Somatypus

A Platypus-based variant calling pipeline for cancer data

Adrian Baez-Ortega

University of Cambridge

Version 1.3

December2016

www.tcg.vet.cam.ac.uk

Contents

Licence

Author: Adrian Baez-Ortega (ORCID 0000-0002-9201-4420; ab2324@cam.ac.uk)

1. What is Somatypus?

The identification of mutations —also known as variant calling — in cancer sequence data is a challenging task, because of the presence of copy number alterations, sample contamination from surrounding healthy tissue, and difficulties associated with aligning cancer DNA sequence reads to a reference (healthy) genome. As a result, novel computational tools for variant calling have been designed to specifically handle cancer data.

Transmissible cancers, which arise in one individual before spreading clonally throughout the population as an invasive allograft, pose a particular challenge. Transmissible cancer cells are somatic cells from a founder animal with the ability to infect other individuals. Therefore, unlike in “conventional” cancers, comparisons between the cancerous and normal tissue of the same individual cannot be applied here. Variant calling in transmissible cancer must address the issue that the cancer cells are derived from a different host, and that variation between hosts occurs. Until now, there was no available software tool or pipeline directed toward variant calling in transmissible cancer, mainly due to the extremely rare occurrence of the latter(only three diseases of this kind have been described, none of them in humans).

We present Somatypus, an open-source pipeline for identifying germline and somatic SNVs and indels in sequencing data coming from a set of unpaired samples. It has been designed to offer great sensitivity and specificity, even for low-frequency somatic mutations found in cancer genomes. Although this pipeline can be applied to any cancer sequence data, it is particularly useful for the processing of data obtained from many tumour samples that are closely related to each other and lack matched normal samples, such as transmissible cancer or metastasis data.

Somatypus is built upon the powerful variant caller Platypus, developed at the Wellcome Trust Centre for Human Genetics (**http://www.well.ox.ac.uk/platypus**). We selected Platypus as our variant caller after confirming that its sensitivity and specificity are comparable to or above those of other renowned tools. Also, Platypus offers performances up to 100 times better than those of other publicly available variant callers. We have designed our pipeline around several executions of Platypus, aimed at achieving maximum sensitivity in the initial calling steps, followed by diverse genotyping and filtering steps to provide a highly precise removal of false-positive and dubious variant calls, rendering a final set of high-confidence variants. Every step and setting in the pipeline has been assessed in detail through visual inspection of randomly sampled subsets of the resulting variant sets.

Although Somatypus still needs to be statistically evaluated using public benchmark datasets, we believe that it offers remarkable calling power and accuracy. Its estimated false positive rate is <2%. Although the false negative rate is difficult to estimate without a benchmarking data set, we estimate it to be comparable to that of other callers. We have confirmed that in many cases of missed variants, Somatypus does initially call them, and subsequently filters them out based on reasonable criteria about their reliability or sequence context.

Somatypus has been tested on Ubuntu (14.04.4) systems, and it should work well on any Linux distribution. It has not been tested on Mac systems, although it may work after some minor code modifications.

1.1 Somatypus does…

- Call single nucleotide variants (SNVs) and short insertions/deletions (indels) from any number of sequence alignment files (subject to memory constraints), at a probably unmatched speed.
- Filter low-quality or ambiguous variants, while keeping those variants occurring with low frequency (in a few samples) or aberrant copy number (not fitting a diploid model).
- Allow the use of additional calling options for Platypus, as long as they do not override the ones defined in the pipeline.
- Run seamlessly from a single command.
- Resume execution after an unexpected interruption.

1.2 Somatypus does not…

- Call long indels, structural variants or copy number changes.
- Manage samples in pairs, or distinguish between “tumour” and “host/normal” samples (or between germline and somatic variants). Identification of somatic variants must be performed downstream.

- Take into account cross-sample contamination. Identification of variants caused by contamination must be performed downstream.
- Allow extensive workflow customisation (unless the source code is altered).

2. Installation

2.1 Dependencies

Somatypus has been designed as a set of Bash and Python scripts that do not require a real installation, but simply adding the software directory to the PATH environment variable, thus making set-up straightforward. Nevertheless, it does have some software dependencies:

- **Python2.6 or later** (it has not been tested on Python 3.X), including development libraries.
- **Platypus** (**http://www.well.ox.ac.uk/platypus**), which in turn requires **htslib** (**http://www.htslib.org**), which in turn requires **zlib** (**http://zlib.net**).
- **VCFtools** (**https://vcftools.github.io**), for using the `vcf-sort` command.
- The **tabix** package (**http://sourceforge.net/projects/samtools/files/tabix**), for using the `tabix` and `bgzip` commands. It should come together with htslib.

