MIP_api Documentation

Release 2.0

Måns Magnusson, Robin Andeer

Sep 27, 2017

Contents

Release 2.0.

MIP is a pipeline for clinical analysis of whole exome and whole genome sequence data.

Contents:

MIP enables identification of potential disease causing variants from sequence data.

CHAPTER 1

Overview

MIP performs whole genome or target region analysis of sequenced single-end and/or paired-end reads from the Illumina plattform in fastq(.gz) format to generate annotated ranked potential disease causing variants. MIP performs QC, alignment, coverage analysis, variant discovery and annotation, sample checks as well as ranking the found variants according to disease potential with a minimum of manual intervention. MIP is compatible with Scout for visualization of identified variants.

CHAPTER 2

Features

- **Autonomous**
  - Checks that all dependencies are fulfilled before launching
  - Builds/downloads references and/or files lacking before launching
  - Splits and merges files for samples and families when relevant
- **Automatic**
  - A minimal amount of hands-on time
  - Tracks and executes all module without manual intervention
  - Creates internal queues at nodes to optimize processing
  - Minimal IO between nodes and login node
- **Flexible:**
  - Design your own workflow by turning on/off relevant modules
  - Restart an analysis from anywhere in your workflow
  - Process one, or multiple samples using the module(s) of your choice
  - Supply parameters on the command line, in a pedigree file or via config files
  - Simulate your analysis before performing the actual analysis
  - Redirect each modules analysis process to a temporary directory (@nodes or @login)
  - Limit a run to a specific set of genomic intervals
- **Fast**
  - Analyses an exome trio in approximately 6 h
  - Rapid mode analyzes a WGS sample in approximately 4 h using a data reduction and parallelization scheme
- **Traceability**

  - Recreate your analysis from the MIP log
  - Logs sample meta-data and sequence meta-data
  - Logs version numbers of softwares and databases
- **Standardized**
  - Use standard formats whenever possible
- **Visualization**
  - Output is directly compatibel with Scout

CHAPTER 3

Example Usage

```
perl mip.pl -pMosaikBuild 0 -configFile 1_config.yaml
```

CHAPTER 4

Getting Started

Installation

MIP is written in Perl and therfore requires that Perl is installed on your OS (See *Installation*).

Prerequisites

MIP will only require prerequisites when processing a modules that has dependencies (See *Setup*). However, some frequently used sequence manipulation tools e.g. samtools, PicardTools, Bedtools are probably good to have in your path.

Meta-Data

Meta data regarding the pedigree, gender and phenotype should be supplied for the analysis.

- Pedigree file (PLINK-format; See *Pedigree File* & MIP´s github repository).
- Configuration file (YAML-format; See *Dynamic Configuration File* & MIP´s github repository).

Usage

MIP is called from the command line and takes input from the command line (precedence), a config file (yaml-format) or falls back on defaults where applicable.

Lists are supplied as comma separated input, repeated flag entries on the command line or in the config using the yaml format for arrays.

Note: List or repeated entries need to be submitted with the same order for each element across all supplied lists.

Only flags that will actually be used needs to be specified and MIP will check that all required parameters and dependencies (for these flags only) are set before submitting to SLURM.

Program parameters always begins with "p" followed by a capital letter. Program parameters can be set to "0" (=off), "1" (=on) and "2" (=dry run mode). Any program can be set to dry run mode and MIP will create sbatch scripts, but not submit them to SLURM for these modules. MIP can be restarted from any module, but you need to supply previous dependent programs in dry run mode to ensure proper file handling.

MIP will overwrite data files when reanalyzing, but keeps all "versioned" sbatch scripts for traceability.

MIP allows individual target file calculations if supplied with a pedigree file or config file containing the supported capture kits for each sample.

You can always supply `perl mip.pl -h` to list all available parameters and defaults.

Example usage:

```
$ perl mip.pl -f 3 -sampleid 3-1-1A,3-2-1U -sampleid 3-2-2U -pFQC 0 -pMosaikBuild 2 -
↪pMosaikAlign 2 -c 3_config.yaml
```

This will analyze *family 3* using *three individuals* from that family and begin the analysis with programs *after MosaikAlign* and use all parameter values as specified in the *config file*, except those supplied on the command line, which has precedence.

Input

MIP requires the input Fastq files to follow a naming convention to accurately and automatically handel individual runs and lanes (See *Setup*).

Fastq files (gziped/uncompressed) should be place within the `-inFilesDirs`.

Note: MIP will automatically compress any non gzipped files if `-pGZip` is enabled. All files ending with .fastq or .fast.gz will be included in the run.

All MIP scripts (including mip.pl) should be placed in the script directory specified by `-inScriptDir`.

All references and template files should be placed directly in the reference directory specified by `-referencesDir`, except for ANNOVAR db files, which should be located in *annovar/humandb*.

Output

Analyses done per individual is found under respective sampleID subdirectory and analyses done including all samples can be found under the family directory.

Sbatch Scripts

MIP will create sbatch scripts (.sh) and submit them in proper order with attached dependencies to SLURM. These sbatch script are placed in the output script directory specified by `-outScriptDir`. The sbatch scripts are versioned and will not be overwritten if you begin a new analysis. Versioned "xargs" scripts will also be created where possible to maximize the use of the cores procecessing power.

Data

MIP will place any generated datafiles in the output data directory specified by `-outDataDir`. All datatfiles are regenerated for each analysis. *STDOUT* and *STDERR* for each program is written in the *<program>/info* directory prior to alignment and in the *<aligner>/<program>info* directory post alignment.

Analysis Types

Currently, MIP handles WES `-at exomes`, WGS `-at genomes` or Rapid analysis `-at rapid` for acute patient(s).

The rapid analysis requires `BWA_MEM` and selects the data that overlaps with the regions supplied with the `-bwamemrdb` flag. MIP will automatically detect if the sequencing run is single-end or paired-end and the length of the sequences and automatically adjust accordingly.

Note: In rapid mode; Sort and index is done for each batch of reads in the `BWA_Mem` call, since the link to infile is broken by the read batch processing. However `pPicardToolsSortSam` should be enabled to ensure correct fileending and merge the flow to ordinary modules.

Project ID

The `-projectID` flag sets the account to which core hours will be allocated in SLURM.

Aligner

Currently MIP officially supports two aligners Mosaik and BWA, but technically supports any aligner that outputs BAM files. Follow the instructions in *Adding a new program* to add your own favorite aligner.

