Contents

Chapter 1

Human Background Variation Database Tools

1.1 Introduction

The main architecture of these tools is very straight forward, as can be seen in figure 1.1. At first, it is composed of Insert/Update (*bvd-add*) and Inquiry(*bvd-get*) function. This allows user to add and use their data. Later, the requirement to have more than one production site, a place at which these tools are installed, prompted Merge (*bvd-merge*) function, which can merge content of the database from one production site with the other.

Beside the main functions, there are a few other supported features that make this database more reliable and easy to maintain, like database backup-restore (section 1.5.1), duplicated individual-variant files checking (section 1.5.2) and log file (section 1.5.3).

1.2 Design and Coding Concept

Because these tools are not aimed to be used by single person, but to be used by wide range of users, there are a couple of concepts that are embedded into the requirement of this project.

- **User friendly** - To allow all levels of users to use these tools. The only prior knowledge required is the purpose of these tools.
- **Portable** - With the requirement to have more than one production site, these tools must not be specific to one machine. Installation must not require much effort. In the best case, the tools should be able to execute after download.
- **Well-documented** - To allow the maintenance of these tools not to be restricted to one person.
- **Robust** - It should be well-written enough to be handle exceptions.

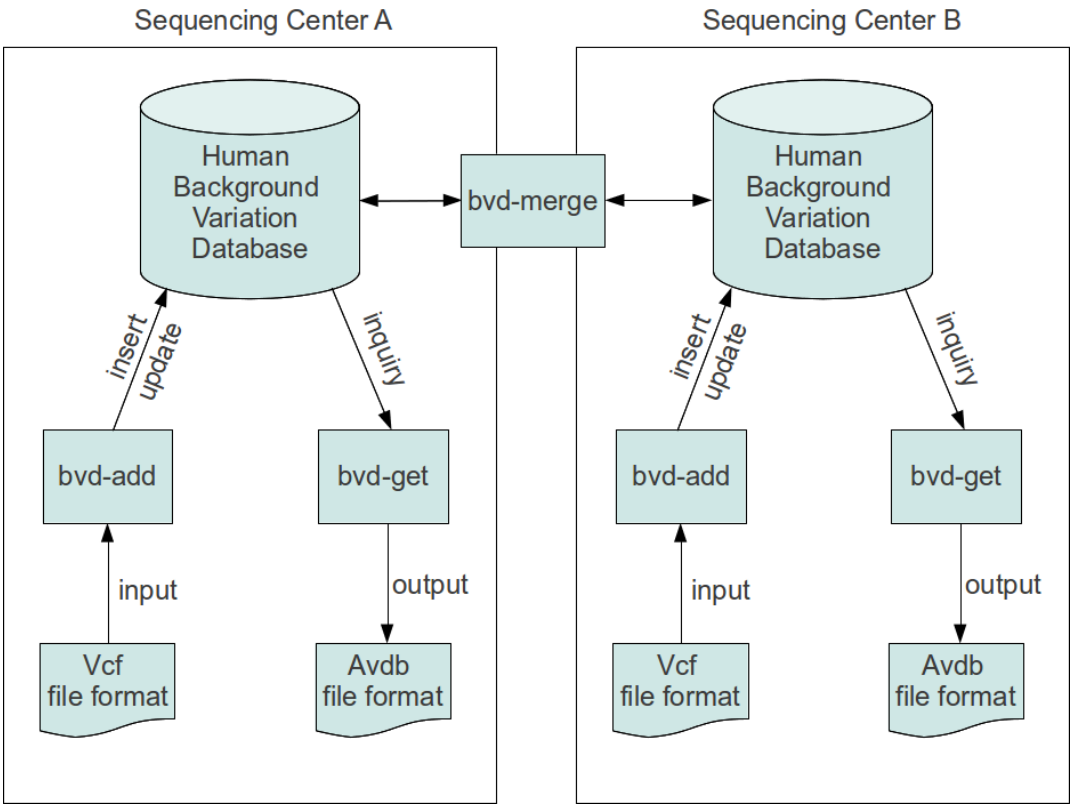

Figure 1.1. Human Background Variantion Database architecture

- **Command line interface** - These tools can be a part of any pipelines because of this feature.