2.2 Installing Somatypus

Information on how to install the dependencies and the pipeline itself is provided in the INSTALL.txt file, as well as in the project's **GitHub page**. These instructions assume that you have administrator privileges in the system you are using (i.e. that you can do "sudo"); otherwise, you should contact you system administrator.

3. Description of the pipeline

3.1 Overview

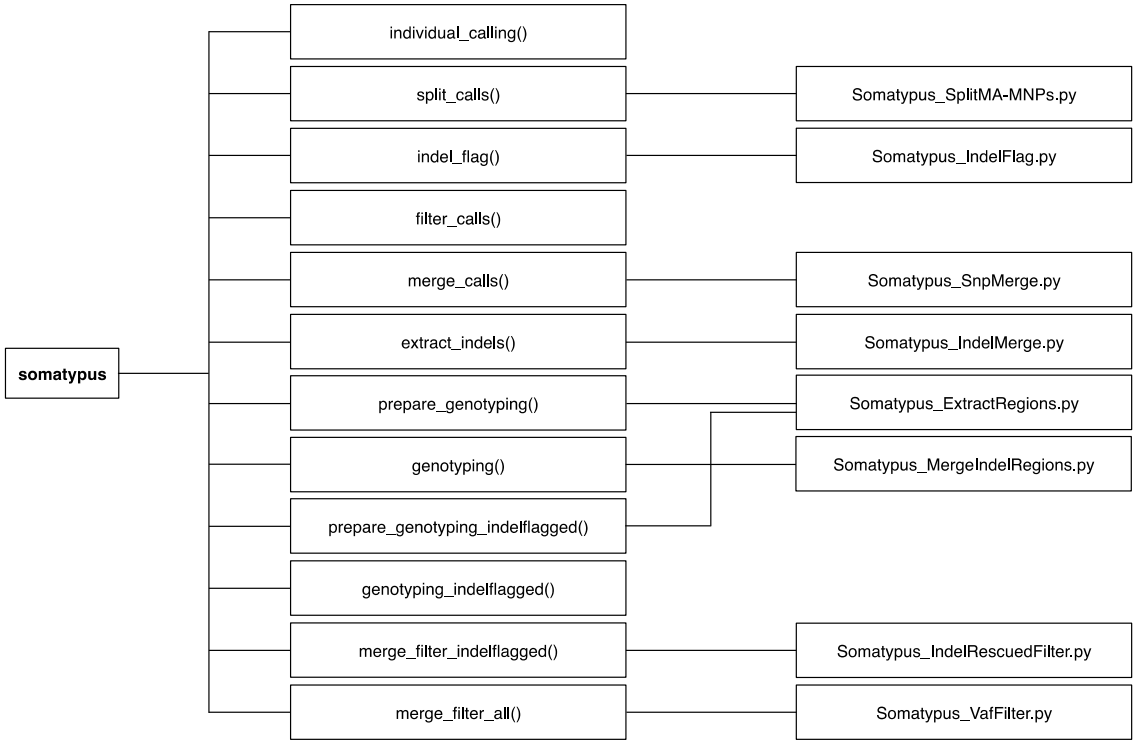
Somatypus is designed to extract a high-confidence set of variants (in VCF format) from a collection of input sequence alignment files (in BAM format, one file per sample) and a reference genome (in FASTA format). It can also take a set of regions wherein to perform the calling, specified in a text file. The input BAM and FASTA files must be accompanied by .bai and .fai index files, respectively. These can be generated by indexing the input files with the commands `samtools index` and `samtools faidx`, respectively.

The pipeline is structured in a modular way: one core Bash shell script, named *somatypus*, is directly called by the user, processes the input and controls the overall

execution. This script contains 12 functions, each of which manages one step of the processing workflow; some functions will be called more than once, so there are more pipeline steps than functions. Some of these steps demand operations involving statistical computation and file parsing and editing; these are performed by a set of 8 Python scripts, which are called by the Bash functions when necessary.

