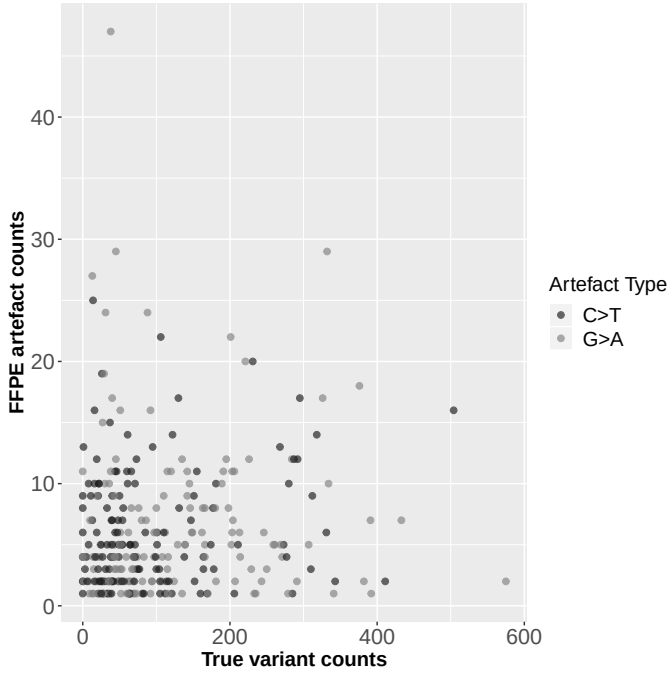
To study the classification algorithm of FUSAC, the unfiltered, UMI-filtered and non-FFPE-treated data sets were used as input for FUSAC with the default setting to only classify FFPE artefacts of the C->T, G->A type. For sake of clarity, the output generated by FUSAC will from this point onward be referred to as the ***FFPE Classified*** (***FC***) data sets (FC-unfiltered, FC-UMI-filtered, FC-non-FFPE-filtered).

To study the distribution of the unique molecular counts for true variants and FFPE artefacts for FFPE classed variant records, three scatter plots (figure 10 on page 28) were generated from the FC-unfiltered, FC-UMI-filtered and FC-non-FFPE-treated data sets respectively. For the FC-unfiltered and FC-UMI-filtered data sets, the plots show similar patterns with the majority of variant records having comparatively few reads supporting FFPE artefacts or true variants. The unique molecular counts for true variants were as expected on average significantly higher than for that of FFPE artefacts. Indicating that for both data sets, true variants are more common then FFPE artefacts.

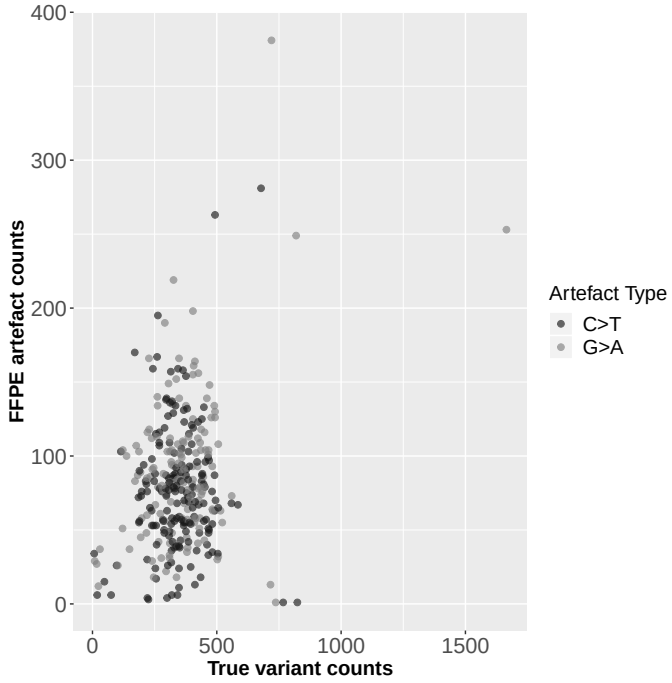
For the FC-non-FFPE-treated data set, the unique molecular counts for FFPE artefacts and for true variants were on average higher then for that of the FFPE treated FC-unfiltered and FC-UMI-filtered data sets, with the unique molecular counts for FFPE artefacts on average being higher in the FC-non-FFPE-treated data set.



(a) Unfiltered



(b) UMI-filtered



(c) Non-FFPE-Treated

Figure 10: *Scatter plot for the unfiltered (**a**), umi-filtered (**b**) and non-ffpe-treated (**c**) data sets, showing unique molecular counts for true variant vs. FFPE artefact classified by artefact-type. Each dot representing a separate variant record in the VCF-file, the x-axis representing the unique molecular counts for a true variant, and the y-axis representing the unique molecular counts for FFPE artefacts*

As the FC-non-FFPE-treated data set originates from fresh tissue, the unique molecular counts for FFPE artefacts were expected to be lower then for that of the FFPE treated data sets. To investigate this further, the distribution of UMI occurrences in the non-FFPE-treated data set were recovered (table 8) and compared with the distribution of UMI occurrences in the unfiltered data set (table 9 on the next page). The UMI-filtered data set was not included, on the basis that the lowest number of occurrences for any UMI is 3 or more. For the non-FFPE-treated data set, 85.88% of UMI's appear once, 12.22% appear twice, and 1.65% of appear thrice. Because PCR and sequencing errors can not be corrected in the majority of these reads, they can not be distinguished from FFPE artefacts. In this scenario FUSAC is expected to classify all read-pairs with unique UMI's of C->T, G->A type as FFPE artefacts, which is in line with the results shown in figure 10 on the preceding page. Furthermore, these results are also in line with the results shown in figure 9 on page 25, as the high number of low-frequency UMI's with read-errors would be flagged as variant records with strand-variation. The unfiltered data set instead showed a much larger (48.12%) frequency of UMI's occurring 3 times or more in the data set, indicating that while there is still the risk of read-errors being classified as FFPE, it is substantially smaller then for the non-FFPE-treated data set.

Table 8: Fraction of UMI-occurences for the non-FFPE-treated data set.

No. UMI occurences	Fraction in the data set
1	0.858794
2	0.122231
3	0.01646

All occurrences below a frequency of 1% are not listed.

As can be seen in table 8, the vast majority of UMI's in the non-ffpe-treated data set (85.8794%) occur once, while 12.2231% occur twice. Thus, if one wishes to minimize the influence of sequencing-errors, only 1.8975 % of the original data set can be used.

Table 9: Fraction of UMI-occurences for the unfiltered data set.

No. UMI occurences	Fraction in the data set
1	0.352784
2	0.165992
3	0.128637
4	0.101995
5	0.078513
6	0.057817
7	0.040762
8	0.027679
9	0.018115
10	0.011441

All occurences below a frequency of 1% not listed.

As can be seen in table 9, 35.2784% of UMI's in the unfiltered data set occur once, while 16.5992% occur twice. Thus, if one wishes to minimize the influence of sequencing-errors, 48.1224 % of the original data set can be used.

4.3 Classifications

4.3.1 Unfiltered data set

For the unfiltered data set, out of the total 30,828,724.00 reads, a total of 347,063.00 paired reads with unique UMI's were found to align to variant positions in the VCF-file. Out of these, 0.83% (2,897.00) were classified by the algorithm as FFPE artefacts (the consensus read pair containing a C->T:G->A mismatch at the variant position called by the variant record). Out of a total of 3,779.00 variant records, 12.62% (477) were deemed to have at least one FFPE classed paired read with a unique UMI align to their position. A total of 0.24% (9) of the 3,779.00 variant records were deemed to be true FFPE artefacts as they had no unique counts for a true variant, but unique counts for FFPE artefacts.

Table 10: FUSAC output read and variant record data for the unfiltered data set.

Counts	Meaning
30,828,724.00	Total number of reads in the BAM-file
347,063.00	Paired reads with unique UMI's aligning to variant positions in the VCF-file
5,376.00	Reads classified as FFPE artefacts
3,779.00	Total number of variant records in the VCF.file
3,302.00	variant records without FFPE artefacts
477	variant records classified with FFPE artefacts
468	variant records with unique molecular counts for both FFPE artefacts and true variants
9	Variant records with true FFPE artefacts

4.3.2 UMI-filtered data set

For the UMI-filtered data set, out of 14,565,219.00 reads, a total of 168,074.00 paired reads with unique UMI's were found to align to variant record variant positions in the VCF-file. Out of these, 1.09% (1,840.00) were classified by the algorithm as FFPE artefacts. Furthermore, out of a total of 3,395.00 variant records, 8.57%. (291) were deemed to have at least one FFPE classed paired read with a unique UMI align to their position. The collapsing and filtering process of the unfiltered BAM-file lead to 10.16% (384) of the variant records present in the VCF generated from the unfiltered data-file being excluded from analysis when compared to the VCF-file generated from the unfiltered data set. Furthermore, filtering process resulted in the loss of 52.86% (16,296,672.00) reads when compared to the unfiltered data set, and a loss of 61% (264,972.00) read-pairs with unique UMI's overlapping with the variant record positions in the VCF file. A total of 0.29% (10) of the 3,395.00 variant records were deemed to be true FFPE artefacts as they had no unique molecular counts for a true variant, but unique molecular counts for FFPE artefacts.

Table 11: FUSAC output read and and variant record data for the unfiltered data set.

Counts	Meaning
14,565,219.00	Total number of reads in the BAM-file
168,074.00	Paired reads with unique UMI's aligning to variant positions in the VCF-file
1,840.00	Reads classified as FFPE artefacts
3,395.00	Total number of variant records in the VCF.file
3,104.00	variant records without FFPE artefacts
291	variant records classified with FFPE artefacts
281	variant records with unique molecular counts for both FFPE artefacts and true variants
10	Variant records with true FFPE artefacts

4.3.3 Non-FFPE-Treated data set

For the Non-FFPE-treated data set, out of 21,120,584.00 reads, a total of 235,774.00 paired reads with unique UMI's were found to align to variant record variant positions in the VCF-file. Out of these, 11.01% (26,130.00) were classified by the algorithm as FFPE artefacts. Furthermore, out of a total of 1,597.00 variant records, 19.85%. (317) were deemed to have at least one FFPE classed paired read with a unique UMI align to their position. No true FFPE artefacts were found in the data set.