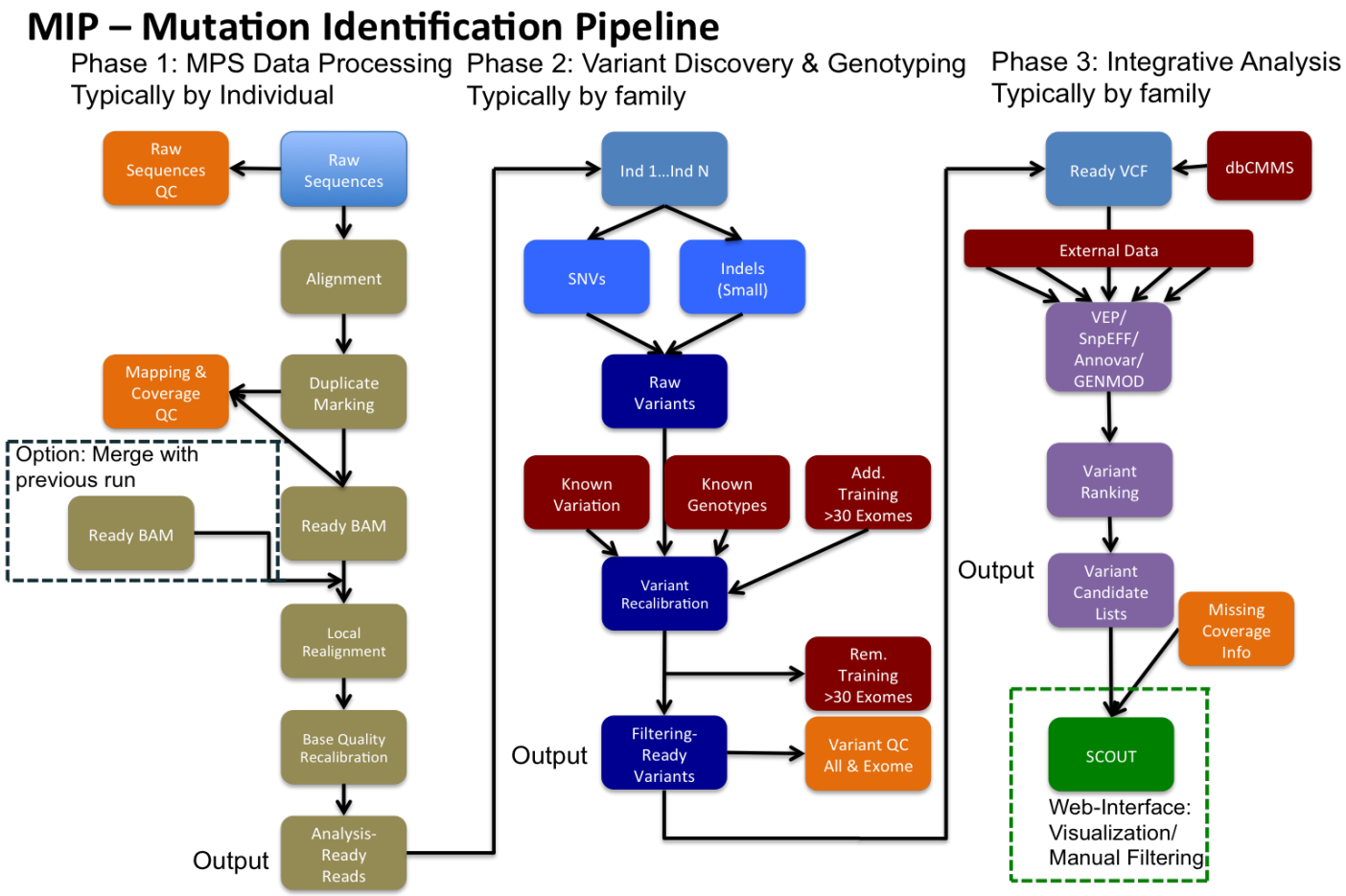
Log

MIP will write the active analysis parameters and *STDOUT* to a log file located in: `{OUTDIRECTORY}{FAMILYID}/{MIP_LOG}/{SCRIPTNAME_TIMESTAMP}`

Information, such as infile, programs, outdatafiles etc, for each analysis run is dynamically recorded in the a yaml file determined by the `-sampleInfoFile` flag. Information in the sampleInfo file will be updated in each analysis run if identical records are present and novel entries are added. The sampleInfo file is used in *QCCollect* to extract relevant qc metrics from the MPS analysis.

Pipeline WorkFlow

This is an example of a workflow that MIP can perform (used @CMMS).


MIP – Mutation Identification Pipeline
Phase 1: MPS Data Processing
Typically by Individual
Phase 2: Variant Discovery & Genotyping
Typically by family
Phase 3: Integrative Analysis
Typically by family
Raw Sequences QC
Raw Sequences
Alignment
Mapping & Coverage QC
Duplicate Marking
Option: Merge with previous run
Ready BAM
Ready BAM
Local Realignment
Base Quality Recalibration
Output
Analysis-Ready Reads
Ind 1...Ind N
SNVs
Indels (Small)
Raw Variants
Known Variation
Known Genotypes
Add. Training >30 Exomes
Variant Recalibration
Rem. Training >30 Exomes
Output
Filtering-Ready Variants
Variant QC All & Exome
Ready VCF
dbCMMS
External Data
VEP/ SnpEFF/ Annovar/ GENMOD
Variant Ranking
Output
Variant Candidate Lists
Missing Coverage Info
SCOUT
Web-Interface: Visualization/ Manual Filtering

CHAPTER 5

Installation

1. Install a fresh copy of Perl

   On UNIX, Perl5 can be installed by following these instructions. It uses Perlbrew.

2. To switch to the new Perl installation, you might need to run:

```
$ INSTALLER_PERL_VERSION=5.16.0
$ perlbrew switch perl-$INSTALLER_PERL_VERSION
```

3. "Install" MIP

```
clone the official git repository
$ git clone https://github.com/henrikstranneheim/MIP.git
$ cd MIP
$ perl mip.pl -h
```

   After this you can decide whether to make MIP an "executable" by either adding the install directory to the `$PATH` in e.g. "`~/.bash_profile`" or move all the files from this directory to somewhere already in your path like "`~/usr/bin`". Remember to make the file(s) executable by `chmod +x file`.

4. Dependencies

   You need to make sure all depedencies are installed and loaded (See *Setup*). However, MIP should tell you if something is missing.

5. To install the dependencies - use cpanm:

```
cpanm <dependency>
$ cpanm YAML
```

CHAPTER 6

Setup

Filename convention

The permanent filename should follow the following format:

```
{LANE}_{DATE}_{FLOW CELL}_{IDN}_{BARCODE SEQ}_{DIRECTION 1/2}.fastq.qz
```

Note: The *familyID* and *sampleID(s)* needs to be unique and the sampleID supplied should be equal to the {IDN} in the filename.