The major advantage of this kind of modular organisation is that, should the workflow experience an interruption before finishing, it can be resumed from the last uncompleted step, thanks to the maintenance of a checkpoint file along the execution.

Below is a diagram of the described structure of scripts and functions. These are arranged, from left to right and from top to bottom, according to their relative execution order in the workflow.



3.2 Running the pipeline

If the pipeline is run via the `somatypus` command without any options (or with `-h`), it will display the following usage information:

```
| SOMATYPUS
| A Platypus-based variant calling pipeline for cancer data
| Version x.x
|
| Required input:
|    -i  Absolute path to folder containing the input BAM files (accompanied by BAI indices).
|    -g  Absolute path to reference genome FASTA file (accompanied by FAI index).
|    -o  Absolute path to the output folder (it will be created if needed).
|
| Optional input:
|    -r  Absolute path to file of regions to use, one per line in CHR:START-END format.
|    -c  Number of CPUs (processes) for Platypus *(should not exceed 8 due to a bug)*.
|    -p  Additional options for Platypus, within quotes and separated by spaces.
|
| Options:
|    -h  Print this usage information and exit.
|    -v  Print version and exit.
|
| Usage:
|    somatypus -i /path/to/bams_dir -o /path/to/out_dir -g /path/to/genome.fna
-r /path/to/regions.txt -c <1-8> -p "--option=VAL --option=VAL"
```

It is advisable that all the input paths be absolute, rather than relative. An optional regions file allows the user to define a set of genomic regions (e.g. exons) wherein to perform the calling. The regions file must be a text file containing one region per line, in CHR:START-END format (e.g. 1:1028676-1028844.Chromosome labels must match those in the reference FASTA and in the sample BAM files).

Specifying the number of CPUs to use is also optional (default is 1) but, if input, it should be a positive integer not above 8 (or even less, depending on the amount of data), due to an inveterate Platypus bug affecting memory allocation (see **Known issues**).

Finally, additional calling options may be passed to Platypus through the `-p` argument. The whole options string must be quoted, and options must be separated by spaces. However, these must not override the options already defined in the pipeline, namely: `--logFileName`, `--refFile`, `--bamFiles`, `--regions`, `--minPosterior`, `--minReads`, `--minFlank`, `--trimReadFlank`, `--source`, `--getVariantsFromBAMs`, `--nCPU`, or `-output` (or `-o`). For obvious reasons, they should neither include `--help` nor `-h`.

The full list of Platypus options can be consulted via:
`Platypus.py callVariants -h`.

The full log of the pipeline execution will be stored in a file named SOMATYPUS_<*date+time*>.log, in the *logs* subfolder of the output directory, together with the log files of most of the steps. The log files and the temporary folders containing intermediate files will be numbered according to the number of the steps that interact with them.

3.3 Pipeline steps

Below is a description of each step involved in the execution of the pipeline, including the names of the scripts and functions that take part in it.

3.3.1 Initialisation

This preliminary step of the workflow is composed by a series of initial validity checks. First, the script checks that the required software dependencies (Platypus, VCFtools, tabix and the Somatypus scripts themselves) are installed and accessible from the command line. Second, the existence of all the input files is verified, as well as the presence of the necessary index (.bai and .fai) files; the value of the argument indicating the number of processors to use is also checked to be greater than 0. Third, the script looks for a checkpoint file (which is created inside the *logs* folder during the execution of the pipeline) and, if it exists (due to a previous execution in the same output folder), it reads from it the name and index of the last completed step of the pipeline, and resumes the execution from the following step. Therefore, an output folder that has already been used in a previous run should be indicated only if said run ended unexpectedly and needs to be resumed.

Scripts: somatypus

3.3.2 Individual variant calling

This first processing step performs variant calling individually on each sample (BAM file). This step actually has two parts, because variants are called under two distinct configurations as a way to maximise sensitivity. There is a checkpoint after each of these steps.

The reason for considering each sample individually, instead of all together, is that Platypus, being a haplotype-based caller, tends to group the variants into haplotypes and look at the distribution of these haplotypes across the “population”, using a Hardy-Weinberg equilibrium model (commonly applied to diploid populations). This causes some rare, low-frequency alleles to be ignored because they are not present in the population in a way that satisfies the model’s expectations. By calling the variants individually in each sample (using special configurations), and then genotyping the filtered set of candidate variants in all the samples at the same time, the pipeline can effectively bypass the restrictions of the model, providing enhanced sensitivity.