Table 12: FUSAC output read and variant record data for the non-FFPE-treated data set.

Counts	Meaning
21,120,584.00	Total number of reads in the BAM-file
235,774.00	Paired reads with unique UMI's aligning to variant positions in the VCF-file
26,130.00	Reads classified as FFPE artefacts
1,597.00	Total number of variant records in the VCF.file
1,280.00	variant records without FFPE artefacts
317	variant records classified with FFPE artefacts
317	variant records with unique molecular counts for both FFPE artefacts and true variants
0	Variant records with true FFPE artefacts

4.4 FFPE support in the UMI-filtered data set

Based on the assumption that true FFPE artefacts would show low support for true somatic variants, all variant records with higher unique molecular counts for FFPE artefacts then for true variants were selected in the FC-UMI-filtered data set. A total of 16 variant records were found with higher counts for an FFPE artefact at the variant record variant position than for a true variant. Out of these, 10 had no support for a true variant at the variant record variant position (table 13 on the following page, row 1-10). For a majority of the 10 variant records where the unique molecular counts for a true variant were shown to be equal to zero, the unique molecular counts for FFPE artefacts is low as well (table 13 on the next page, row 1-10), indicating poor coverage at these positions.

4.4.1 Variant allele frequency and coverage in the FC-UMI-filtered data set

To study the overall frequency of FFPE artefacts in the UMI-filtered data set, a line-plot (figure 11) was generated using the FFPE artefact VAF, calculated according to equation (2) on the next page for each variant record. The majority of variant records in the data set show a FFPE VAF in the 1-10% range (table 13 on the following page, consistent with the findings by Wong *et al.* (2014). Because positions with low coverage may have an erroneous representation of alleles, giving rise to misleading results through false VAF-values, I went on to investigate if some of the higher frequency variants suffered from lower coverage. To investigate the effects of coverage on VAF, the 16 variant records investigated in section 4.4 were extracted from the VAF data set (table 13 on the following page).

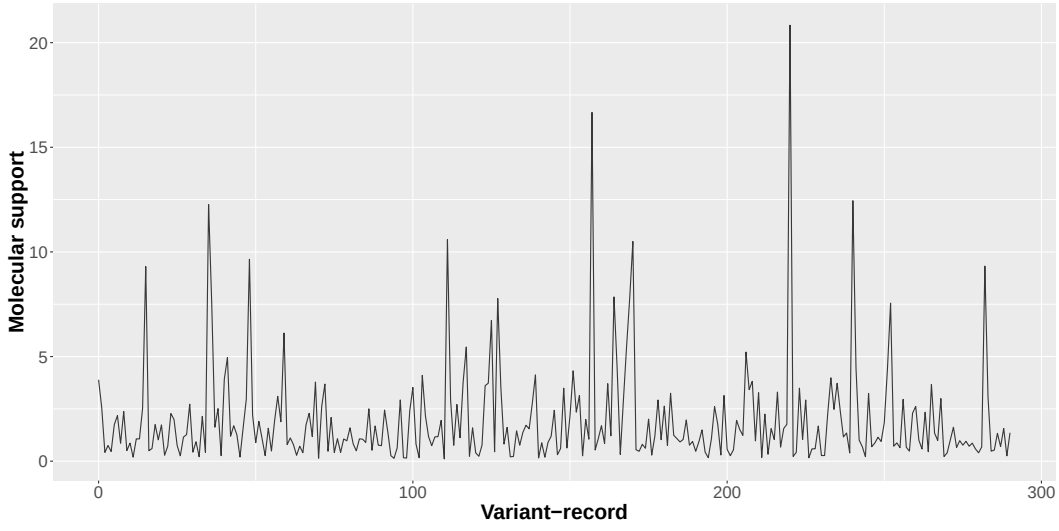


Figure 11: *Line-plot for the VAF of variant records showing unique molecular counts for FFPE artefacts in the FC-UMI-filtered data set*

$$VAF = \frac{C_{Allele}}{C_{tot}} \tag{2}$$

where:

VAF = Variant Allele Frequency
C_{Allele} = Coverage for allele at position
C_{tot} = Total coverage at position

One outlier (no. 221) seen in figure 11 on the previous page was shown to be a part of the 16 variant records with higher unique molecular counts for FFPE artefacts than for true variants (table 13, row 14). All variant records, with the exception of 3 (table 13, row 11, row 14, row 15) show a VAF within the 1-10% range, consistent with figure 11 on the preceding page and the findings of Wong *et al.* (2014). Among the variant records with higher counts for FFPE artefacts then for true variants, all had relatively good (>50) unique UMI coverage, thus their reported VAF's are likely to be correct. No clear relationship between VAF and coverage could be observed, but it is likely that variant records with low coverage would have showed a larger variance in the VAF for FFPE artefacts.

Table 13: VAF and Coverage for variant records where the unique molecular counts for FFPE artefacts were bigger then for that of a true variant.

Var. rec. no.	FFPE Artefact counts	True Variant counts	VAF	Coverage
49	11	0	9.6491228	114
60	9	0	6.1224490	147
169	8	0	5.4794521	146
152	6	0	4.3165468	139
284	4	0	3.0769231	130
140	4	0	4.1237113	97
55	4	0	1.5748031	254
44	2	0	1.6949153	118
129	2	0	3.5087719	57
56	1	0	0.4854369	206
36	13	1	12.2641509	106
165	27	13	7.8488372	344
205	2	1	1.5151515	132
221	25	14	20.8333333	120
112	47	38	10.5855856	444
166	10	8	4.4247788	226

Variant records ordered with respect to FFPE/True Variant counts ratio.

Discussion

Precision medicine relies on the ability to identify genetic markers for therapeutic drugs, thus removal of sequence artifacts is of great importance when designing a treatment plan. Failure in the removal of sequence artefacts, especially if the amount of starting template is limited, will result in an increase of the relative frequency of such mutations. The current methodology at Clinical Genomics in Uppsala for assessment of a variant allele as a biomarker, is to have experts determine if the position fulfills a set of criteria. Thus, FFPE artefacts introduces a time-consuming challenge of correctly identifying the false-positive variant calls from true somatic variants, a process which delays, and in the worst case may lead to erroneous treatment of a patient.

5.1 Artefact type and support

The FFPE treated data sets in figure 7 on page 23 show a similar distribution in the counts for variant record substitution types, with G->A and C->T being the most prevalent, followed by A->G and T->C. These results are consistent with the findings of Kim *et al.* (2017), and correlate well with the claim by Wong *et al.* (2014) that C->T:G->A mismatches make up an important fraction of variation in FFPE tissue samples. The output generated by FUSAC, showing the counts and substitution type for variant records with strand-variation, shows potential for indicating what variant records may be of interest when searching for true variants. In figure 7 on page 23, as all variant records (red) not covered by FUSAC (blue) have shown no strand-variation on a molecular level for all paired-reads aligning to their position, they can be regarded as true variants. The FUSAC output from the non-FFPE-treated data set further show what applications UMI-based variant-calling can have in a clinical or research based setting. The high ratio of variant records showing strand variation to total number of variant records for all substitution types in the data set, indicates that most variant records harbor true strand variation. However, in conjunction with the generated bar-plot for the frequency of UMI-occurrence in the data set, one can draw the conclusion that the large number of variant records showing strand-variation likely is due to a large number of PCR-artefacts or read-errors as the vast majority of read-pairs in the data set only occurred once. These findings seem to indicate that FUSAC is able to identify and correctly classify strand-variation in variant records based on UMI-tagged read-pairs. Furthermore, the output from FUSAC can be of use when attempting to identify what variant records are of interest when attempting to identify true variants in sequence data.

5.2 Classification

The results generated by FUSAC in section 4.3.1 on page 30 and section 4.3.2 on page 31 show that the program correctly identifies genomic variation in UMI-tagged data, producing a detailed output regarding the characteristics of each variant record supplemented by the input VCF. It should be noted that the data used to generate the unfiltered and UMI-filtered data set were not filtered for low coverage before being used as input for FUSAC. Realistically, this will likely not be the case for practical use, because low coverage variant records seen in section 4.4 on page 33 would have been flagged in the current clinical setup.

Despite the larger size of the unfiltered data set when compared to the UMI-filtered data set, the UMI-filtered data set was shown to contain more ***pure*** (variant records with no unique molecular counts for a true variant) FFPE artefacts (10 vs. 9). To elaborate on this, one must take into consideration that the current classification methodology of FUSAC is to classify any variant record as ***FFPE*** if it shows unique molecular counts for even one FFPE artefact. The filtering and collapsing lead to the loss of 16,552,612.00 reads. Thus, it is likely that many low-coverage UMI's were removed from the data set, leading to the increase in frequency of variant records showing no unique molecular counts for true variants.

When classifying the unfiltered data set through FUSAC, a total of 12.62% of the variant records listed in the VCF-file were deemed to contain at least one FFPE artefact. Likewise, for the UMI-filtered data set a total of 8.57% of the variant records were deemed to have support for at least one FFPE artefact. Inversely, this means that 87.38% and 91.43% of the variant records can be classified as true variants (containing no unique molecular counts for FFPE artefacts). Furthermore, by studying the output for the remaining FFPE classed variant records in section 4.4 on page 33, true FFPE artefacts (variant records with no coverage for true variants) could be distinguished alongside detailed information for each individual variant record. Thus, FUSAC shows promise as a tool for identifying variant records containing true variants that are free from FFPE artefacts in a data set, allowing clinicians/researchers to safely asses the potential of variants as molecular targets for therapeutic drugs. Saving time and effort which can instead be focused on diagnosis and treatment of the patient.