- **Product quality** - The package must have some procedures to make sure that these tools work correctly, according to specification.

1.3 Programming Language and Third Party Software

With the requirement of these tools to be portable and command line interface, one of several suitable programming languages is Perl. Another reason to choose Perl is that the third party software used to parse the VCF files in this project is VCFtools. The advantage of using a Perl library in VCFtools, *vcf.pm*, is how the VCF files are handled. With the VCFtools library, the one who maintain Human Background Variation Database project doesn't have to be concerned changes in the VCF format. As long as VCFtools is up-to-date, it will handle all these changes automatically.

1.4 Main Functions

1.4.1 Bvd-add

The purpose of *bvd-add* is to transform variant information in VCF file into variant frequency and insert this into the database. One of the important features of this module is to allow users to add an *exclude tag* to each variant in a dataset. This tag has a further use in *bvd-get*. As the name imply, users can query variants frequency from the database by excluding certain tags.

Another important feature here is that the users can specify their target database. Normally, variant frequencies parsed by this module will be sent to the default database. Having this feature will allow having more than one database in the same production site.

There are a few issues regarding writing operation found here. One is that users may accidentally execute *bvd-add* for two times for one individual collection of variants. Another one is user errors, like executing *bvd-add* with the wrong tags or the wrong VCF files. How can a user roll back the database if errors happen? And which of the database should users roll back to? The answer to all these issues can be found in Supported Features (section 1.5).

1.4.2 Bvd-get

All variants frequencies in the database will be retrieved by default once *bvd-get* is executed. By having features corresponding to *bvd-add*, this module can retrieve variants frequencies that exclude some tags specified by users. Also, users can specify their source database if they don't want the frequencies from the default one.

1.4.3 Bvd-merge

bvd-merge is designed to combine the variants frequencies from another production site with the current site, the one that host this executable file. The input of this module is simply the location of the database of other production site. The issues here are the same with *bvd-add* since it's also a writing operation. The solution can be found again in Supported Features (section 1.5).

1.5 Supported Features

The features here were not implemented separately, instead, they were embedded in the main modules. The reason that these features are separately described here is that they are implemented in more than one modules. Mainly, these features are related to writing operations which make changes to the database and may cause a lot of trouble if it isn't properly handled.

1.5.1 Database Backup-Restore

Database backup and restore in this system is quite different from Database Management System (DBMS) which, normally, requires users to do the backup manually or automatically via some configuration. The concept here is very simple. Everytime, before the tools in this system perform writing operations, it will automatically perform a database backup first. The backup method is simply to make a copy of the database files with date-time suffix. The way to restore this backup file is to replace the current database files with the backup files that users want to restore.

The drawback of this feature is the usage of disk space because, by default, the backup is performed automatically in any writing operation. In this version of the tools, there is a parameter, *–savediskspace*, to disable this automatically feature in each time these tools are executed.

1.5.2 Duplicated Individual-Variant File Checking

Users may accidentally try to insert variants from the same individual more than one time. To prevent this, these tools must detect if the content of the VCF file that is to be accessed is already in the system. The very straight forward way to prevent this is to keep the content of all inserted VCF files which can be used as an unique identifier for each VCF file inserted. Unfortunately, doing this may be against individuals' privacy.

Our solution to this is to use a message digest, *SHA512sum* which hashes the whole content of file and converts it into a string with certain length. Each of the inserted VCF files will be hashed and kept together with the database. Before a new VCF file is inserted into the database, its content will be hashed and evaluated to see if the hash value already exists in the system.

1.5.3 Log File

The log file implemented in this system has a similar purpose to that of other systems. It keeps track of any crucial steps in any module. By having this feature, when users find out that something incorrect happened in the system, users can use this log file to trace back and roll back the database to the point where everything is correct.

1.6 Testing Procedure

Since the aim for these tools is to be widely used, we need to have something guarantee that every output is correct according to the specification. In the software development life cycle, testing is one of the mandatory steps that must be done to ensure the quality of the software. Two types of testing were performed in this project. One was unit test: the test to check if all the modules, *bvd-add*, *bvd-get* and *bvd-merge*, work correctly according to specification. Another one was a

performance test. This one checks if all the modules can be run in a real environment with a high workload. All the tests here were written in bash script so that they can be automatically executed using one single command. The list of test cases can be found in appendix A.1.