Therefore, the aim here is to call as many variants as possible; in subsequent steps, the variants coming from each sample will be filtered, merged into a single VCF file, and genotyped to obtain reliable read counts, genotypes and quality estimates by considering all samples simultaneously; a final filtering step will follow.

The first part of this step runs Platypus on each sample, using settings that are very similar to the default ones, except for two changes:

- The minimum number of reads supporting a variant in a sample for the variant to be called (`--minReads` option) has been increased from 2 to 3. The reason is that we do not consider a variant supported by only two reads to be reliable.
- The minimum posterior probability (as defined in the **Platypus supplementary material**) that a variant must have for being called (`--minPosterior` option) has been reduced from 5 to 0. This allows calling variants that would otherwise not be considered real, because they do not fit a diploid segregation model. Setting the minimum posterior to 0 causes many false positives to be called at this stage, but also rescues many real somatic variants that would otherwise be overlooked.

The second part of the step calls individually on each sample again, but now using an alternative, less stringent configuration. Apart for the two changes noted above, the following options have been altered:

- The option which defines how long groups of SNVs (called MNVs) can be, as well as how many bases at both ends of reads are not considered for calling (`--minFlank`) has been reduced from 10 to 0.
- The option which defines how many bases at both ends of reads are forced to have base quality 0, in order to prevent calling in these unreliable regions (`--trimReadFlank`) has been increased from 0 to 10. The aim is to correct for the side effect of allowing calling variants in the first and last 10 bp of reads, caused by the `--minFlank` option controlling two different settings (see above).

The original purpose of this modification was to call individual SNVs instead of MNVs, making the output less redundant and more reliable. However, it turned out that these relaxed settings are able to rescue variants which would be overlooked by the default settings (together with a large number of false positives that need to be filtered out). The opposite was also true, with some true variants being missed by the alternative settings, so the only apparent solution was to combine the both output sets. Fortunately, the high performance of Platypus allows calling variants twice for every sample without significant time costs, in most cases.

Scripts: somatypus, individual_calling()

Step number: 1, 2

3.3.3 Splitting of multi-allelic and MNV calls

The set of VCF files generated by the individual calling have two features that need to be altered: the grouping of several variants occurring in the same position (and thus sharing a reference allele) into the same file line (henceforth referred to as multi-allelic variants or calls) and the grouping of multiple SNVs into multi-nucleotide variant (MNV) calls. In order to standardise the data before subsequent steps, all the calls are transformed into bi-allelic calls (having only one reference and one alternate allele),

and MNVs are split into their constituent SNV calls, so that every line contains the bi-allelic call of either an SNV or an indel.

For multi-allelic calls, the information which is specific for each alternate allele is easily split between the resulting bi-allelic calls, so they are as accurate as the original call. However, since MNVs are considered a single variant, the data values for each one of the constituent SNVs cannot be inferred from the values corresponding to the whole MNV. As a consequence, all the SNVs coming from a concrete MNV will share the same exact information, which is now imprecise and unreliable, since it applies to the whole MNV. In addition, if one SNV is originally present in more than one variant, the split process will render duplicated SNV calls, each of them having different values (this was the original reason of considering using alternative calling settings). This would pose a problem should the information contained in these calls be final, but this is not the case. We are only interested in the variants themselves, and will obtain new, reliable information at the genotyping stage.

Scripts: somatypus, split_calls(), Somatypus_SplitMA-MNPs.py

Step number: 3

3.3.4 Filtering of individual SNV calls

The variants have to be quality-filtered at this point, because they include a large proportion of low-quality calls that are not of interest and whose further processing is unnecessary. Filtering is performed individually, based on the presence of the following filter flags applied by Platypus. (An extended version of the information shown below can be found in the **Platypus supplementary material**.)

- **badReads**. This filter is triggered when, across the reads supporting a variant, the median of the minimum base quality close to the focal site (default 7 bp either side, configurable via `--badReadsWindow`) is too low (default 15 or less, configurable via `--badReadsThreshold`). This identifies systematic local sequencing issues causing an excess of read errors, which are not always accurately reflected in the read quality scores. It is also triggered when more than a given proportion of reads (by default 0.7, configurable via `--filteredReadsFrac`) are filtered out at the candidate generation stage.