5.3 Issues and limitations

The algorithm employed by FUSAC, is not without its limitations. The following section aims to discuss and highlight the issues or limitations encountered when dealing with data sets of varying quality or size.

5.3.1 UMI occurrence

The approach employed by FUSAC requires comparative analysis between nucleotide pairs belonging to the positive and negative strand respectively. As such, if each nucleotide call is based on a single read, or low number of reads, the software will not be able to distinguish a PCR-artefact from an FFPE artefact. Filtering the data to always have at least three entries for every UMI is one way of dealing with this issue, as one can utilize majority-voting to select against such low-frequency variants. However, doing so comes at the price of reduced coverage. To illustrate this further, in the case described in section 4.3.2 on page 31, for the non-FFPE-treated data set, over 85% of all UMI's were found to occur only once, and only 1.65% of all UMI's occurred three or more times. As a consequence of this, when collapsing that datset using at least 3 observations for each UMI, FUSAC would only be able to use 1.65% of the original non-FFPE-treated data set.

5.3.2 Coverage

Because FUSAC treats each variant record as a separate entity, variant records with poor coverage can influence the overall profile presented to a researcher. Poor coverage not only influences statistics such as VAF, but may also lead to false FFPE classifications by the algorithm due to the absence of true variants at a position as a consequence of insufficient coverage. One way of handling this issue is to simply remove records with low coverage. However, this approach naturally comes at the possible risk of removing low coverage variant records containing true somatic variants of interest. As a whole, the influence of coverage and UMI-occurrence with respect to classification may present challenges for samples with low amounts of starting-material, or for samples of poor quality. Inversely, higher sequencing depth will of course improve these numbers, but the effects of coverage is beyond the scope of the current work.

5.3.3 Performance and classification

The ***FFPE*** filter-classification process is currently limited to classifying any variant record showing unique molecular counts for the prescence of FFPE artefacts at the variant position. This detailed approach cannot distinguish between variant records with good coverage and a high FFPE VAF, from variant records with poor coverage or low VAF. It also does not judge if a variant with few counts supporting an FFPE artefact and many supporting a true variant is a true variant or not, that is left to the user. A more stringent filtering may lead to the loss of information and that may possibly have a negative impact on the process of identifying true

somatic variants.

5.4 Future applications and improvements

During the development of FUSAC, the biggest limiting factor was time. There exists many interesting and possible extensions for improving the functionality and performance of FUSAC, which will be discussed in detail in this section.

5.4.1 Utilization of Single-strand read data

In the extraction process of FUSAC, a UMI may sometimes only retrieve reads amplified from one of the strands of origin (positive or negative). As the algorithm employed by FUSAC is based on consensus nucleotide comparison between the positive and negative strand of a UMI-tagged read-pair, such reads cannot currently be used for variant-type classification. One current approach for handling such single-strand reads (section 3.3.2 on page 15) is to list the single-strand read support for the variant and reference nucleotide on the positive and negative strand respectively. Based on the assumption that a region that has experienced strand bias, will likely have a significant difference in the unique molecular counts for either the positive or negative strand. An automated process for flagging variant records showing signs of strand bias, could potentially improve model accuracy. Guo yi *et al.* (2012) found in their study that regions with high strand bias also indicate a high false-positive rate for single nucleotide polymorphisms (SNP's). The presence of false-positive SNP's could potentially lead to false-positive FFPE or true variant-calls by FUSAC. And by extension erroneous diagnosis or treatment due to a false positive variant-call being identified as a potential molecular target. As such, information regarding strand-bias could potentially be used to filter out or flag variant records showing high strand bias. Such a process would increase transparency in the data set, allowing users to identify and lessen the influence of regions containing false-positive SNP's.

5.4.2 Classification algorithm improvements

At its core, FUSAC is attempting to perform a binary classification (FFPE or PASS) based on given input data in the form of the UMI and SUMI fields for the variant record of interest. With respect to this, the implementation of supervised binary classifiers such as artificial neural networks is a topic of great interest for improving the classifying step of FUSAC. As was covered in section 2.3 on page 8, the quality of FFPE tissue sequence data is influenced by many factors such as the protocols used for FFPE fixation, period of storage and the environment, as well as the protocol used for extraction. With respect to this, a learning algorithm such as the one found in back-propagation artificial neural networks could theoretically be trained on data from similar origins to improve classification [52]. Coverage and VAF are factors that have previously been shown to correlate with the presence of FFPE artefacts [5], and could potentially through training be used to build more accurate models for classification with respect to the overall quality of the data set.

To further evaluate the utility of FUSAC and investigate if threshold levels for coverage and VAF can be set by default for classification. Tests against a wider range of sequence depths and different NGS chemistries are required. In particular, tests on data sets with a known number of FFPE artefacts are required to evaluate the performance of FUSAC and determine the feasibility of VAF/Coverage ratio as a metric for classification of FFPE artefacts. Unfortunately, at the time of the study, I was not able to identify any data sets meeting these criteria. An alternative approach would be to test the performance through the use of UMI-tagged synthetic reads. As such, further performance evaluation and optimization of the classification algorithm for FUSAC is likely warranted before routine clinical use.

5.4.3 Applications in clinical settings and research

Despite the issues brought up in section 5.3 on page 37, FUSAC has clear applications in bioinformatic pipelines as it outputs detailed information regarding the unique counts for the variant type as well as the unique counts for the reference-base and called variant for both paired and single-reads. The comprehensive overview given through FUSAC's output can thus be used to gain a better understanding of the distribution and molecular structure of reads aligning to the variant-call. Furthermore, while FUSAC's ability to classify variant records is limited to any variant record showing unique molecular counts for FFPE artefacts in it's current state, the program has a clear use as a fast and reliable way to quickly identify variant records without unique counts for FFPE artefacts. The unfiltered and UMI-filtered data sets had 12.62% and 8.57% of their variant records flagged as affected by FFPE. This effectively means that 87.38% of the original data set, or 91.43% of the UMI-filtered data set can be identified as true variants. This information would allow researchers to safely asses the potential of variants within these variant records as molecular targets without the risk for false-positive classification of FFPE artefacts.

Conclusion

FUSAC, as has been presented in this thesis, is a novel computational tool specialized in identifying FFPE artefacts in UMI-tagged data. The results from FUSAC are promising, though further tests against a reference or synthetic read reference data set will be needed before it is used in a clinical setting. FUSAC has a clear use for the purpose of analyzing FFPE tissue samples, and as a novel software without any real alternatives in regards to it's approach, it fills an important role in the still greatly unexplored realm of FFPE artefact classification.

Special thanks and credits

I would like to thank Claes Ladenvall & Patrik Smeds at Clinical Genomics Uppsala for their unwavering support and expertise. Without you this program would not have been made. Furthermore i would like to thank Adam Ameur at the Department of Immunology, Genetics and Pathology and Uppsala Genome Center for creative feedback and helpful suggestions.

Bibliography

[1] Lodish H, Berk A, Zipursky SL, Matsudaira P, Baltimore D, Darnell J. 2000. Structure of Nucleic Acids. Molecular Cell Biology. 4th edition

[2] Behjati S, Tarpey PS. 2013. What is next generation sequencing? Archives of Disease in Childhood Education and Practice Edition 98: 236–238.

[3] Ginsburg GS, Phillips KA. 2018. Precision Medicine: From Science to Value. Health affairs (Project Hope) 37: 694–701.

[4] Thavarajah R, Mudimbaimannar VK, Elizabeth J, Rao UK, Ranganathan K. 2012. Chemical and physical basics of routine formaldehyde fixation. Journal of Oral and Maxillofacial Pathology : JOMFP 16: 400–405.

[5] Wong SQ, Li J, Tan AY-C, Vedururu R, Pang J-MB, Do H, Ellul J, Doig K, Bell A, MacArthur GA, Fox SB, Thomas DM, Fellowes A, Parisot JP, Dobrovic A. 2014. Sequence artefacts in a prospective series of formalin-fixed tumours tested for variants in hotspot regions by massively parallel sequencing. BMC Medical Genomics 7: 23.

[6] Sena JA, Galotto G, Devitt NP, Connick MC, Jacobi JL, Umale PE, Vidali L, Bell CJ. 2018. Unique Molecular Identifiers reveal a novel sequencing artefact with implications for RNA-Seq based gene expression analysis. Scientific Reports 8: 13121.

[7] Thun MJ, DeLancey JO, Center MM, Jemal A, Ward EM. 2010. The global burden of cancer: priorities for prevention. Carcinogenesis 31: 100–110.

[8] Cancer and Radiation Therapy: Current Advances and Future Directions. online: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3298009/. Accessed June 10, 2019.

[9] Mokhtari RB, Homayouni TS, Baluch N, Morgatskaya E, Kumar S, Das B, Yeger H. 2017. Combination therapy in combating cancer. Oncotarget 8: 38022–38043.

[10] Nurgali K, Jagoe RT, Abalo R. 2018. Editorial: Adverse Effects of Cancer Chemotherapy: Anything New to Improve Tolerance and Reduce Sequelae? Frontiers in Pharmacology, doi 10.3389/fphar.2018.00245.

[11] Berkey FJ. 2010. Managing the adverse effects of radiation therapy. American Family Physician 82: 381–388, 394.

[12] Habeeb NW-A, Kulasingam V, Diamandis EP, Yousef GM, Tsongalis GJ, Vermeulen L, Zhu Z, Kamel-Reid S. 2016. The Use of Targeted Therapies for Precision Medicine in Oncology. Clinical Chemistry 62: 1556–1564.

[13] Clark SL, Rodriguez AM, Snyder RR, Hankins GDV, Boehning D. 2012. Structure-Function of the Tumor Suppressor BRCA1. Computational and Structural Biotechnology Journal, doi 10.5936/csbj.201204005.