Appendix A

Test Cases

A.1 Unit Test

Table A.1: Test cases of *bvd-add*

Group	Case Name	Description
1	case_bvd_add_08	1. To test bvd-add with real variants with random CHROM and POS from vcf file. 2. To test if there isn't any existing bvdb database. 3. To test if database '-d' is not presented 4. To test if tags '-T' is not presented.
	case_bvd_add_01	1. To test bvd-add with real variants with random CHROM and POS from vcf file. 2. To test if there isn't any existing bvdb database. 3. To test if database '-d' is not presented. 4. To test if tags '-T' is presented.
2	case_bvd_add_02	Using data from case_bvd_add_08 1. To test bvd-add with real variants with random CHROM and POS from vcf file. 2. To test if it has existing bvdb database. 3. To test if database '-d' is not presented. 4. To test if no tags in current database. 5. To test if tags '-T' is not presented.
	case_bvd_add_09	Using data from case_bvd_add_08 1. To test bvd-add with real variants with random CHROM and POS from vcf file. 2. To test if it has existing bvdb database. 3. To test if database '-d' is not presented. 4. To test if no tags in current database. 5. To test if tags '–tag' is presented.

Continued on next page

Table A.1 – continued from previous page

Group	Case Name	Description
	case_bvd_add_10	Using data from case_bvd_add_01 1. To test bvd-add with real variants with random CHROM and POS from vcf file. 2. To test if it has existing bvdb database. 3. To test if database '-d' is not presented. 4. To test if there are tags in current database. 5. To test if tags '-T' is not presented.
	case_bvd_add_11	Using data from case_bvd_add_01 1. To test bvd-add with real variants with random CHROM and POS from vcf file. 2. To test if it has existing bvdb database. 3. To test if database '-d' is not presented. 4. To test if there are tags in current database. 5. To test if tags '-T' is presented.
3	case_bvd_add_03	To test the validity of frequency calculation as well as the order of the data for all possible case that can happen from individual variant files. There are 2 groups of data here 1. Real data from 200danes database 2. Synthetic data which is modified from some varaints in 200danes database
4	case_bvd_add_04	To test with all possible combination of tags for each variant. There are 8 vcf files for all expected combination.
5	case_bvd_add_05	To test with multi column vcf file. The input variants are the same with those from case_bvd_add_03 but, instead of in separated vcf files, variants are combined into one single multicolumn vcf file.
6	case_bvd_add_06	1. To test bvd-add with real variants with random CHROM and POS from vcf file. 2. To test if there isn't any existing bvdb database. 3. To test if database '-d' is presented.
7	case_bvd_add_07	1. To test bvd-add with real variants with random CHROM and POS from vcf file. 2. To test if it has existing bvdb database. 3. To test if parameter '–database' is presented.
8	case_bvd_add_12	To handle some strange characters in GT fields, '.'.
9	case_bvd_add_13	Test if the database system, both database itself and message digest, are properly backup.
10	case_bvd_add_14	Test if the parameter '-s' is presented.
	case_bvd_add_15	Test if the parameter '–savediskspace' is presented.

Table A.2: Test cases of *bvd-get*

Group	Case Name	Description
1	case_bvd_get_02	To test with all possible combinations of normal excluded tags by using tags and variants generate from case_bvd_add_04 There are 3 tags in case_bvd_add_04 - colon_cancer - lung_cancer - prostate_cancer So it has 16 possible normal combinations 1. <no excluded tags> 2. colon_cancer 3. colon_cancer,lung_cancer 4. colon_cancer,lung_cancer,prostate_cancer 5. colon_cancer,prostate_cancer 6. colon_cancer,prostate_cancer,lung_cancer 7. lung_cancer 8. lung_cancer,colon_cancer 9. lung_cancer,colon_cancer,prostate_cancer 10. lung_cancer,prostate_cancer 11. lung_cancer,prostate_cancer,colon_cancer 12. prostate_cancer 13. prostate_cancer,colon_cancer 14. prostate_cancer,colon_cancer,lung_cancer 15. prostate_cancer,lung_cancer 16. prostate_cancer,lung_cancer,colon_cancer This testcase consist of 16 sub testcases. Each represent one combination above. Side note : Those 16 combinations can be reduced to 8 actual combinations a. <no excluded tags> b. colon_cancer c. colon_cancer,lung_cancer d. colon_cancer,lung_cancer,prostate_cancer e. colon_cancer,prostate_cancer f. lung_cancer g. lung_cancer,prostate_cancer h. prostate_cancer.