Dependencies

Make sure you have loaded/installed all dependencies and that they are in your `$PATH`. You only need to load the dependencies that are required for the modules that you want to run. If you fail to install dependencies for a module, MIP will tell you what dependencies you need to install (or add to your `$PATH`) and exit. Version after the software name are tested for compatibility with MIP.

Program/Modules

- Perl YAML.pm module and Log::Log4perl.pm since this is not included in the Perl standard distribution
- Simple Linux Utility for Resource Management (SLURM)
- FastQC (version: 0.11.2)
- Mosaik (version: 2.2.24)
- BWA (version: 0.7.10)
- SAMTools (version: 1.1)
- BedTools (version: 2.20.1)

- PicardTools (version: 1.125)
- Chanjo (version: 2.3.0)
- GATK (version: 3.3-0)
- VEP (version: 76)
- vcfParser.pl (Supplied with MIP; see *vcfParser*)
- SnpEff (4.0)
- ANNOVAR (version: 2013-08-23)
- GENMOD (version: 1.7.7)
- Score_mip_variants (version: 0.5.4)
- VcfTools (version: 0.1.12b)
- PLINK (version: 1.07)

Depending on what programs you include in the MIP analysis you also need to add these programs to your `$PATH`:

- FastQC
- Mosaik
- BWA
- SAMTools
- Tabix
- BedTools
- VcfTools
- PLINK

and these to your python `virtualenvironment`:

- Chanjo
- GENMOD
- Score_mip_variants
- Cosmid (version: 0.4.9.1) for automatic download

To make sure that you use the same commands to work on the virtualenvironment, you need to install a virtual environment wrapper. We recommend pyenv and pyenv-virtualenvwrapper. To enable the virualenvwrapper add: `pyenv virtualenvwrapper` to your `~/.bash_profile`.

Databases/References

Please checkout Cosmid to download references and/or databases on your own or via MIP.

MIP can build/download many program prerequisites automatically:

Note: Download is only enabled when using the default parameters of MIP and requires a Cosmid installation in your python virtualenvironment.

Automatic Download:

1. Human Decoy Genome Reference (1000G)

2. The Consensus Coding Sequence project database (CCDS)
3. Relevant references from the 1000G FTP Bundle (mills, omni, dbsnp etc)

Automatic Build:

Human Genome Reference Meta Files:

1. The sequence dictionnary (".dict")
2. The ".fasta.fai" file

Mosaik:

1. The Mosaik align format of the human genome {mosaikAlignReference}.
2. The Mosaik align jump database {mosaikJumpDbStub}.
3. The Mosaik align network files {mosaikAlignNeuralNetworkPeFile} and {mosaikAlignNeuralNetwork-SeFile}. These will be copied from your MOSAIK installation to the MIP reference directory.

BWA:

1. The BWA index of the human genome.

Note: If you do not supply these parameters (Mosaik/BWA) MIP will create these from scratch using the supplied human reference genom as template.

Capture target files:

1. The "infile_list" and .pad100.infile_list files used in {pPicardToolsCalculateHSMetrics}
2. The ".pad100.interval_list" file used in by some GATK modules.

Note: If you do not supply these parameters MIP will create these from scratch using the supplied latest supported capture kit ".bed" file and the supplied human reference genome as template.

ANNOVAR: The choosen Annovar databases are downloaded before use if lacking in the annovar/humandb directory using Annovars built-in download function.

Note: This applies only to the supported annovar databases. Supply flag "–annovarSupportedTableNames 1" to list the MIP supported databases.

CHAPTER 7

Adding a new program

You need to perform a series of tasks to properly add a program to MIP. An overview of the steps can be found here:

1. *Call DefineParameters*
2. *Command line arguments in GetOptions*
3. *if-block run checker in MAIN*

1. Print program name to `MIPLOGG` and `STDOUT`
2. Call your custom subroutine (ses below) with relevant parameters

4. *Custom subroutine*

1. Writes SBATCH headers
2. Figure out i/o files
3. Builds out the body of the SBATCH script
4. Calls *FIDsubmitJob*

More details follow below. Chanjo, a program which is part of the coverage analysis, will be used as an example.

Call DefineParameters

This subroutine takes a number of input parameters. There are basically three parameter types: "program", "file", and "attribute". Try to group your parameter definitions with related programs.

```
DefineParameters("pChanjoBuild", "program", 1, "MIP", 0, "nofileEnding",
↪"CoverageReport");

DefineParameters("chanjoBuildDb", "path", "CCDS.current.txt", "pChanjoBuild", "file");

DefineParameters("pChanjoCalculate", "program", 0, "MIP", 0, "nofileEnding", "MAIN");

DefineParameters("chanjoCalculateCutoff", "program", 10, "pChanjoCalculate", 0)
```

Table 7.1: DefineParameters - parameters

Parameter	Example	Description
Name	pChanjoBuild	Program names start with 'p' by convention, otherwise it's up to you.
Type	program	Can be either program or path.
Default	1	**Program**: 1/0 as on/off, **file**: <path to file> or 'nodefault', **attribute**: e.g 10 or 'nodefault'
Associated program	MIP	Typically the program that calls this program. **program**: usually MIP, **file/attribute**: <Name>.
Exists check	0	Perform a check that a file is in the reference directory. Either: 0, 'file', 'directory'.
File ending	nofileEnding	File ending when module is finished. MIP uses this to determine input files downstream in the Chain. **file/attribute**: skip.
Chain	MAIN	The chain to which the program belongs to. **file/attribute**: skip.
Check install	chanjo	The program handle to check whether it is in the `$PATH`. **file/attribute**: skip.

Command line arguments in *GetOptions*

This is the method that parses the command line input and stores the options. To add your own defined parameters you need to add lines like this:

```
'<short_option>|<long_option>:<s(tring)/n(umber)>' => \$parameter{'<long_option>'}{
↪'value'},
```

You should replace anything that looks like `<placeholder>`:

```
'pCh|pChanjoBuild:n' => \$parameter{'pChanjoBuild'}{'value'},  # ChanjoBuild coverage␣
↪analysis
'chbdb|chanjoBuildDb:s' => \$parameter{'chanjoBuildDb'}{'value'},  # Central SQLite␣
↪database path
'pCh_C|pChanjoCalculate:n' => \$parameter{'pChanjoCalculate'}{'value'}, # Chanjo␣
↪coverage analysis
'chccut|chanjoCalculateCutoff:n' => \$parameter{'chanjoCalculateCutoff'}{'value'}, #␣
↪Cutoff used for completeness
```

Again, program options begin with a leading "p" by convention. Make sure you don't cause any naming conflicts.