- **MQ**. Platypus computes the root-mean-square mapping quality of all reads covering the variant site, and flags the variant if this value is less than 40 (configurable via`--rmsmqThreshold`). This statistic is computed using all the reads mapping to the region, before any filtering is applied. The default filter threshold works well for Stampy and BWA-mapped reads; for other mappers this threshold may need to be reduced.

- **strandBias**. This filter identifies variants whose support is skewed in terms of reads mapping to the forward and reverse strands, relative to the distribution seen in all reads. Platypus uses the distribution seen overall, rather than say a binomial distribution centred around a fraction of 0.5, because certain

experimental designs can give rise to bona fide strand biases. Specifically, the reads supporting the variant are tested against a Beta-binomial distribution with parameters α and β such that the smallest of these is 20, and the parameters are such that the mean of the distribution equals the ratio observed in all reads. Variants are accepted if the *p*-value exceeds 0.001.

- **SC**. Polymerase slippage in low-complexity regions is a known cause of spurious indel calls, and in the alignment model in Platypus accounts for a higher incidence of these errors. However, in certain instances, particularly when two different but substantial low-complexity regions abut, compensating errors in both low-complexity regions can result in sequence that does not support an indel, but does support an artefactual SNV. Although these occurrences are rare, when they occur these false calls are difficult to spot, because the bases can be of high quality. To avoid calling these variants, a sequence complexity statistic is computed that measures the contribution of the two most frequent nucleotides among the 21 around a site; if this measure exceeds 95%, SNV calls are flagged as suspicious.

- **QD**. In order to avoid calling variants using data from many low quality reads, a value (the QD score) is computed that reflects the total evidence in favour of the variant per read supporting the variant. As a proxy for the total evidence, Platypus uses the Phred-scaled variant posterior as reported in the QUAL field of the VCF file. To avoid a downward bias in low-coverage samples, the Phred-scaled prior of the variant is added to this. The resulting score is divided by the number of reads that support the variant. Variants with a QD value lower than 10 (configurable via `--qdThreshold`) are flagged as suspicious.

Calls showing any of the flags above are discarded from the individual VCF file; since a variant will be often found in many individual files, it would need to be flagged in all of them in order to be entirely filtered out before the subsequent merging step, thus making this initial filtering very lax.

Given the particularities of cancer sequence data, we allow the following flags to be present in our samples:

- **alleleBias**. This filter identifies variants that show support in too few reads compared to the expectation under heterozygous segregation in a diploid organism.

- **HapScore**. This filter indicates that too many (>4) haplotypes are supported by the data in this region.

- **Q20**. The posterior for calls is calculated in the usual way (as prior times likelihood of the call, divided by the sum of prior times likelihood over all genotypes considered), and reported as a “genotype quality Phred score” (the Phred-scaled probability of the call being wrong, or 10 times negative 10-log of one minus the posterior) in the per-sample GQ field. All variants with posteriors below 20 are flagged as suspicious. (This flag must be allowed for coherence

with the `--minPosterior` modification introduced in order to call variants with very low posteriors.)

The remaining flags mentioned in the Platypus documentation (GOF, hp10, QualDepth, and REFCALL) were not found in any of our variant sets, and are almost certainly not applied in the latest Platypus version, so they have not been considered here. The PASS flag indicates that the variant did not trigger any quality filter.

Scripts: somatypus, filter_calls()

Step number: 4

3.3.5 Identification of SNVs near indels

Visual assessment of randomly sampled variants during the pipeline design revealed that many false positives in the resulting variant set were caused by the presence of nearby indels, which shifted the flanking sequence in a way that was not be detected by the aligner in regions close to the read ends. As a consequence of this, what should have been called as an indel (and sometimes was, in a proportion of the reads), had instead been called as a series of spurious SNVs. To avoid this, those SNVs located at 5 or less bp from any end of an indel (not just from the position where it is called), in the same sample where the indel is found, are added to an exclusion list, in order to exclude them from the standard genotyping stage; instead, they will be genotyped and filtered separately, using more-stringent settings. These variants are hereafter referred to as *indel-excluded* SNVs.