[14] Mansoori B, Mohammadi A, Davudian S, Shirjang S, Baradaran B. 2017. The Different Mechanisms of Cancer Drug Resistance: A Brief Review. Advanced Pharmaceutical Bulletin 7: 339–348.

[15] Personalized medicine could transform healthcare. online: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5492710/. Accessed June 12, 2019.

[16] Joo WD, Visintin I, Mor G. 2013. Targeted cancer therapy – Are the days of systemic chemotherapy numbered? Maturitas 76: 308–314.

[17] Widakowich C, Castro G de, Azambuja E de, Dinh P, Awada A. 2007. Review: Side Effects of Approved Molecular Targeted Therapies in Solid Cancers. The Oncologist 12: 1443–1455.

[18] Kokkat TJ, Patel MS, McGarvey D, LiVolsi VA, Baloch ZW. 2013. Archived Formalin-Fixed Paraffin-Embedded (FFPE) Blocks: A Valuable Underexploited Resource for Extraction of DNA, RNA, and Protein. Biopreservation and Biobanking 11: 101–106.s

[19] Ofner R, Ritter C, Ugurel S, Cerroni L, Stiller M, Bogenrieder T, Solca F, Schrama D, Becker JC. 2017. Non-reproducible sequence artifacts in FFPE tissue: an experience report. Journal of Cancer Research and Clinical Oncology 143: 1199–1207.

[20] Moens LNJ, Falk-Sörqvist E, Ljungström V, Mattsson J, Sundström M, La Fleur L, Mathot L, Micke P, Nilsson M, Botling J. 2015. HaloPlex Targeted Resequencing for Mutation Detection in Clinical Formalin-Fixed, Paraffin-Embedded Tumor Samples. The Journal of Molecular Diagnostics 17: 729–739.

[21] Targeted or whole genome sequencing of formalin fixed tissue samples: Potential applications in cancer genomics. online: https://www.researchgate.net/publication/281282581_Targeted_or_whole_genome_sequencing_of_formali Accessed June 13, 2019.

[22] Stratton MR, Campbell PJ, Futreal PA. 2009. The cancer genome. Nature 458: 719–724.

[23] Evaluating the evaluation of cancer driver genes. - PubMed - NCBI. online: https://www.ncbi.nlm.nih.gov/pubmed/27911828. Accessed June 18, 2019.

[24] Fox C.H., Johnson F.B., Whiting J., Roller P.P. Formaldehyde fixation. J. Histochem. Cytochem. 1985;33:845–853. doi: 10.1177/33.8.3894502.

[25] Hykin SM, Bi K, McGuire JA. 2015. Fixing Formalin: A Method to Recover Genomic-Scale DNA Sequence Data from Formalin-Fixed Museum Specimens Using High-Throughput Sequencing. PLoS ONE, doi 10.1371/journal.pone.0141579.

[26] DNA damage reduces Taq DNA polymerase fidelity and PCR amplification efficiency. Sikorsky JA, Primerano DA, Fenger TW, Denvir J: Biochem Biophys Res Commun. 2007 Apr 6: 355(2):431-7.

[27] Hashimoto S, Anai H, Hanada K. 2016. Mechanisms of interstrand DNA crosslink repair and human disorders. Genes and Environment, doi 10.1186/s41021-016-0037-9.

[28] Huang H, Hopkins PB. 1993. DNA interstrand cross-linking by formaldehyde: nucleotide sequence preference and covalent structure of the predominant cross-link formed in synthetic oligonucleotides. Journal of the American Chemical Society 115: 9402–9408.

[29] Guyard A, Boyez A, Pujals A, Robe C, Tran Van Nhieu J, Allory Y, Moroch J, Georges O, Fournet J-C, Zafrani E-S, Leroy K. 2017. DNA degrades during storage in formalin-fixed and paraffin-embedded tissue blocks. Virchows Archiv: An International Journal of Pathology 471: 491–500.

[30] Gaffney EF, Riegman PH, Grizzle WE, Watson PH. 2018. Factors that drive the increasing use of FFPE tissue in basic and translational cancer research. Biotechnic & Histochemistry 93: 373–386.

[31] Schormann N, Ricciardi R, Chattopadhyay D. 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: 1667–1685.

[32] Do H, Dobrovic A. 2012. Dramatic reduction of sequence artefacts from DNA isolated from formalin-fixed cancer biopsies by treatment with uracil-DNA glycosylase. Oncotarget 3: 546–558.

[33] Ehrlich M, Gama-Sosa MA, Huang LH, Midgett RM, Kuo KC, McCune RA, Gehrke C. 1982. Amount and distribution of 5-methylcytosine in human DNA from different types of tissues of cells. Nucleic Acids Research 10: 2709–2721.

[34] Garibyan L, Avashia N. 2013. Research Techniques Made Simple: Polymerase Chain Reaction (PCR). The Journal of investigative dermatology 133: e6.

[35] Wardle J, Burgers PMJ, Cann IKO, Darley K, Heslop P, Johansson E, Lin L-J, McGlynn P, Sanvoisin J, Stith CM, Connolly BA. 2008. Uracil recognition by replicative DNA polymerases is limited to the archaea, not occurring with bacteria and eukarya. Nucleic Acids Research 36: 705–711.

[36] Functions and Malfunctions of Mammalian DNA-Cytosine Deaminases. - PubMed - NCBI. online: https://www.ncbi.nlm.nih.gov/pubmed/27585283. Accessed May 29, 2019.

[37] Kivioja T, Vähärautio A, Karlsson K, Bonke M, Enge M, Linnarsson S, Taipale J. 2012. Counting absolute numbers of molecules using unique molecular identifiers. Nature Methods 9: 72–74.

[38] UMI-Reducer: Collapsing duplicate sequencing reads via Unique Molecular Identifiers — bioRxiv. online: https://www.biorxiv.org/content/10.1101/103267v1.article-info. Accessed May 29, 2019.

[39] Pysam online: https://github.com/pysam-developers/pysam Accessed February 28, 2019.

[40] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010): http://conference.scipy.org/proceedings/scipy2010/mckinney.html

[41] R Core Team (2017). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

[42] 2019. Validated, scalable, community developed variant calling, RNA-seq and small RNA analysis: bcbio/bcbio-nextgen. Blue Collar Bioinformatics

[43] 2019. Tools for working with genomic and high throughput sequencing data.: fulcrumgenomics/fgbio. Fulcrum Genomics

[44] Li H.*, Handsaker B.*, Wysoker A., Fennell T., Ruan J., Homer N., Marth G., Abecasis G., Durbin R. and 1000 Genome Project Data Processing Subgroup (2009) The Sequence alignment/map (SAM) format and unique molecular counts. Bioinformatics, 25, 2078-9. [PMID: 19505943]

[45] Meyer E, Aglyamova GV, Wang S, Buchanan-Carter J, Abrego D, Colbourne JK, Willis BL, Matz MV. 2009. Sequencing and de novo analysis of a coral larval transcriptome using 454 GSFlx. BMC Genomics 10: 219.

[46] Error filtering, pair assembly and error correction for next-generation sequencing reads — Bioinformatics — Oxford Academic. online: https://academic.oup.com/bioinformatics/article/31/21/3476/194979. Accessed June 19, 2019.

[47] Kozarewa I, Ning Z, Quail MA, Sanders MJ, Berriman M, Turner DJ. 2009. Amplification-free Illumina sequencing-library preparation facilitates improved mapping and assembly of GC-biased genomes. Nature methods 6: 291–295.

[48] Kim S, Park C, Ji Y, Kim DG, Bae H, van Vrancken M, Kim D-H, Kim K-M. 2017. Deamination Effects in Formalin-Fixed, Paraffin-Embedded Tissue Samples in the Era of Precision Medicine. The Journal of molecular diagnostics: JMD 19: 137–146.

[49] Yang Z, Yoder AD. 1999. Estimation of the transition/transversion rate bias and species sampling. Journal of Molecular Evolution 48: 274–283.

[50] Do H, Dobrovic A. 2015. Sequence Artifacts in DNA from Formalin-Fixed Tissues: Causes and Strategies for Minimization. Clinical Chemistry 61: 64–71.

[51] https://github.com/clinical-genomics-uppsala/FUSAC

[52] Min S, Lee B, Yoon S. 2017. Deep learning in bioinformatics. Briefings in Bioinformatics 18: 851–869.

[53] Guo Y, Li J, Li C-I, Long J, Samuels DC, Shyr Y. 2012. The effect of strand bias in Illumina short-read sequencing data. BMC genomics 13: 666.

Appendix A

For those interested in reading about FUSAC's functionality in detail, appendix A contains a full software documentation, in the form of a copy of the README file found on FUSAC's github page.