Lists can also be specified with a special syntax. Basically you need to assign the option to an array instead of `$scriptParameters`.

```
'ifd|inFilesDirs:s'  => \@inFilesDirs, #Comma separated list
```

Later in your code when you would like to access those values you would join on ",".

```
@inFilesDirs = join(',', @inFilesDirs);
```

Note: MIP doesn't use True/False flags, all options take at least one argument. For program options it's possible to turn on (1), off (0) and run programs in dry mode (2). All program options should specify "n(umber)" as argument type.

if-block run checker in MAIN

The if-block checks whether the program is set to run but it also has a number of additional responsibilities.

Perhaps the most important is to define dependencies. This is done by placing your if-statement after the closest upsteam process to yours. ChanjoBuild, for example, needs to wait until *PicardToolsMarkDuplicates* has finished processing the BAM-files before running.

```
# Closest upsteam dependency for Chanjo
if ($scriptParameter{'pPicardToolsMarkduplicates'} > 0) {
  # Body...
}

# This is where Chanjo fits!
if ($scriptParameter{'pChanjoBuild'} > 0) {
  # Body...
}
```

Next (inside the if-block) it should print an announcement to two file handles:

```
for my $fh (STDOUT, MIPLOGG) { print $fh "\nChanjoBuild\n"; }
```

Lastly it should call a *Custom subroutine*, e.g. for each individual sample or per family, which will write a SBATCH script(s), submit them to SLURM, which executes the module.

Note: `$sampleInfo` is a hash table storing sample information, for example filename endings from different stages of the pipeline. It's used to determine input filenames for your program.

Custom subroutine

First up, let's choose a relevant (and conflict free) name for our subroutine.

```
sub ChanjoBuild {
  # Body...
}
```

If we pass ALL nessesary variables into the subroutine and assign them as scoped variables it's easy to overview variables used inside.

```
my $sampleID = $_[0];
my $familyID = $_[1];
my $aligner = $_[2];
# etc ...
```

a) SBATCH headers

SBATCH headers are written by the *ProgramPreRequisites* subroutine. It takes a number of input arguments.

```
ProgramPreRequisites($sampleID, "ChanjoBuild", "$aligner/coverageReport", 0,␣
↪*CHANJOBUI, 1, $runtimeEst);
```

Table 7.2: ProgramPreRequisites - paramaters

Parameter	Example	Description
Directory	11-1-1A	Either a sample ID (e.g. IDN) or family ID depending on where output is stored.
Program	chanjo	Used in SBATCH script filename.
Program directory	`$aligner/coverageReport`	Defines output directory under *Directory*. Path should include current aligner by convention.
Call type	0	Options: *SNV*, *INDEL* or *BOTH*. Can be set to: 0 ???
File handle	`*CHANJO`	The program specific file handle which will be written to when generating the SBATCH script. Always prepend: '*'.
Cores	1	The number of cores to allocate.
Process time	1.5	An estimate of the runtime for the particular sample in hours.

b) Figure out i/o files

It's up to you to figure out where your program should store output files. Basically you need to ask yourself whether putting them in the family/sample foler makes the most sense.

It's a good idea to first specify both in- and output directories.

```
my $baseDir = "$outDataDir/$sampleID/$aligner";
my $inDir = $baseDir;
my $outDir = "$baseDir/coverageReport";
```

If you depend on earlier scripts to generate infile(s) for the new program it's up to you to figure out the closest program upstream. After that you can ask for the file ending.

```
my $infileEnding = $sampleInfo{ $familyID }{ $sampleID }{'pPicardToolsMarkduplicates'}
↪{'fileEnding'};
```

`$sampleInfo` is a hash table in global scope.

MIP supports multiple infiles and therefore MIP needs to check if the file(s) have been merge or not.This is done with the *CheckIfMergedFiles* subroutine, which returns either a 1 (files was merged) or 0 (no merge of files)

```
my ($infile, $mergeSwitch) = CheckIfMergedFiles($sampleID);
```

Note: `$infilesLaneNoEnding` is a global hash table containing information about the filename-bases (compare filename-endings).

c) Build SBATCH body

This is where you fit relevant parameters into your command line tool interface. Print everything to the file handle you defined above.

```
print CHANJOBUI "
# ------------------------------------------------------------
#  Create a temp JSON file with exon coverage annotations
# ------------------------------------------------------------\n";
print CHANJOBUI "chanjo annotate $storePath using $bamFile";
```

```
print CHANJOBUI "--cutoff $cutoff";
print CHANJOBUI "--sample $sampleID";
print CHANJOBUI "--group $familyID";
print CHANJOBUI "--json $jsonPath";

# I'm done printing; let's drop the file handle
close(CHANJOBUI);
```

Note: A `wait` command should be added after submitting multiple processes in the same SBATCH script with the `&` command. This will ensure SLURM waits for all processes to finish before quitting on the job.

d) Call *FIDSubmitJob*

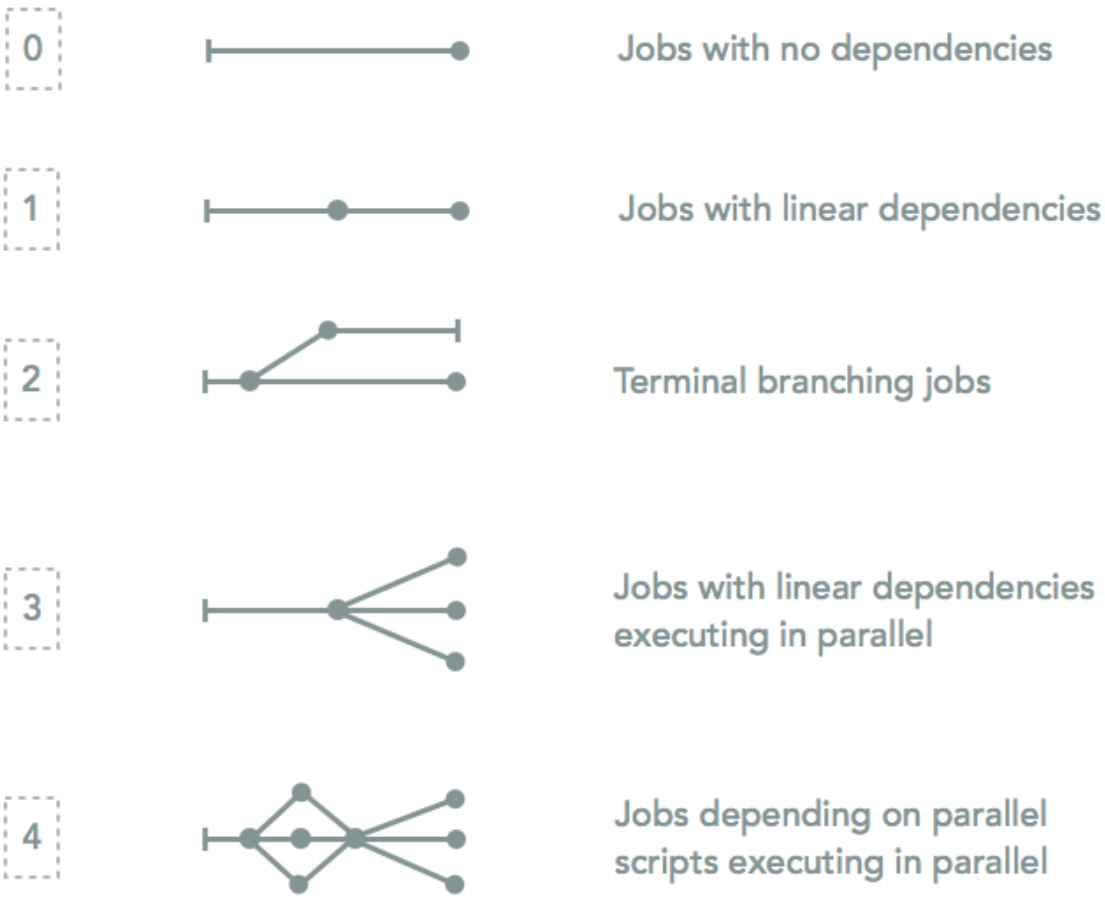
This subroutine is responsible for actually submitting the SBATCH script and handling dependencies. You should only call this if the program is supposed to run for real (not dry run).

```
if ( ($runMode == 1) && ($dryRunAll == 0) ) {
  # ChanjoBuild is a terminally branching job: linear dependencies/no follow up
  FIDSubmitJob($sampleID, $familyID, 2, $parameter{'pChanjoBuild'}{'chain'},
↪$filename, 0);
}
```