Continued on next page

Table A.2 – continued from previous page

Group	Case Name	Description
2	case_bvd_get_01	To test with some variants in different chromosome from real data and also with some expected tags.
	case_bvd_get_05	Same as case_bvd_get_01 but with parameter '-d' presented
	case_bvd_get_06	Same as case_bvd_get_01 but with parameter '–database' presented
3	case_bvd_get_03	This is the extension from case_bvd_get_02. The purpose is to test with a weird excluded-tags combination. Only one case so far: lung_cancer, colon_cancer, lung_cancer Data here are from case_bvd_add_04
4	case_bvd_get_04	This is the extension from case_bvd_get_02. The purpose is to test with two tags, one tag that exist in database and another not exist. Only one case so far: lung_cancer, breast_cancer Data here are from case_bvd_add_04

Table A.3: Test cases of *bvd-merge*

Group	Case Name	Description
1	case_bvd_merge_01	To test if one of the two given databases does not exist
2	case_bvd_merge_02	To test if the one of the input databases doesn't have the database file
3	case_bvd_merge_03	To test if the one of the input databases doesn't have the message-digest file
4	case_bvd_merge_04	To test function 'validate_bvdb' from command line with the database and without database file
5	case_bvd_merge_05	To test if 1. Local database exist with default location 2. Some of the content from the input databases was already in the current local database
	case_bvd_merge_06	To test if 1. Local database exist with default location 2. Some of the content from the input databases was duplicated with each other
	case_bvd_merge_07	To test if 1. Local database does not exist 2. Some of the content from the input databases was duplicated with each other

Continued on next page

Table A.3 – continued from previous page

Group	Case Name	Description
	case_bvd_merge_09	To test if 1. Local database exist with location from '-d' parameter 2. Some of the content from the input databases was already in the current local database
6	case_bvd_merge_08	To test with some variants in different chromosome from real data in. And also with some expected tags. With following conditions 1. 2 vcf files in default local database. 2. 2 vcf files in the first input database. 3. 1 vcf file in the second input database.
	case_bvd_merge_10	To test with some variants in different chromosome from real data in. And also with some expected tags. With following conditions 1. 2 vcf files in local database specified by parameter '-d'. 2. 2 vcf files in the first input database. 3. 1 vcf file in the second input database.
	case_bvd_merge_11	To test with some variants in different chromosome from real data in. And also with some expected tags. With following conditions 1. No default local database. 2. 3 vcf files in the first input database. 3. 2 vcf file in the second input database. 4. Output to default local database.
	case_bvd_merge_12	To test with some variants in different chromosome from real data in. And also with some expected tags. With following conditions 1. No default local database. 2. 3 vcf files in the first input database. 3. 2 vcf file in the second input database. 4. Output to local database specified by parameter '-d'.
7	case_bvd_merge_13	To compare the result of bvd-add and bvd-merge with 5 real vcf files with expect tags. In bvd-merge, it'll merge the data from 5 databases, including the local one.
8	case_bvd_merge_14	Test if the parameter '-s' is presented
	case_bvd_merge_15	Test if the parameter '–savediskspace' is presented
9	case_bvd_merge_16	To test if the database is properly backup

A.2 Performance Test

Table A.4: Test cases of performance test

Group	Case Name	Description
bvd-add	case_performance_01	To see how well bvd-add be able to add variants from 200 real vcf files.
	case_performance_02	To see if Bvd-add can handle large DB, 256,000,000 variants
	case_performance_03	To see if Bvd-add can handel DB with expected number of tags,100,000 variants with 1,000 tags
bvd-get	case_performance_04	To see if Bvd-get can handel DB with expected number of tags, 100,000 variants with 1,000 tags, number of excluded tags = 100
	case_performance_05	same as case_performance_04 but more extreme, number of excluded tags = 999 and not sorted