8.1 Readme

FUSAC (FFPE tissue UMI based Sequence Artefact Classifier) is a python-based program for the identification and classification of FFPE artefacts in UMI-tagged sequence data. Using a VCF and BAM-file as input, FUSAC is able to successfully identify groups and collapse all reads aligning to a position called by the VCF, generating consensus sequences for each UMI as well as identifying their string of origin before amplification. These consensus sequences are then compared with their mate, and thus FUSAC is able to not only identify C->T:G->A artefacts left by hydrolytic deamination, but also identify true variants, deletions, unknowns and any other type of mismatch.
FUSAC requires the user to have basic understanding of the data they wish to study, namely the location of the UMI and how it is structurally stored within the BAM-file. More specifically it requires the user to define both the location of the tag (query name or the RX-field) as well as the character the UMI is separated by if such a character exists.

```
NDX550220_RUO:8:HMTKVBGX9:2:13110:1259:16191:UMI_TGAGCTG+TGCACCA   1107  chr8  11780  60
NDX550220_RUO:8:HMTKVBGX9:2:13110:1259:16191:UMI_TGAGCTG+TGCACCA   1187  chr8  11780  60
NDX550220_RUO:8:HMTKVBGX9:2:13111:26743:16207:UMI_TGAGCTG+TGCACCA  83    chr8  11780  60
NDX550220_RUO:8:HMTKVBGX9:2:13111:26743:16207:UMI_TGAGCTG+TGCACCA  163   chr8  11780  60
```

Figure 12: *BAM-file read example in which the UMI-tag is located in the query-name, and is separated by the character +. Output generated by SAMtools*

```
26_umi:6210799:UMI_GGTTACTGGTTACT   129  chr1  10234   11  45M6S   chr7   148516716  0
26_umi:19:UMI_TACCTGTATTCACA        163  chr1  131165  22  92M     =      131268     196
26_umi:19:UMI_TACCTGTATTCACA        83   chr1  131268  22  93M     =      131165     -196
26_umi:133:UMI_ATTCACAAAGGATG       163  chr1  133410  14  93M     =      133538     221
```

Figure 13: *BAM-file read example in which the UMi-tag is located in the query-name and is not separated by any character. Output generated by SAMtools*

From this input, FUSAC generates a modified VCF-file as output. The output VCF is a copy of the input VCF but has a modified ***FILTER*** field where any classified FFPE artefact will display ***FFPE***. Furthermore, the output VCF will also have a modified ***FORMAT*** field where the unique molecular counts for the variant position variant classification (Reference nucleotide, True variant, FFPE artefact, Unknown or Deletion) are listed. This field also contains the unique molecular counts for the reference genome nucleotide as well as the called variant nucleotide for paired reads on the positive and negative strand, as well as the unique molecular counts on single strand reads belonging to the positive or negative strand.

8.2 Prerequisites

FUSAC is based on the python module Pysam 0.15.0, and thus requires this to be installed. Furthermore, FUSAC requires the library Pandas 0.24.2. These can be obtained for free through their respective github page, or easily installed through pip.

Table 14: Example on how to install the required python-modules.

Command
sudo pip install pysam=0.15.0
sudo pip install pandas=0.24.2

There are three fundamental assumptions made by FUSAC to yield the desired output:

1. The input sequence data is UMI-tagged
2. The UMI is located either in the query-name or RX-tag fields in the BAM (translated to SAM) file.
3. If the UMI is located in the query-name, it is located as the last entry and separated from the rest of the query name through a character.
   (a) Example_Name_With_Tag_At_End_UMI_ACTACTA+ACTACTA

The first assumption is necessary due to FUSAC's algorithm working in a classifying manner. To identify all reads stemming from a source molecule, a common identifier in the form of the UMI is vital for collapsing reads into a consensus sequence. The second assumption is necessary to properly locate the called variant within each subsequent read belonging to a UMI of interest. FUSAC uses the reference genome to identify the correct position for each subsequent read. And thus, if gapped bases are not included, this position will be incorrect, thus yielding an incorrect comparison. The third and fourth assumption are both necessary to ensure that the UMI-tagged data can be properly extracted from each read.

8.3 Interpreting FUSAC's output

The output from FUSAC is generated in the form of a modified VCF-file. The modified VCF is identical to the input VCF with the exception of a custom FILTER tag (***FFPE***) for classified FFPE variants, as well as two custom FORMAT fields called ***UMI*** (Unique Molecular Identifier counts) and SUMI (Singleton Unique Molecular Identifier counts). Both these fields contain the unique counts for the five variant-types (table 15 on the next page), the unique counts for the reference and variant nucleotides split by strand for read pairs where fragments from both strands have been detected, and the unique counts for the reference and variant nucleotides on single strand reads belonging to the positive and negative strand.

Table 15: Variant-type classifications in the UMI and SUMI fields.

Variant classification	Definition
Reference nucleotide	Reference nucleotide
True variant	Variant nucleotide
FFPE artefact	C->T:G->A mismatch
Unknown	N
Deletion	-

8.3.1 Example Unmodified VCF-record

Example of a unmodified variant record in a VCF-file (table 16). The field INFO have been cut out for sake of clarity and are marked as **...**. The ***FORMAT*** field values are displayed separately in table 17 for clarity.

Table 16: Example unmodified variant record.

Field	Value
CHROM	chr1
POS	4,367,323.00
ID	rs1490413
REF	G
ALT	A
QUAL	.
FILTER	FFPE
INFO	...
FORMAT	GT:AD:AF:DP

Table 17: Example unmodified VCF FORMAT-fields.

FORMAT field	Value
GT:AD:AF:DP:	0/1:571,632:0.527:1203:

8.3.2 Example modified VCF-record

Example of the FUSAC output (table 5 on page 17) based on the variant record as shown in table 3 on page 16. The field ***INFO*** have been cut out for sake of clarity and is marked as **...**. The ***FORMAT***, ***UMI*** and ***SUMI*** fields are displayed in a separate table for clarity (table 19 on the following page). As can be seen, the ***FILTER*** field has been modified to display ***FFPE*** instead of pass. This tells us that that the algorithm detected at least one read pair with a unique UMI and a C->T:G->A mismatch at the variant record variant-call position.

Table 18: Example FFPE classed variant record generated by FUSAC.

Field	Value
CHROM	chr1
POS	4,367,323.00
ID	rs1490413
REF	G
ALT	A
QUAL	.
FILTER	FFPE
INFO	...
FORMAT	GT:AD:AF:DP:UMI:SUMI

Table 19: Example FFPE classed FORMAT-fields, generated by FUSAC.

FORMAT field	Value
GT:AD:AF:DP:	0/1:571,632:0.527:1203:
UMI:	235;313;15;0;0;313;328;250;235;272;322;306;256:
SUMI:	0;0;0;0;0;0;0;0;0;0;0;0;0

Further investigation in the ***FORMAT*** field, more specifically the two fields UMI (table 20 on the next page) and ***SUMI*** generated by FUSAC, shows us that the unique molecular counts for reference nucleotide is 235, the counts for a true variant is 313 and the counts for FFPE artefacts is 15. The counts for a deletion or an unknown is deemed to be 0, the counts for the reference nucleotide (G) on the positive strand for paired reads as 313, and on the negative strand 328. Finally, the counts for the variant nucleotide (A) on the positive strand for paired reads is deemed to be 250, and on the negative strand 235. We can quickly see that this score is consistent with the 15 FFPE artefact classifications that were found by FUSAC. For single strand reads, the unique molecular counts for the reference nucleotide is 272 on the positive strand and 322 on the negative strand. Whereas the counts for the variant nucleotide is 306 on the positive strand, and 256 on the negative strand. Finally, the field ***SUMI*** indicates that no unique molecular counts for singletons were found at the given variant record.

Table 20: Example UMI-field values generated by FUSAC.

Value	Meaning
235;	Unique molecular counts for reference nucleotide
313;	Unique molecular counts for true variant
15;	Unique molecular counts for FFPE artefact
0;	Unique molecular counts for unknown
0;	Unique molecular counts for deletion
313;	Unique molecular counts for reference nucleotide on positive strand for paired reads
328;	Unique molecular counts for reference nucleotide on negative strand for paired reads
250;	Unique molecular counts for variant nucleotide on positive strand for paired reads
235;	Unique molecular counts for variant nucleotide on negative strand for paired reads
272;	Unique molecular counts for reference nucleotide on positive strand for single strand reads
322;	Unique molecular counts for reference nucleotide on negative strand for single strand reads
306;	Unique molecular counts for variant nucleotide on positive strand for single strand reads
256;	Unique molecular counts for variant nucleotide on negative strand for single strand reads

8.4 Quickstart

Required input arguments for running FUSAC are ***-b*** and ***-v***, which are the respective paths to the input BAM and VCF file. Furthermore, an indexed BAM (***BAI***) file is required for extracting desired segments of the BAM-file. The other input flags are not required, but should be changed if the default value is not representative of the desired output. To minimize run-time and CPU-load FUSAC can run on multiple threads. Unfortunately, as pickling cannot deal with open file handles, multiprocessing is not a viable option as this would require the file to be opened for every read aligning to the variant position. Instead, FUSAC uses the python ***threading*** module with a producer-consumer approach, where the producer generates and populates a queue, and the consumer thread extracts the inhabitants of this queue for analysis. To control this threading process, the arguments ***threads*** (***-t***) and ***queueSize*** (***-qs***) determine the number of threads to be run and the size of the threading queue respectively. The default values for threads and queueSize respectively are one active thread and an infinite queue, but can be set to any integer value desired.

The default FFPE classification mode focuses solely on C->T:G->A artefacts, however if desired the program can also identify any mismatching consensus nucleotides using the input flag ***ffpeBases*** (***-fb***) with the option ***all***. Lastly, FUSAC is entirely dependent on the UMI being properly extracted to ensure that reads are assigned to the positive strand or the negative strand as origin. Therefore, the user can specify through the ***umiPosition*** (***-up***) tag if the UMI is located in the ***query-name*** (***qrn***) or the ***RX-tag*** respectively (***rx***). Furthermore, the UMI needs to be split from the query name, and then in half to be rearranged correctly. As the type of characters can vary from data set to data set, the parameters ***querySplitCharacter*** (***qsc***) can be used to split the UMI-tag from the query name, and the parameter ***UMISplitCharacter*** (***usc***) can be set to split the UMI-tag. For reads where the UMI is not separated by a tag, an empty string (***””***) should be used for ***-usc*** to split the tag in half. The ***csvFile*** (***cf***) input controls whether or not FUSAC generates an output CSV file based on the FUSAC output. This CSV generates a separate row for each variant record with columns for the unique molecular counts for the reference genome nucletoide, the variant-call nucleotide, the number of FFPE calls, the overall frequency of FFPE artefacts for each variant record, and the type of mismatch for the variant record. The default setting is to generate the CSV, but if this is not required the function can be turned off using the input ***no***. Finally, the ***percentageExclude*** (***pe***) input controls whether or not to filter the results when generating the output CSV-file based on the frequency of detected FFPE artefacts in a output variant record. The input can be set to any two integer values, 0-100 being default for no filtering. If we were to select 1-99% as an example, all variant records with FFPE VAF below 1% and above 99% would be excluded from the output CSV-file.