Table 7.3: FIDSubmitJob - paramaters

Parameter	Example	Description
Sample ID	11-1-1A	The sample ID/person IDN
Family ID	11	The family ID
Dependency type	2	Choose between type 0-4 (see below)
Chain key	$parameter{'pChanjo'}{'chain'}	The chain defined in *DefineParameters*
SBATCH filename	`$filename`	Always use this variable. It automagically points to your SBATCH script file.
Script tracker	0	Huh? Something about parallel processes...

To figure out which option (integer) to supply as the third argument to *FIDSubmitJob* you can take a look at this illustration.



Note: `$filename` is a variable that is created in *ProgramPreRequisites*. It points to your freshly composed SBATCH script file and should be supplied to *FIDSubmitJob* by all custom subroutines.

Note: `$parameter{'pChanjoBuild'}{'chain'}` is just the chain that you set in *DefineParameters*. In this case we could've replaced it with "MAIN".

Further information

For your convinience a template program module can be found in the project folder hosted on GitHub. [ADD LINK TO TEMPLATE]

CHAPTER 8

Structure

mip.pl

Central hub and likely the only script most users will ever interact directly with.

```
$ echo "Running MIP on Uppmax, analyzing all samples in family 10"
$ mip.pl -c CMMS_Uppmax_config.yaml -f 10
```

Sequence QC

Raw sequence quality control: FastQC

Alignment

Currently MIP supports these aligners:

1. Mosaik (WES, WGS)
2. BWA (WES, WGS, Rapid WGS)

BAM file manipulation

- Sorting and indexing: PicardTools (SortSam)
- Duplicate marking: PicardTools (MarkDuplicates)
- Realignment and base recalibration: GATK (Realigner & BaseRecalibration)

Coverage QC

- Coverage Report and QC metrics: Chanjo & BedTools
- QC metrics: PicardTools (MultipleMetrics & HSmetrics)

Variant calling

- Variant discovery and recalibration: GATK (HaploTypeCaller, GenoTypeGVCFs & VariantRecalibration)

Variant QC

- All variants: GATK (VariantEval)
- Exonic variants: GATK (VariantEval)

Variant Selection

Select transcripts that overlap a gene list: *vcfParser*

Variant annotation

Collect transcript and amino acid information and information from external databases as well as annotation of inheritance models: VEP, *vcfParser*, SnpEff, ANNOVAR, GENMOD

Variant evaluation

Score and rank each variant using weighted sums according to disease causing potential: Score_mip_variants (see *score_mip_variants*)

qcCollect.pl

Collects QC data from the MPS analysis in YAML format. (see *QCCollect*).

covplots_exome.R / covplots_genome.R

Plots coverage across chromosomes.

CHAPTER 9

vcfParser

Parses vcf files to reformat/add INFO fields and metaData headers and/or select entries belonging to a subgroup e.g. a list of genes. Input can be piped or supplied as an infile.

Usage

```
vcfParser.pl infile.vcf > outfile.vcf
```

```
vcfParser.pl infile.vcf --parseVEP 1 -rf External_Db.txt -rf_ac 3 -sf genes.
v1.0.txt -sf_mc 3 -sf_ac 3,4,11,15,17,20 -sof selected_genes.vcf > outfile.vcf
```

Installation

vcfParser is written in Perl, so naturally you need to have Perl installed. The perl module Set::IntervalTree is required and are used to add "ranged" annotations.

VEP

Parses the output from VEP to include RefSeq transcripts. The transcript and protein annotations, moste severe consequence and gene annotations are also included in the output . Transcript protein predictions (Sift and Polyphen) can also be included.

Select Mode

A list of genes and their corresponding HGNC Symbol can be used to fork the analysis into "selected" genes and "orphan" genes.

GuideLines on format for database of genes

- The database file should contain a header line starting with "#".
- The number of headers should match the number of field elements for each entry.
- Do not use whitespace in headers.
- Do not use ";" in file.
- Separate elements in fields with ",". Do not use ", ".
- No whitespace in the beginning or end within fields.
- No entries should be duplicated within database.
- Length of gene coordinates should be greater than 0
- Only digits in gene coordinate entries

Range Annotations

vcfParser can also add range annotations to the vcf by using the Set::IntervalTree perl cpan module and a file with chromosomal coordinates and features to be annotated.

CHAPTER 10

QCCollect

Collects information on MPS analysis from each analysis run. Uses YAML files for input and output. QCCollect uses a yaml file for matching the outdata produced in each run to another yaml file with regular expression used to actually collect the data from the output files. The collected data is then written to disc in yaml format.

MIP produces a sampleInfo yaml file, containing all sample and family information used in each analysis run.

Usage

```
perl qcCollect.pl -si [SampleInfoFilePath] -r [regularExpressionFilePath] -o
[Outfile]
```

Installation

qcCollect is written in Perl, so naturally you need to have Perl installed.