Scripts: somatypus, indel_flag(), Somatypus_IndelFlag.py

Step number: 5

3.3.6 Merging of individual SNV calls

In this step, the filtered SNV calls coming from each sample are merged into a single VCF file, so that the same chromosome (CHROM), position (POS), reference allele (REF) and alternate allele (ALT) values will not be present more than once in the merged VCF. The exclusion list created in the indel-flagging step is used here to prevent SNVs located near indels from being included in the merged file; they are instead gathered in a different VCF file for separate genotyping.

In order to avoid the creation of multi-allelic calls — caused by more than one substitution occurring at the same genomic position — during the genotyping stage, up to three merged VCFs are generated, so that if there are three different SNVs occurring at the same location, each of them will be considered as a different allele and output to a different merged file. In consequence, this step's output includes one "allele 1" VCF, one "allele 2" VCF and one "allele 3" VCF, as well as other three analogous files for the indel-excluded SNVs. This is aimed at preventing different variants from sharing certain statistics, and more importantly, sharing Platypus filter flags (which are assigned to each call, not each variant). This could cause some false positives to pass the quality filters because they are called together with a true variant, acquiring its "good" flags.

It is evident that, since each call in the merged VCF comes from a particular sample, only the CHROM, POS, REF and ALT values are of interest; the other fields would be valid only for the individual sample in which the variant was called. Because only the genomic position and base change of SNVs need to be known at this stage, this is not a concern.

Scripts: somatypus, merge_calls(), Somatypus_SnpMerge.py

Step number: 6

3.3.7 Identification of high-confidence indels

Now, indels are extracted from the individual VCFs and merged into a new VCF file. When selecting indels, a more stringent approach is followed: only indel calls which are bi-allelic (their calls having only one alternate allele) and fulfil the quality criteria defined in the individual filtering step, in terms of Platypus flags (see **Filtering of individual SNV calls**),are extracted from the original, individual VCF files, and merged according to genomic position and sequence change. The set is also examined for indels that share the same position but occur in different samples, thus never sharing position in a same VCF file; these are also removed.

Scripts: somatypus, extract_indels(), Somatypus_IndelMerge.py

Step number: 7

3.3.8 Genotyping preparation

This step involves the generation of the VCF and regions files that are needed in the genotyping of SNVs and indels. First, the four merged VCF files corresponding to the three SNV “alleles”, as well as the merged indels, are sorted, compressed and indexed, in order to be readable by Platypus. Then, if the user did not specify a file of genomic regions, a set of regions is defined for genotyping that includes a window of ±200 bp around each variant in the merged VCF files. This is done in order to substantially improve the performance of genotyping in whole genomes, since only the regions around target variants will be examined. After this, a new regions file will be made for every merged VCF (including indels), including only the regions that contain one or more variants belonging to the VCF in question (there will be, therefore, four new regions files). This also contributes to speeding the genotyping up, since only the relevant regions will be examined. Finally, a new file containing the paths to each of the input BAM files is created, which is needed as an input to Platypus.

Scripts: somatypus, prepare_genotyping(), Somatypus_ExtractRegions.py

Step number: 8

3.3.9 SNV genotyping

Genotyping of SNVs is divided in three steps (with checkpoints between them), corresponding to the genotyping of the variants belonging to alleles 1, 2 and 3. These

three Platypus runs are executed serially; however, the run time of the genotyping for alleles 2 and 3 will expectably be much shorter than for allele 1, because of the considerably smaller number of variants to examine. The settings applied for the genotyping are the same as in the first individual calling step (`--minReads=3`, `--minPosterior=0`). However, now Platypus does not need to call variants from the BAM files (`--getVariantsFromBAMs=0`), instead reading the variants directly from the (sorted and compressed) merged VCF file of the corresponding allele (`--source` option). Thus, instead of calling variants *de novo*, the merged variants are genotyped in every sample, generating three new VCF files.

However, genotyping is often convoluted when employing cancer data. Because some of the variants present in the input merged VCF will not fit well in the population model used by Platypus (one of the reasons for making the calling individual to each sample), they will not be included in the output genotyped VCF. (The percentage of these missing variants after the genotyping is normally below 5%.) In order to rescue these variants, a second, fast Platypus run is done for each allele, where the regions to examine correspond to the exact positions occupied by the missing variants(these are obtained by comparing the merged and genotyped VCFs).This second genotyping, which is performed only if missing variants are detected, generates up to three additional VCF files.

Scripts: somatypus, genotyping()

Step number: 9, 10, 11

3.3.10 Indel genotyping

This step is analogous to the previous one, with the difference that the input variants are taken from the merged indel VCF, and the output goes to (usually) two genotyped indel VCFs.