Table 21: FUSAC input flags and their respective function.

Flag	Name	Function	Req.	Def.	Alt.
-b	inputBAM	Input BAM file	Yes	N/A	Any
-v	inputVCF	Input VCF file	Yes	N/A	Any
-t	threads	No. threads to run the program	No	1	Any
-qs	queueSize	Threading queue-size	No	Infinite	Any
-fb	ffpeBases	Bases used for FFPE classification	No	C->T:G->A	All
-up	umiPosition	Location of the UMI in a read	No	Query-name	RX-tag
-qsc	querySplitCharacter	Split character for the query-name and UMI	No	_	Any
-usc	UMISplitCharacter	Split character for the UMI	No	+	Any
-cf	csvFile	Generate an output CSV file	No	yes	no
-pe	percentageExclude	Exclude variant records from output CSV-file with FFPE frequency under input value	No	yes	no

Req. = Required, Def. = Default, Alt. = Alternative.

8.4.1 Example 1

We wish to classify all mismatches belonging to the file example_bam using the example_vcf file. The Reads in the example_bam file have their UMI stored in the query-name, which is separated by the character _. The program is being run on a laptop with 4 cores, and we wish to limit the queue to 9 variant records.

Table 22: FUSAC call-sequence for the case presented in Example 1.

Function call
python fusac.py -b example_bam.bam -v example_vcf.vcf -t 3 -qs 9 -b all -sc _

8.4.2 Example 2

We wish classify only C->T:G->A artefacts belonging to the file example_bam using the example_vcf file. The reads in the example_bam file have their UMI stored in the RX-tag, and are not separated by any character. The program is being run on a cluster with 16 cores. We do not wish to limit the queue, but rather have it infinite. Furthermore, we do not wish to generate a CSV file.

Table 23: FUSAC call-sequence for the case presented in Example 2.

Function call
python fusac.py -b example_bam.bam -v example_vcf.vcf -t 15 -up rx -sc "" -cf no

8.4.3 Reference manual

The reference manual covers all functions belonging to FUSAC, describing their purpose, methodology and input/output. Use it to get a better understanding of FUSAC or if you have any questions.

8.4.4 Main

After the user has supplemented their desired input arguments using the flags covered in the section ***Quickstart***, the main function will use the vcf-file as an input argument to populate a newly generated double-ended-queue (deque) of size ***-qs***, with the variant-calls found within the VCF-file. Based on the flag ***threads***, the function will then create ***-t*** consumer threads, their results appended to a result-queue. Once the VCF has been iterated through entirely, the output will be written to an output vcf-file.

8.4.5 QueueThread

The QueueThread function is a producer which takes the variant-calls found within the VCF-file and populates a deque (while not full) to be used by the consumer function.

Table 24: Input arguments for the QueueThread function.

Input	Function
vcf_file	VCF filehandle
thr_que	Deque to be populated

8.4.6 ResultThread

While the deque is not empty, the ResultThread function calls upon the vcf_extract function using the variant record extracted as input. The subsequent results are then stored in a separate thread while not None.

Table 25: Input arguments for the ResultThread function.

Input	Function
bam_path	Path to BAM-file
thr_que	Populated deque to be used as input for vcf_extract
res_que	Queue to be populated with the results from vcf_extract
ffpe_n	Optional input argument controlling which mismatches to consider for FFPE classification
ext_fun	Function for extracting the UMI from a read
spl_fun	Function used for splitting the UMI in a read
q_spl_cha	Character separating UMI from the query name
u_spl_cha	Character for splitting the UMI-tag

8.4.7 vcf_extract

The vcf_extract function uses the supplemented variant record to extract all reads in the BAM-file overlapping with it's position. This newly generated list is used for the var_extract function to return molecular data. The output from var_extract is then subsequently used in the inf_builder function. Finally, the output from inf_builder is added to the copied input record and returned.

Table 26: Input arguments for the vcf_extract function.

Input	Function
record	variant record of interest
bam_file	BAM-file file-handle
ffpe_n	Optional input argument controlling which mismatches to consider for FFPE classification
ext_fun	Function for extracting the UMI from a read
spl_fun	Function used for splitting the UMI in a read
spl_cha	Character used for splitting the UMI

8.4.8 var_extract

The var_extract function first calls the umi_maker function which creates a dict based on the directionality and UMI's of the supplemented reads in the bam_lst. This dict is then used to call the pos_hits function which will return a dict of consensus nucleotides for each UMI. For paired reads (ie: exists on both the positive strand and the negative strand for a UMI) the output from the pos_hits function is then used to call the ffpe_finder function which returns a dict containing the variant type for each UMI along with the unique molecular counts for each variant type. The pos_checker function returns a dict with data regarding the unique molecular counts for each variant type on the variant record position, as well as the nucleotides present for each variant with respect to their UMI.

Table 27: Input arguments for the var_extract function.

Input	Function
record	variant record of interest
bam_lst	Input list of BAM-reads aligning to the variant call
rec_pos	The position of the variant in the reference genome
var_nuc	The nucleotide called in the variant record
ref_nuc	The nucleotide found in the reference genome at the variant-call position
ffpe_n	Optional input argument controlling which mismatches to consider for FFPE classification
ext_fun	Function for extracting the UMI from a read
spl_fun	Function used for splitting the UMI in a read
spl_cha	Character used for splitting the UMI

Table 28: Input and output for the var_extract function.

Dict	Structure
Input	Example list: bam_lst = [read_1, read_2 ...]
Output	Example dict: mpd_res[umi_key] = {"Single_Hits": Pos_Str_Hits: {}, Neg_Str_Hits:{C,T}, "Mate_Hits": True_Variant_Hits": {}, "FFPE_Hits": {"Pos_Str": C, "Neg_Str": T}, "N_Hits": {}, "Del_Hits": {}, "Reference_Support": 0, "True_Variant_Support": 0, "FFPE_Support": 1, "N_Support": 0, "Del_Support": 0}}

8.4.9 inf_builder

The inf_builder function uses the output from var_extract to generate a list containing strings representing the data found for each record, more specifically unique molecular counts for each variant-type, as well as the unique molecular counts for the reference and variant call for the positive and negative strand.
The inp_dict is meant to be the output from the var_extract function,and is divided into two dicts named Single Hits and Mate Hits. The Single-dict contains the unique molecular counts for the reference genome nucleotide and the variant nucleotide based on all reads without a mate. The Mate-Hits dicts instead contains data regarding the variant-classification, the unique molecular counts for each variant type, and the unique molecular counts for the reference genome nucleotide and the variant nucleotide based on reads with a mate.
Returns a list containing the unique molecular counts for each variant-type, as well as the unique molecular counts for the reference and variant call for the positive and negative strand.

Table 29: Input arguments for the inf_builder function.

Input	Function
read	Read of interest
inp_dict	Input dict dict for mapped and unmapped reads. Each of these dicts containing a single-hits and a mate-hits dict. The mate-hits dict in turn contains data regarding if the variant is a variant, reference nucleotide, FFPE artfefact deletion or N-call. Whereas the single-hits dict contains positional data for reads with no mate
ref_nuc	Nucleotide in reference genome for the variant record variant position
var_nuc	Variant nucleotide for the variant record variant call

Table 30: Input and output for the inf_builder function.

Dict	Structure
Input	Example dict for a FFPE artefact: inp_dict = umi_key: {”Single_Hits”: Pos_Str_Hits: {}, Neg_Str_Hits:{C,T}, ”Mate_Hits”: True_Variant_Hits”: {},”FFPE_Hits”: {”Pos_Str”: C, ”Neg_Str”: T}, ”N_Hits”: {}, ”Del_Hits”: {}, ”Reference_Support”: 0, ”True_Variant_Support”: 0, ”FFPE_Support”: 1, ”N_Support”: 0, ”Del_Support”: 0}}
Output	Example output list for one FFPE artefact: [0;0;1;0;0;1;0;1;0;0;0;0;0]

8.4.10 csv_maker

The csv_maker function generates an output CSV-file based on the FUSAC output containing data for each variant record. More specifically regarding the unique molecular counts for the reference genome nucleotide, the variant-call nucleotide,the number of FFPE calls, the overall frequency of FFPE artefacts for each variant record, and the type of mismatch for the variant record. Through extracting this info and calling the csv_record_maker function, it populates a series of list that are then written to the new CSV.

Table 31: Input arguments for the csv_maker function.

Input	Function
read	Read of interest
vcf_file	The output VCF file generated by FUSAC
ref_nuc	Nucleotide in reference genome for the variant record variant position
ffpe_n	Optional input argument controlling which mismatches to consider for FFPE classification
per_exl	Exclude data from variant records with FFPE frequency under input value

8.4.11 csv_record_maker

The csv_record_maker extracts information from a FFPE tagged variant-record (classified by FUSAC as showing unique molecular counts for a FFPE artefact at the variant position), more specifically from the ***samples*** field. The function generates data regarding the unique molecular counts for the nucleotide in the reference genome for the variant-call position, the variant nucleotide in the variant record, the number of FFPE artefacts found, the number of unknowns found, the number of deletions found, as well as the percentile ratio of FFPE artefacts in the variant record. The generated data is then used to populate the lists used as input.