SetUp

1. The regular expression file needs to be created. The regExp file used at CMMS can be printed from qcCollect using the `-preg & -prego` flags

Table 10.1: qcCollect Parameters

Short/Long	Default Value	Type	Summary
-si/–sampleInfoFile	Na	`String`	The sample info file (Yaml;supply whole path)
-r/–regExpFile	Na	`String`	The regular expresion file (Yaml;supply whole path)
-o/–outfile	"qcmetrics.yaml"	`String`	The output file
-preg/–printRegExp	0	`Integer`	Print RegExp YAML file used at CMMS switch
-prego/–printRegExpOutFile	"qc_regExp.yaml"	`String`	The RegExp YAML outfile
-h/–help	Na	Na	Display help message
-v/–version	Na	Na	Display version

CHAPTER 11

score_mip_variants

Score_mip_variants uses the weighted sum model (WSM) approach to rank the most likely pathogenic variant.

Generally, the higher value the more likely pathogenic variant.

Score_mip_variants uses config files to define the rank model, which enables customized set-up and versioning of rank models.

The WSM uses the following alternatives and weights in rank model "gm_cmms_v1.2":

Rank score range: -25 <= rs <= 21

Consequence

Each alleles variant effect on individual transcripts are evaluated using a rule-based approach defined by SO-terms. The SO-terms themselves are ranked in order of severity and this ranking is used to defined the weight of the consequence alternative. The performance score is based on the most severe consequence within each gene.

Performance value for the SO-terms:

- transcript_ablation = 5
- splice_donor_variant = 4
- splice_acceptor_variant = 4
- stop_gained = 4
- frameshift_variant = 4
- stop_lost = 4
- initiator_codon_variant = 4
- inframe_insertion = 3
- inframe_deletion = 3
- missense_variant = 3

- transcript_amplification = 3
- splice_region_variant = 3
- incomplete_terminal_codon_variant = 3
- synonymous_variant = 1
- stop_retained_variant = 1
- coding_sequence_variant = 1
- mature_miRNA_variant = 1
- 5_prime_UTR_variant = 1
- 3_prime_UTR_variant = 1
- non_coding_exon_variant = 1
- nc_transcript_variant = 1
- intron_variant = 1
- NMD_transcript_variant = 1
- upstream_gene_variant = 1
- downstream_gene_variant = 1
- TFBS_ablation = 1
- TFBS_amplification = 1
- TF_binding_site_variant = 1
- regulatory_region_variant = 1
- regulatory_region_ablation = 1
- regulatory_region_amplification = 1
- feature_elongation = 1
- feature_truncation = 1
- intergenic_variant = 0

Frequency

The alternative allele frequency (AF) in public databases (1000G, ExAC). The highest reported alternative frequency reported from the databases is used to calculate the performance value.

Definitions:

- Not reported: AF Na
- Rare: AF <= 0.005
- Intermediate: 0.005 <= AF <= 0.02
- Common: AF > 0.02

Performance value for maximum AF:

- Not reported = 3
- Rare = 2

- Intermediate = 1
- Common = -12

Inheritance Model(s)

The segregation pattern for the variant within the family. These models are currently annotated using genmod.

Definitions:

- Autsomal Recessive, denoted 'AR_hom'
- Autsomal Recessive denovo, denoted 'AR_hom_dn'
- Autsomal Dominant, 'AD'
- Autsomal Dominant denovo, 'AD_dn'
- Autosomal Compound Heterozygote, 'AR_comp'
- X-linked dominant, 'XD'
- X-linked dominant de novo, 'XD_dn'
- X-linked Recessive, 'XR'
- X-linked Recessive de novo, 'XR_dn'

Performance value for inheritance models:

- Valid model = 1
- No model = -12

Protein Functional Prediction

The predicted functional effect on the protein. Currently 2 protein effect predictors are used (Sift, PolyPhen2). Each predictors can contribute 1 point each to the overall protein predictor performance score.

SIFT predicts whether an amino acid substitution is likely to affect protein function based on sequence homology and the physico-chemical similarity between the alternate amino acids [1].

PolyPhen-2 predicts the effect of an amino acid substitution on the structure and function of a protein using sequence homology, Pfam annotations, 3D structures from PDB where available, and a number of other databases and tools (including DSSP, ncoils etc [2].

Definitions:

- Sift Terms:
  - "D" Deleterious (score<=0.05)
  - "T" Tolerated (score>0.05)
- PolyPhen2HumVar Terms:
  - "D": Probably damaging (>=0.909)
  - "P": Possibly damaging (0.447<=pp2_hvar<=0.909)
  - "B": Benign (pp2_hvar<=0.446)

Performance value for protein predictors:

- Sift:
  - D = 1
- PolyPhen2HumVar:
  - D or P = 1

Variant Quality Filter

Each variant call has a filter tranche attached to it indicating the quality of the actual variant call.

Definitions:

- PASS
- Other (Tranches e.g. For GATK [3]: "VQSRTrancheBOTH99.90to100.00"

Performance value for variant quality filter:

- PASS = 3
- Other = 0

Conservation

The level of conservation for a sequence element (PhastCons [4]), nucleotides or classes of nucleotides PhyloP [5] both from the Phast [6] package as well as genomic constraint score GERP [7] is used. The Phast datasets used in the conservation calculation were generated by the UCSC/Penn State Bioinformatics comparative genomics alignment pipeline. A description of this analysis can be found at UCSC. Each type of conservation can contribute 1 point each to the overall conservation performance score.

Definitions:

- Conserved
  - PhastCons: 0.8 >= Score <= 1
  - GERPRS: Score >= 2
  - PhyloP: Score > 2,5

Performance value for conservation:

- Conserved:
  - PhastCons = 1
  - PhyloP = 1
  - GERP = 1

Combined Annotation Dependent Depletion (CADD)

CADD is a tool for scoring the deleteriousness of single nucleotide variants as well as insertion/deletions variants in the human genome. C-scores strongly correlate with allelic diversity, pathogenicity of both coding and non-coding variants, and experimentally measured regulatory effects, and also highly rank causal variants within individual genome sequences. The CADD-score is a pre-calculated for all SNVs and for indel from 1000G-project [8].

Definitions:

- Deleterious (CADD > 20)
- Mildly deleterious (CADD > 10)

Performance value for CADD: - Deleterious = 2 - Mildly deleterious = 1

ClinVar

ClinVar [9] is a freely accessible, public archive of reports of the relationships among human variations and phenotypes, with supporting evidence.

Definitions:

- Uncertain significance = 0
- Not provided = 1
- Benign = 2
- Likely benign = 3
- Likely pathogenic = 4
- Pathogenic = 5
- Drug response = 6
- Histocompatibility = 7
- Other = 255

Performance value for ClinVar:

- Uncertain significance = 0
- Not provided = 0
- Benign = -1
- Likely benign = 0
- Likely pathogenic = 1
- Pathogenic = 2
- Drug response = 0
- Histocompatibility = 0
- Other = 0

CHAPTER 12

Dynamic Configuration File

MIP uses dynamic configuration files in YAML format to load parameters for each analysis run. An example configuration file can be found here.