Scripts: somatypus, genotyping(), Somatypus_MergeIndelRegions.py

Step number: 12

3.3.11 Genotyping preparation for SNVs near indels

After genotyping of “normal SNVs” and indels, separate genotyping of the SNVs located near indels (*indel-excluded*) is performed. This set of excluded SNVs has been observed to contain a significant proportion of true positives, so a special workflow has been devised to rescue these real variants located. A small proportion of false positives may result from this process, but these are normally cases where only human inspection can identify them as artefacts caused by indels or sequencing noise.

Preparations for genotyping involve the same steps for indel-excluded SNVs as for the other variant types, with the only difference that the VCF files to process are the ones containing the indel-excluded SNVs.

Scripts: somatypus, prepare_genotyping_indelflagged(), Somatypus_ExtractRegions.py

Step number: 13

3.3.12 Genotyping of SNVs near indels

The genotyping of indel-excluded SNVs is identical to the standard SNV genotyping (except for the input and output files), comprising three Platypus executions. As in the previous genotyping step, missing variants are identified and re-genotyped, so that the output is normally made up of two genotyped VCFs per allele.

Scripts: somatypus, genotyping_indelflagged()

Step number: 14, 15, 16

3.3.13 Merging and filtering of SNVs near indels

After all the variants have been genotyped, the indel-excluded SNVs in the (up to) six VCFs resulting from genotyping are merged into a single file. Then, they are quality-filtered using the same criteria as in the individual VCF filtering (see **Filtering of individual SNV calls**).Finally, a specific filter is applied, which discards SNVs with median read coverage below 20, or median VAF (variant allele fraction, the fraction of reads supporting the variant at its locus) below 0.2 or above 0.9. The median VAF is computed considering only samples with at least three reads supporting the examined variant. These thresholds have been designed to filter out most false positives in the indel-excluded set, and have proven to be especially effective.

Scripts: somatypus, merge_filter_indelflagged(), Somatypus_IndelRescuedFilter.py

Step number: 17

3.3.14 Merging and filtering of all variants

The last step involves merging all the variants (those coming from the up to 8 genotyped VCFs generated through “normal” genotyping, and the filtered indel-excluded SNVs) into two files, corresponding to SNVs and indels. These are subsequently quality-filtered using the same criteria as in the individual VCF filtering (see **Filtering of individual SNV calls**).An additional filter discards those variants with VAF above 0.9 in *all* the samples, since these are almost certainly sequencing artefacts. Finally, the output VCFs are sorted and the pipeline finishes.

Scripts: somatypus, merge_filter_all(), Somatypus_VafFilter.py

Step number: 18

3.4 Output

The output variant set is composed of the files **Somatypus_SNVs_final.vcf** and **Somatypus_Indels_final.vcf**, located in the output directory. The remaining files and folders are temporary and can be deleted if not needed.

The VCF files generated by Platypus begin with a 48-line header. The last line in the header contains the names of the data columns. From line 49 onwards, each line in the VCF contains a single variant, which is described by the following fields.

- **CHROM**. Chromosome where the variant is located.
- **POS**. Genomic position of the variant within the chromosome.
- **ID**. Variant identifier. In Platypus calls, it is always a dot.
- **REF**. Reference allele.
- **ALT**. Alternate allele.
- **QUAL**. Numeric quality value of the variant, corresponding to its Phred-scale posterior. (The Platypus documentation discourages the use of this field.)
- **FILTER**. Platypus filter flag. Only PASS, Q20, alleleBias and HapScore flags are accepted in the pipeline (see **Filtering of individual SNV calls**).
- **INFO**. Contains statistical information about the variant. The meaning of each value in this field is described in the VCF header.
- **FORMAT**. Defines the order and meaning of each of the values that make up the sample data. In this case, the sample data are always displayed as strings with 6 sub-fields, delimited by colons: GT:GL:GOF:GQ:NR:NV. Each sub-field is described below.
    - **GT**. Unphased genotype (usually, 0/0, 0/1, 1/0 or 1/1). Since the somatic variants coming from cancer samples are not necessarily bi-allelic, the use of this field is discouraged.
    - **GL**. Genotype log10-likelihoods for ref-ref, ref-alt and alt-alt genotypes. Since the somatic variants coming from cancer samples are not necessarily bi-allelic, the use of this field is discouraged.
    - **GOF**. Goodness-of-fit value. (We have not found this value to be actually indicative of variant quality.)
    - **GQ**. Genotype quality as Phred score. Since the genotype information can be inaccurate for cancer variants, the use of this field is discouraged.
    - **NR**. Total number of reads (depth coverage) at the variant position.