Table 32: Input arguments for the csv_record_maker function

Input	Function
read	Read of interest
pos_lst	List containing the position for records
change_lst	List containing the mismatch for studied variant records
ref_lst	List containing the unique molecular counts for the reference genome nucleotide for each studied variant call position respectively
var_lst	List containing the unique molecular counts for the variant record variant-call position for each studied variant call position respectively
ffpe_lst	List containing the unique molecular counts for FFPE artefacts on the variant record variant-call position for each studied variant call position respectively
perc_lst	List containing the percentual ratio of unique molecular counts for FFPE artefacts to the total number of molecules studied for each studied variant call position respectively
record	The variant record of interest
per_exl	Exclude data from variant records with FFPE frequency under input value

8.4.12 qrn_ext

The qrn_ext function extracts a UMI from a read, based on the key being present as the last item in the query-name. Returns the UMI.

Table 33: Input arguments for the qrn_ext function

Input	Function
read	Read from which the UMI is to be extracted from

8.4.13 rx_ext

The rx_ext function extracts a UMI from a read, based on the key present in the RX-tag. Returns the UMI.

Table 34: Input arguments for the rx_ext function.

Input	Function
read	Read from which the UMI is to be extracted from

8.4.14 cha_splt

The cha_splt function splits the umi_string based on the split-character argument. Returns a list containing the UMI split into two components.

Table 35: Input arguments for the cha_splt function.

Input	Function
umi_str	A string representing the UMI to be split
char	Character to split the umi-string by

8.4.15 hlf_splt

The hlf_splt function splits the umi_string in half based on it's length. Returns a list containing the UMI split into two components.

Table 36: Input arguments for the rx_splt function.

Input	Function
umi_str	A string representing the UMI to be split
char	Character to split the umi-string by (not used but required by the function call)

8.4.16 umi_maker

The umi_maker function rearranges the UMI belonging to a read, based on if the read is read 1 or read 2 in combination with its directionality. To extract the UMI from the read the ext_fun function is used, which calls either the qrn_ext or rx_ext function based on user input. The UMI is then transformed from a variant record entry into a string, and used as input for the spl_fun function. Returns the query-name of the read, the strand it belongs to, and the adjusted UMI-sequence

Table 37: Input arguments for the umi_maker function.

Input	Function
read	Read of interest
splt_umi	UMI for the read split into two strings

8.4.17 pos_hits

The pos_hits function selects the most prominent base for a UMI of interest. The function works by iterating through all query-names in the input list and determines if the query-name has a mate or not. The function then calls the base_check function to retrieve the base matching the variant record position for each read. In the next step the retrieved base is matched against a dict, and depending on the outcome adds to a counter representative of the base. This process is repeated for each query name and it's subsequent read, and the resulting dict is then used to determine the most prominent nucleotide for the UMI, effectively collapsing all reads belonging to a UMI. Returns the consensus nucleotide

Table 38: Input arguments for the pos_hits function

Input	Function
inp_dict	Input list of reads categorized by their query-name
rec_pos	The position of the called variant in the reference genome

Table 39: Input and output for the pos_hits function

Dict	Structure
Input	Example dict: input_dict = {example_name_UMI_ACTGCA+ACTGCA: {read1, read2}, example_2_name_UMI_TGACGT+TGACGT: {read2}}
Output	Example list: cons_lst = [UMI_tag_1: C, UMI_tag_2: T]

8.4.18 base_check

The base_check function checks the variant record position against the supplemented read, and then extracts the nucleotide belonging to this position in the read. Returns the nucleotide in the read mapping against the variant record position.

Table 40: Input arguments for the base_check function.

Input	Function
read	Read of interest
rec_pos	The position of the called variant in the reference genome

8.4.19 ffpe_finder

The ffpe_finder function is made to classify the variant type for paired UMI-reads. All-together the UMI and it's variant record position can be classified as:

1. Reference nucleotide
2. True Variant
3. FFPE artefact
4. Unknown
5. Deletion

The function uses a dict of paired reads containing their consensus nucleotides categorized through their UMI, which is then iterated through for every UMI. It then uses the consensus nucleotide originating from the positive and negative strand to classify the UMI through comparing these to one another. If the two consensus nucleotides are equal to one another and furthermore equal to the base in the reference genome, the UMI is determined to be ***Reference nucleotide***. If the two consensus nucleotides are equal to one another and furthermore equal to the variant in the variant record, they are instead deemed to be a ***True variant***. In default mode, a FFPE classification only occurs if there is a mismatch between the two consensus nucleotides, if one of the consensus nucleotides is equal to the variant in the variant record and finally if the mismatch is of a C->T or G->A type. Alternatively, if the flag ***ffpe_b*** has been called with the input ***all***, the function instead classifies any mismatch between the consensus nucleotides for the positive strand and the negative strand as a FFPE artefact. If any of the consensus nucleotides are equal to ***N*** or **-** the UMI is instead deemed to be ***Unknown*** or ***Deletion*** respectively.
After a UMI is classified, a counter is added too, and the UMI is stored within a dict named after the variant type. Once the algorithm has iterated through every UMI within the cons_dict, it creates a new dict containing all variant-type dicts as well as their unique molecular counts, which is then returned.

Table 41: Input arguments for the ffpe_finder function.

Input	Function
cons_dict	Dict containing the consensus nucleotides for the positive strand and the negative strand classified through their UMI.
var_nuc	The nucleotide called in the variant record
ref_nuc	The nucleotide found in the reference genome at the variant-call position
ffpe_b	Optional input argument controlling which mismatches to consider for FFPE classification

Table 42: Input and output for the ffpe_finder function.

Dict	Structure
Input	Example dict for a FFPE artefact: cons_dict = {Pos_Str_Hits: C, Neg_Str_Hits: T}
Output	Example dict for a FFPE artefact: var_dict = {"Single_Hits": Pos_Str_Hits: {}, Neg_Str_Hits:{C,T}, "Mate_Hits": True_Variant_Hits": {},"FFPE_Hits": {"Pos_Str": C, "Neg_Str": T}, "N_Hits": {}, "Del_Hits": {}, "Reference_Support": 0, "True_Variant_Support": 0, "FFPE_Support": 1, "N_Support": 0, "Del_Support": 0}}

8.4.20 mol_count

The mol_count function uses the output generated by the var_extract function, more specifically the unique molecular counts for each variant classification type as well as the counts for the reference and variant call for the positive and negative strand. Returns a list consisting of the unique molecular counts for each variant type

Table 43: Input arguments for the mol_count function.

Input	Function
inp_dict	The output dict from the var_extract function

Table 44: Input and output for the mol_count function.

Dict	Structure
Input	Example dict for a FFPE artefact: inp_dict = umi_key: {"Single_Hits": Pos_Str_Hits: {}, Neg_Str_Hits:{C,T}, "Mate_Hits": True_Variant_Hits": {},"FFPE_Hits": {"Pos_Str": C, "Neg_Str": T}, "N_Hits": {}, "Del_Hits": {}, "Reference_Support": 0, "True_Variant_Support": 0, "FFPE_Support": 1, "N_Support": 0, "Del_Support": 0}}
Output	Example list for a FFPE artefact: [0,0,1,0,0]

8.4.21 nuc_count

The nuc_count function uses the output generated by the var_extract function, more specifically the unique molecular counts for a given nucleotide of interest. Returns a dict containing subsequent dicts with the unique molecular counts for the nucleotide for paired reads on the positive and negative strand, as well as the unique molecular counts for the nucleotide on single reads belonging to the positive and negative strand.

Table 45: Input arguments for the mol_count function.

Input	Function
inp_dict	The output dict from the var_extract function
nuc	The nucleotide of interest

Table 46: Input and output for the nuc_count function.

Dict	Structure
Input	Example dict for a FFPE artefact: inp_dict = umi_key: {"Single_Hits": Pos_Str_Hits: {}, Neg_Str_Hits:{C,T}, "Mate_Hits": True_Variant_Hits": {},"FFPE_Hits": {"Pos_Str": C, "Neg_Str": T}, "N_Hits": {}, "Del_Hits": {}, "Reference_Support": 0, "True_Variant_Support": 0, "FFPE_Support": 1, "N_Support": 0, "Del_Support": 0}}
Output	Example dict for the nucleotide C for a imaginary input_dict: n_sup = {"Paired": {"Pos_Str": {C: {12}}, "Neg_Str": C: {11}, "Pos_Str_Single": C: {5}, "Neg_Str_Single": C: {2}}

8.4.22 FAQ

The FAQ aims to answer questions the reader may have regarding FUSAC and it's use.

Why do i need a VCF-file to use FUSAC?:
The VCF is required to identify and target known variant calls. Based on the assumption that any FFPE artefact will initially be classified as a variant in the VCF. All variant-records present in the VCF are then classified by FUSAC to evaluate if they are FFPE artefacts or not. When creating FUSAC, I felt no need to construct a custom variant-caller due to the abundance of adequate variant-calling software already existing.

Why is there a need for a BAI file, is the BAM not enough?:
A BAM file is stored in binary format, and thus has no internal structure to use for purposes of fetching. The BAI file provides an indexed form of the BAM, allowing us to fetch specific reads, the FUSAC algorithm requires this functionality.

The unique molecular counts for the variant-call variants in my output seems comparatively low to the read-depth, why is this?:
FUSAC can only classify variant types if the UMI it is studying has both a positive and negative strand consensus sequence. Thus, if the studied data has many singletons or if only one of the strings align to the variant-call position, there will be a limited amount of classifications that can be made. The single paired reads and the singletons instead provide unique molecular counts for the variant call nucleotide as well as the reference genome nucleotide for the variant-call position.

What are the recommended number of threads to run FUSAC on?:
The optimal number of threads depends entirely on the system FUSAC is to be run on. Ideally, one should use n-1 threads, where n is the total number of available processors on the system.

How do i know where the UMI is stored, and how do i know what character is separating it (if there is any)?:
Due to the high variability of sequence data, this is not an automated process by FUSAC, but instead requires the user to manually inspect one of their reads. We recommend the user to utilize the free software SAMtools [44] for this purpose.