To facilitate using different clusters, projects and tailoring the MIP analysis to each run without having to create new configuration files each time you can supply a cluster/project specifc configuration file. Certain paths in the configuration file information will be updated to the current analysis when MIP executes.

This requires that these two entries are added to the configuration file:

1. 'clusterConstantPath: {value}', specifying the project path.
2. 'analysisConstantPath: {value}', specifying the analysis directory.

Entries in the configuration file containing the following "dynamic strings" will be updated in MIP:

- CLUSTERCONSTANTPATH! = 'clusterConstantPath: {value}'
- ANALYSISCONSTANTPATH! = 'analysisConstantPath: {value}'
- ANALYSISTYPE! = 'analysisType: {value}'
- FDN! = '-f familyID' (from command line)
- IDN! = '-s sampleIDs' (from command line), configuration file or supplied pedigree file

For instance, the pedigree file entry in the configuration file can be supplied like this:

'pedigreeFile: CLUSTERCONSTANTPATH!/ANALYSISTYPE!/FDN!/FDN!_pedigree.txt'

and each path element will be replaced with the corresponding value as specified in the configuration file, command line (precedence) or pedigree file.

Note: Any entries not containing "dynamic strings" will not be modified by MIP.

Both updated and constant entries will be written to the analysis specific folder if specified by '-wc'.

Capture kits info supplied in the configuration file should be on sampleID level:

```
FDN:
  IDN:
    Flag: Entry
```

CHAPTER 13

Pedigree File

Family meta data file. Records important metrics for tracking samples and find biases in isolation of DNA or subsequent sequence analysis.

Among other things, the file enables:

1. Automatic coverage specification (correct target file(s))
2. Application of mendelian filtering models, e.g. *autosomal dominant*, based on pedigree, sex and disease status
3. Collection of analysis info for the sequence analysis pipeline

The pedigree file format is defined by PLINK, although we currently only support tab-sep pedigree files.

The first row should start with a "#" (hash) and contain relevant headers separated by tabs describing each column. The first six columns are mandatory. The name and order of the headers should follow:

Table 13.1: Mandatory Columns

ColumnName	Type	Summary
FamilyID	`String`	Family identification number (mandatory)
SampleID	`String`	Sample identification number (mandatory)
Father	`String`	Father identification number (mandatory)
Mother	`String`	Mother identification number (mandatory)
Sex	`String`	"1"=male, "2"=female, "other"=unknown (mandatory)
Phenotype	`String`	"-9"=missing, "0"=missing, "1"=unaffected, "2"=affected

In addition to these mandatory columns we use the pedigree file to record meta data on each individual. Entries within each column should be separated with ";" (semi-colon) and entered in consecutive order. Each individual recorded in the pedigree file is written on one line and a tab should separate each entry. No individual should be recorded twice. The order of individuals below the header line does not matter.

If there is no information on the parents or the grandparents they should be encoded as "0".

An example pedigree file can be found here.

On UPPMAX

Note: Changes to the pedigree file format will be recorded in this document.

The pedigree file should named: `<FDN>_pedigree.txt`.

Table 13.2: Additional columns in the pedigree file

ColumnName	Default Value	Type	Summary
CMMSID	Na	`String`	The clinics identification number for the individual
Tissue_origin	Na	`String`	Tissue of Isolation (DNA/RNA)
Isolation_kit	Na	`String`	Kit used to isolate nucleic acids
Isolation_date	Na	`Integer`	Date of performing isolation of nucleic acids
Isolation_personnel	Na	`String`	Personnel performing isolation of nucleic acids
Medical_doctor	Na	`String`	Responsible clinician(s)
Inheritance_model	Na	`String`	Probable disease genetic model inheritance within pedigree
Phenotype_terms	Na	`String`	Phenotypic terms associated with the disorder
CMMS_seqID	Na	`String`	Batch identification
SciLifeID	Na	`String`	ScilifeLab identification
Capture_kit	Na	`String`	Capture kit used in library preparation
Capture_date	Na	`Integer`	Date of performing capture procedure
Capture_personnel	Na	`String`	Personnel performing capture procedure
Clustering_date	Na	`Integer`	Date of clustering
Sequencing_kit	Na	`String`	Sequencing kit
Clinical_db	dbCMMS	`String`	The clinical database
Clinical_db_gene_annotation	IEM	`String`	Genes associated with a disease group within the clinical database

Pedigree capture kits aliases

- Agilent Sure Select
  - Agilent_SureSelect.V2 => Agilent_SureSelect.V2.GenomeReferenceSourceVersion_targets.bed
  - Agilent_SureSelect.V3 => Agilent_SureSelect.V3.GenomeReferenceSourceVersion_targets.bed
  - Agilent_SureSelect.V4 => Agilent_SureSelect.V4.GenomeReferenceSourceVersion_targets.bed
  - Agilent_SureSelect.V5 => Agilent_SureSelect.V5.GenomeReferenceSourceVersion_targets.bed
  - Agilent_SureSelectCRE.V1 => Agilent_SureSelectCRE.V1.GenomeReferenceSourceVersion_targets.bed
  - Latest => Agilent_SureSelect.V5.GenomeReferenceSourceVersion_targets.bed
- NimbleGen
  - Nimblegen_SeqCapEZExome.V2 => Nimblegen_SeqCapEZExome.V2.GenomeReferenceSourceVersion_targets.bed
  - Nimblegen_SeqCapEZExome.V3 => Nimblegen_SeqCapEZExome.V3.GenomeReferenceSourceVersion_targets.bed

Note: You can use other target region files with MIP but then you have to supply the complete filename with ".bed" ending.

Abbrevations

Abbreviation	Explation
FDN	Family ID
CMMSID	The CMMS sampleID
CMMS SeqID	BatchID e.g. WES8
SciLifeID	The id tag provided by Science for Life Laboratory
AR	Autosomal recessive
AD	Autosomal dominant

CHAPTER 14

Individual Identification Number (IDN)

Ensure that each individual are anonymized and unique. The IDN also facilitates tracking of all operations and analyses performed upon the individual.

Note: Changes to the IDN format will be recorded in this document.