    - **NV**. Number of reads supporting the variant at the variant position. The VAF is computed as NV/NR.

- **Sample data**. After the FORMAT field, each column contains the data of the variant for one sample (identified by the column name), in the format described in the FORMAT field.

4. After running Somatypus

Although Somatypus has been specifically designed to call variants in unpaired samples (because there is no “normal” sample available in transmissible cancers), its main downfall is its inability to distinguish between tumour and normal/host samples, and therefore, between germline variants (or variants arising from cross-sample contamination) and somatic mutations. This must be done in downstream processing steps; a Python script and an R script have been included in the *somatypus-x.x/utils* directory, which perform:

- Extraction of the most valuable data from the VCF files, which substantially reduces the time required for downstream computations (**ExtractVcfData.py**). After installation of the pipeline, this script can be run directly from the command line, receiving as its only argument the path to the input VCF file. It needs to be run separately for the SNVs VCF and for the indels VCF.

- Import of these data into R; the decomposition of the variant set into variants present in any normal/host sample, and variants present exclusively in tumours(having into account contamination between samples); and the plotting of basic VAF and coverage graphs (**BasicManipulation.R**). For this code to work, the tumour samples — unlike the normal/host samples — must contain a “T” letter in their labels (VCF column names). This script is just an example of how to differentiate between the two types of samples, how to adjust for sample contamination, and how to determine which variants are present exclusively in tumours. It should be run step-by-step from an IDE like RStudio, not directly from the command line, because it does not save the resulting data objects; in order to have easy access to them for subsequent analyses in R, they could be saved into an RData file.

5. Change log

The log of changes related to each Somatypus release is provided in the Changelog.txt file, within the *docs* folder.

6. Known issues

Although Platypus is an outstanding variant caller, it is not exempt of bugs and issues. Here are listed the ones that we have encountered during the development and use of Somatypus (using Platypus 0.8.1):

- **The sample names must not overlap.** If this is the case, a bug in Platypus will cause an error, finely reported by the message "**Something is screwy here**". This message necessarily means that some sample names overlap; for example, there can be a sample called 622-T, which overlaps with another called 22-T. In that case, 22-T (and the rest of possibly conflicting samples) should be renamed to something like 022-T. Yet another difficulty lies in knowing where Platypus takes the sample name from. Normally, it will read it from the BAM header ("SM" field), meaning that the files need to be re-headered using SAMtools or an equivalent package. However, if Platypus finds any field in the SAM header which it cannot interpret (this will be recorded at the beginning of the Platypus log), it will take the sample name from the BAM file name. In that case, the files just need to be renamed, which is easier.

- If the number of CPUs exceeds 8, Platypus might try to allocate an extremely excessive memory amount, which will make the involved sub-process to be killed, but will probably not cause the execution to finish (however, a number of genomic intervals will be missed). This surreptitious bug can be spotted by searching for the word "memory" (using `grep -i "memory"`) in the SOMATYPUS_xxxxxxxxxx.log file. If the bug did not occur, no matches will be found; otherwise, you will find an error line including the words "**Cannot allocate memory**". This bug is unlikely to happen if the number of CPUs does not go above 8 (although it seems to become more common as the amount of sequence data growths).

- The message "**Too many reads**" in the Platypus log means that there are more than 5,000,000 reads in one of the calling regions (the region will be indicated in the message). This might indicate a repetitive region displaying very high sequencing coverage, and it is perhaps advisable not to call variants in such regions. In order to allow calling in this kind of regions, the `-p` option can be used to change the options `--maxReads` and `--bufferSize`.

- If the pipeline stops at the genotyping stage because of an error containing the words "**Check that index exists**", this is not actually related to a problem with an index file, but a result of an insufficient amount of memory. This error may occur when using a very large amount (above 1 TB) of input sequence data and should disappear with increased memory.

Should you find any other bug or problem while using Somatypus, please do not hesitate in contacting the developer (ab2324@cam.ac.uk**).**