Importing Next Generation Sequencing Data in LOVD 3.0

J. Hoogenboom

April 17th, 2012

Abstract

Genetic variations that are discovered in mutation screenings can be stored in Locus Specific Databases (LSDBs). *Leiden Open Variation Database* (LOVD)[1] version 3.0, an open-source LSDB system currently in development, will extend its capabilities beyond that of a traditional LSDB by storing genetic variation data anywhere on the genome.

With the advent of Next Generation Sequencing (NGS) techniques, researchers are able to generate large amounts of genetic variation data. Because of the enormous quantity of the data, researchers are in need of a way to import this data into LSDBs automatically.

This report describes the development of NGS data import functionalities in LOVD 3.0. Data can be imported from VCF files[2] or SeattleSeq Annotation files[3].

Variants are described according to HGVS nomenclature guidelines[4] with the use of web services provided by the Mutalyzer project[5].

Finally, additional data about specific variants can be found in other public LOVD installations worldwide though the use of a new Global Search feature.

1 Introduction

Locus Specific Databases (LSDBs) store genetic variation data within the focus of a specific gene or disease. *Leiden Open Variation Database* (LOVD)[1] is an open-source LSDB system that can be downloaded for free from LOVD.nl. LOVD 3.0, which is currently in the Beta stage of development, will extend its capabilities beyond that of a traditional LSDB by being able to store variant data anywhere on the genome, also away of known genes.

LOVD is written in PHP, a programming language developed specifically for creating dynamic websites and web-based applications. The LOVD user interface consists entirely of HTML web pages that are dynamically generated by the PHP back-end. In LOVD 3.0, with the use of JavaScript and AJAX (Asynchronous JavaScript And XML), the web pages are highly interactive and often do not need to be reloaded to do such things as searching or sorting data in a table. Instead, AJAX allows scripts on the page to communicate with the web server and update parts of the page accordingly.

Traditionally, patients are screened for disease-causing mutations using screening techniques that are targeted at a single, potentially defective gene. This is done by selectively amplificating (a fraction of) this gene from the genome to determine the nucleotide sequence of the gene using various traditional techniques like Sanger Sequencing and Southern Blotting. Various screenings are done to determine how a patient's copy of the gene differs from the normally functioning gene product. In case several genes may be involved in a single disease, experiments may need to be repeated for each gene.

With the current rapid developments in the field of Next Generation Sequencing (NGS), it has become possible to get a detailed overview of the complete genome (or exome) of an individual. This way, it has become relatively easy to discover tens of thousands of an individual's genetic variants, across the entire genome. All genes can be screened simultaneously, eliminating the need to select genes of interest and enabling the possibility of finding novel gene-to-disease associations.

LSDBs can aid researchers by storing the data about a patient's variants in a structured way. Additionally, online data sources can be linked to the variants in the LSDB to detect potentially problematic variants (e.g. variants that could cause splicing defects resulting in defective proteins). Because entering all variants from NGS data manually would be a tremendous amount of work, scientists are in need of a way to automatically insert this information into LSDBs. For this reason, LOVD 3.0 will include the option to import genome- or exome-scale variation data from Variant Call Format (VCF) and SeattleSeq Annotation files.

The Variant Call Format[2] is the most popular file format in use to describe an individual's genetic variations. Because VCF files do not contain data about the effects of the variants on transcripts, LOVD 3.0 provides functionality to map the variants onto transcripts using a number of web services provided by the Mutalyzer project[5].

SeattleSeq Annotation[3] is a service provided by the department of Genome Sciences at the University of Washington. Scientists may upload a file with genetic variant information, like VCF files, and the SeattleSeq service will enrich the data with detailed annotations of the variants. These annotations include gene names, transcript accession numbers, predicted protein changes, and functional effects of the variants.

Finally, the variants that were detected in an (NGS) experiment may have been studied before. For example, chances are another LOVD contains detailed information about a variant's pathogenicity. Therefore, a new service was implemented which allows the user to find out whether a given variant has been reported in other LOVD installations. The service will provide links to other databases that share the same variant so the user can easily find any information stored in any LOVD worldwide.

This service makes use of data exposed through the LOVD API (Application Programming Interface). With the LOVD API, the data in LOVD can be accessed in formats that are easily interpreted by computer programs, as opposed to the usual HTML web pages. The API also allows genome browsers to include LOVD data as a special track. LOVD 3.0 does not yet have an API, so LOVD 3.0 installations currently do not show up in the 'Global Search' service.

This report describes the development of the VCF and SeattleSeq import scripts, the automatic mapping functionality, and the Global Search service.

1.1 HGVS nomenclature guidelines

The Human Genome Variation Society (HGVS) provides guidelines[4] for the description of genetic variants. These guidelines were created to replace the various ways researchers used to describe the variants they found. The format was specially crafted to provide unambiguous and understandable descriptions of variants.