IDN Definition

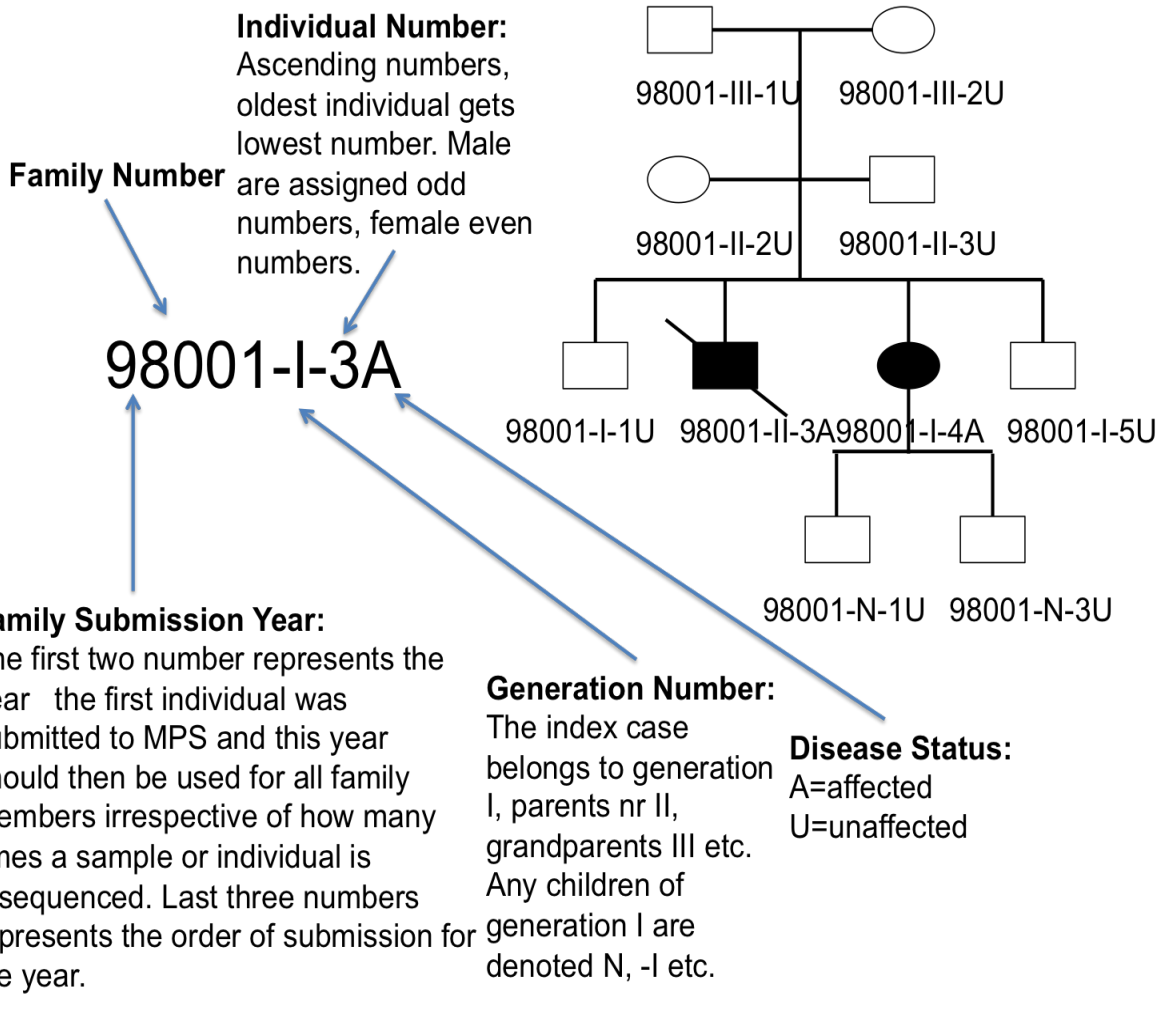
The IDN consists of a three digits connected by dots and a disease status (DS) letter after the last digit. The disease status letter can be either an "A" denoting affected subjects or a "U" denoting unaffected subjects. Each subject can only have 1 IDN and once set it should never be changed.

- The first digit represents the family identification number (FamilyID/FDN). The two first numbers ,in the first digit, represent the year that the first individual of the family was submitted to massively parallel sequencing, MPS. The attached year is determined by the first individual to be submitted to MPS and this year should then be used for all family members irrespective of how many times a sample or individual is resequenced. This rule is enforced to not create multiple IDNs for the same individual and to make sure that all family members are grouped and analyzed in the proper family. The following three numbers are the a continuous number for each family that has been submitted for the year.
- The second digit represents the generation identification number (GenerationID/GDN) within that family. The generation with the affected child is the defined as GDN = "I". Older generation are numbered in ascending order from GDN I, starting with II. Younger generations are numbered in descending order from GDN I starting with "N" (=0 in Roman numerals).
- The last digit represents the subject identification number (SubjectID/SDN) within the family and generation. Male subjects will have odd SDN numbers and female subjects even numbers. The lowest subject IDs will be given to oldest subject within each family and generation and then in ascending order (both even and odd numbers are counted). However, since there can be later additions in the pedigree this is not strictly enforced.
- The letter after the SDN is the disease status (DS) letter, which can be either of two possible letters. A = affected and U = unaffected.

Example

FamilyID.GenerationID.SubjectID(DS) or FDN.GDN.SDN(DS)

A child in the affected child generation being the second oldest male sibling in family 1 and the first to be submitted to sequencing within the family in 1998 would be written as: 98001-1-3A (Figure 1).



CHAPTER 15

The Code

Subroutines

FIDSubmitJob

Handles all communication with SLURM. All jobIDs and SLURM dependencies for all programs are set and submitted here. Each program in MIP belongs to a "path" and together with the sampleID and/or familyID creates a chain of dependencies determining the execution order in SLURM.

Paths

The central flow in MIP is called the *MAIN* path. MIP supports branching from the MAIN path for both familyIDs and sampleIDs. It is also possible to create completely separate paths, which are not associated at all with the MAIN path. However, once branched of from the MAIN path there is currently no support to merge the branch to the trunk (i.e. MAIN path) again.

Each program is supplied with a dependency flag, which determines its dependencies in the path.

Dependency Flags:

```
-1 = Not dependent on earlier scripts, and are self cul-de-sâcs
0 = Not dependent on earlier scripts
1 = Dependent on earlier scripts (within sampleID_path or familyID_path)
2 = Dependent on earlier scripts (within sampleID_path or familyID_path), but are␣
↪self cul-de-sâcs.
3 = Dependent on earlier scripts and executed in parallel within step
4 = Dependent on earlier scripts and parallel scripts and executed in parallel within␣
↪step
5 = Dependent on earlier scripts both family and sample and adds to both familyID and␣
↪sampleId jobs
```

Hash

All jobIDs are saved to the jobID hash using `$jobID{FAMILYID_PATH}{CHAINKEY}`. Only the last jobID(s) required to set the downstrem dependencies are saved.

CHAPTER 16

Indices and tables

- genindex
- modindex
- search