Where can i dowload FUSAC, and is FUSAC free?:
FUSAC is a publically available software, and can be downloaded free of charge from it's github repository: https://github.com/clinical-genomics-uppsala/FUSAC

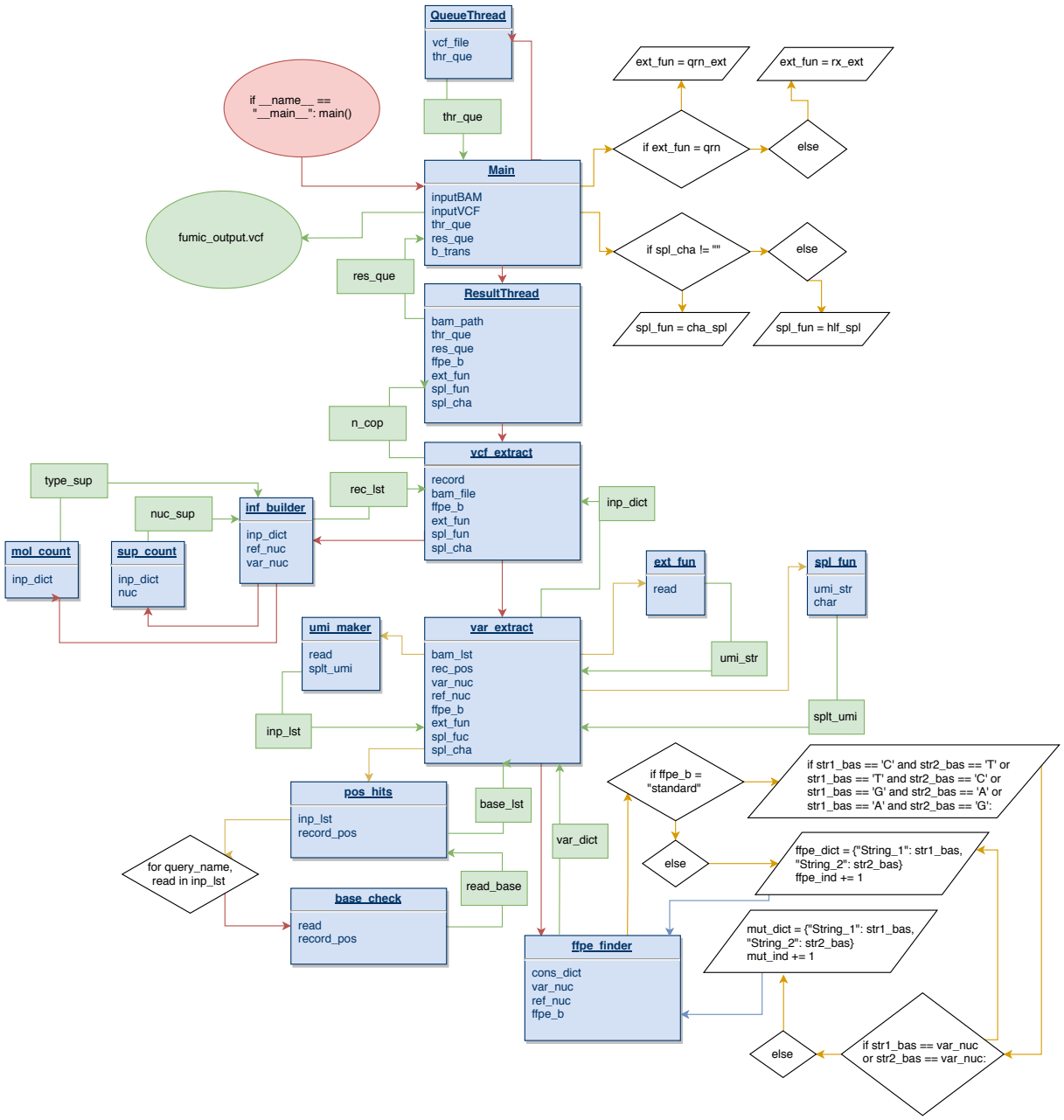
I have identified a bug or have suggestions for improving the program, where can i contact you regarding this?:

All suggestions related to modifying the program should be submitted through it's github repository: https://github.com/clinical-genomics-uppsala/FUSAC

I have a question not covered by the FAQ, where can i contact you regarding this?:
All questions regarding the program should be submitted through it's github repository:
https://github.com/clinical-genomics-uppsala/FUSAC

Appendix B

Flowchart of FUSAC for a run without export to CSV. Red arrows represent the input from a function to another function. Orange arrows represent in-function relations. Blue arrows marks in-function returned values. Green arrows represent the output of a function



Appendix C

Barplots for the output generated by BCFtools in section 4 on page 22

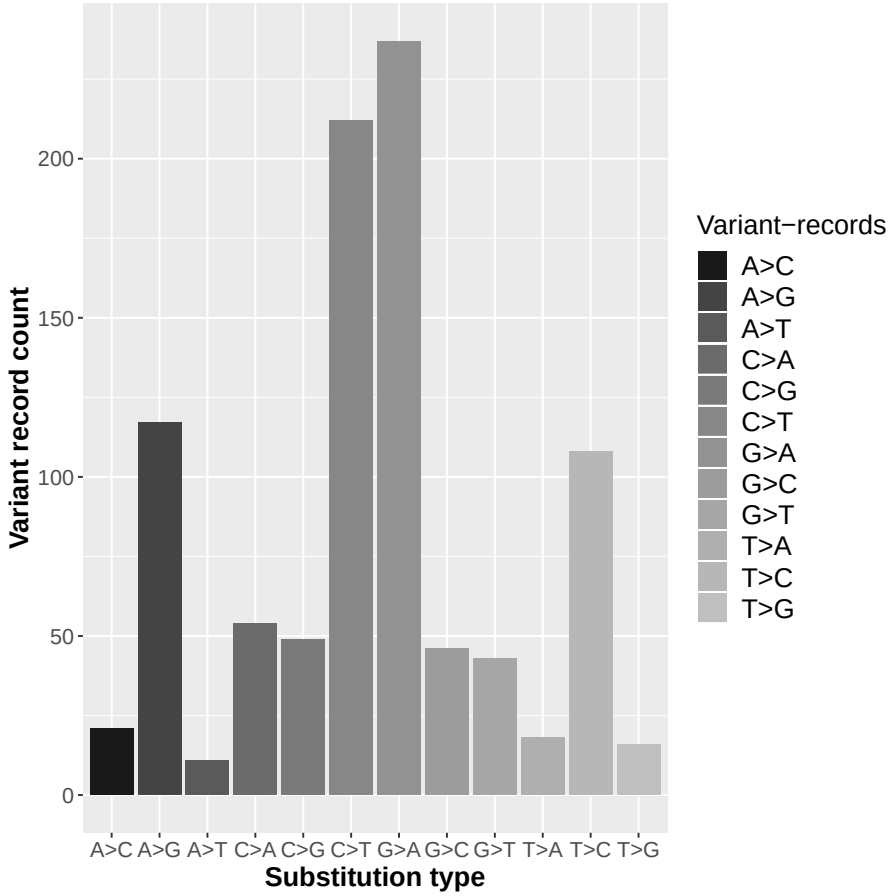


Figure 14: Substitution-type and counts for variant records in the in the UMI-filtered data set, generated from the output of BCFtools

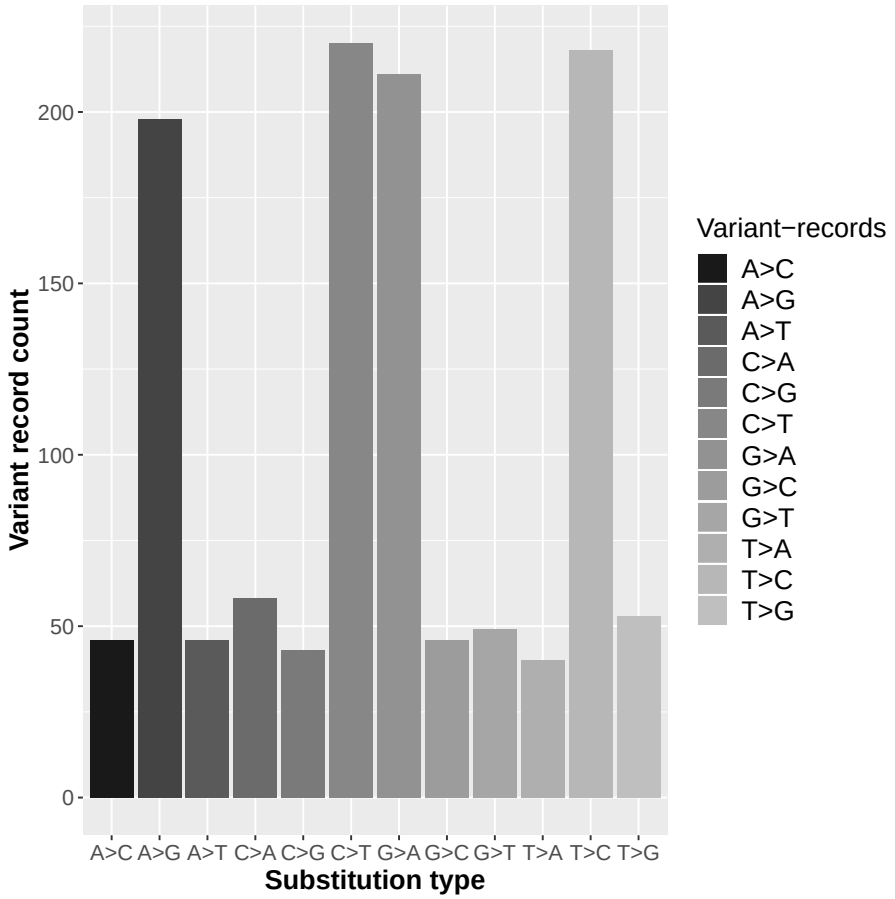


Figure 15: Substitution-type and counts for variant records in the in the non-FFPE-treated data set, generated from the output of BCFtools