To further prevent ambiguity of the descriptions, variants must be described with respect to a specific reference sequence (which is either a chromosome, cDNA, RNA, or protein sequence).

An example HGVS variant description is `NM_004006.1:c.3G>T`. In this example, a transcript reference sequence is specified with `NM_004006.1`. Instead of using an accession number, chromosomes *may* also be specified by `chrX` and genes by their gene symbol for easier recognition. Furthermore, if the reference sequence can be clearly derived from the context, it may be omitted.

The `c.` part specifies that the nucleotides in the sequence are numbered according to coding DNA numbering. The numbering schemes are listed in Table 1. In this case, `c.3` refers to the third nucleotide in the coding DNA sequence, which is usually the 'G' in the 'ATG' start codon. Ranges of nucleotides are described using an underscore (_) between the start and end positions. Both the start and end positions are inclusive. For insertions, the positions point to the two bases (i.e. one base on each side) flanking the insertion.

Prefix	Description
`g.`	Genomic positions. The first nucleotide in the chromosome sequence is number 1.
`m.`	Genomic positions on the mitochondrial DNA sequence.
`c.`	Coding DNA positions. Nucleotide 1 is the 'A' in the 'ATG' start codon; the base before that is -1. Intronic bases are numbered using offsets from the nearest exon. So, the first base in an intron is counted as `c.234+1`, where the previous exon ended at base 234. The first base after the stop codon is numbered as `c.*1`.
`r.`	RNA numbering. Nucleotide 1 is at the transcription initiation site.
`n.`	RNA numbering on a non-coding sequence.
`p.`	Protein numbering. The initiating Methionine is number 1.

Table 1: Numbering schemes in HGVS variant notations.

Finally, `G>T` describes the actual variant. In this case, the nucleotide at position 3 is changed from a 'G' to a 'T' (a substitution). The most common variant types are summarised in Table 2.

1.1.1 Mutalyzer

The Mutalyzer project[5] provides services for converting variant descriptions between different reference sequences and numbering schemes. Calculating these conversions can be rather difficult, for example because they depend on exon boundaries. Mutalyzer was created to aid

Type	Example	HGVS description
Substitution	ACTG ⇒ ACCG	`g.3T>C`
Deletion	ACTG ⇒ ACG	`g.3del`
Insertion	ACTG ⇒ ACGTG	`g.2_3insG`
Duplication	ACTG ⇒ ACTCTG	`g.2_3dup`
Inversion	ACTG ⇒ AAGG	`g.2_3inv`
Deletion/insertion	ACTG ⇒ AGGC	`g.2_4delinsGGC`

Table 2: The most common HGVS variant types.

researchers in making and understanding HGVS variant descriptions. It is also used in LOVD to tell the user whether their variant description is valid HGVS syntax.

Besides variant description conversion, Mutalyzer also provides web services that report detailed information about genes and their transcripts.

1.2 LOVD database tables

Figure 1 shows the relevant tables in the LOVD database model.

In an LOVD database, genes and transcripts are stored in the 'genes' and 'transcripts' tables, respectively. One gene can have several transcripts linked to it, but logically, each transcript belongs to a single gene.

In order to support variants that are not linked to a certain gene, all variants in LOVD are stored in the 'variants' table by their chromosomal positions. Other information stored in the 'variants' table includes the type of variant (whether it is a substitution, deletion, insertion, duplication, etc.) and its predicted functional effect. The actual description of the variant is formatted and stored according to the guidelines provided by the HGVS.

If a variant hits a certain transcript, a separate entry in the 'variants_on_transcripts' table links the genomic variant entry to a specific transcript. The 'variants_on_transcripts' table stores additional information about this variant on the transcript, like its position on the transcript and the HGVS description on the transcript level. Also the functional effect and the possible change in a protein's amino acid sequence that this variant may cause is stored here.

Individuals are stored in the 'individuals' table. When an individual has been screened for variants, a screening entry is added in the 'screenings' table, which describes the type of screening (like NGS) that was performed.

Because several variants may be found with a single screening, and the same variant may be confirmed in different screenings, the connection between 'variants' and 'screenings' is a so-called many-to-many relationship. Since in a proper data model a column can only contain a single value, and the other values should not be duplicated over multiple rows, many-to-many relations are done through a separate two-column table. Thus, in the case of LOVD, connections between 'variants' and 'screenings' are stored in the 'screenings_to_variants' table.

Diseases are stored in the 'diseases' table. If an individual is known to have a certain disease, this link is made through a separate 'individuals_to_diseases' table. Likewise, genes that are associated with a given disease are linked through the 'genes_to_diseases' table. Phenotypical data related to diseases, like blood pressure, are stored in the 'phenotypes' table and reference the individual and the diseases they are related to.

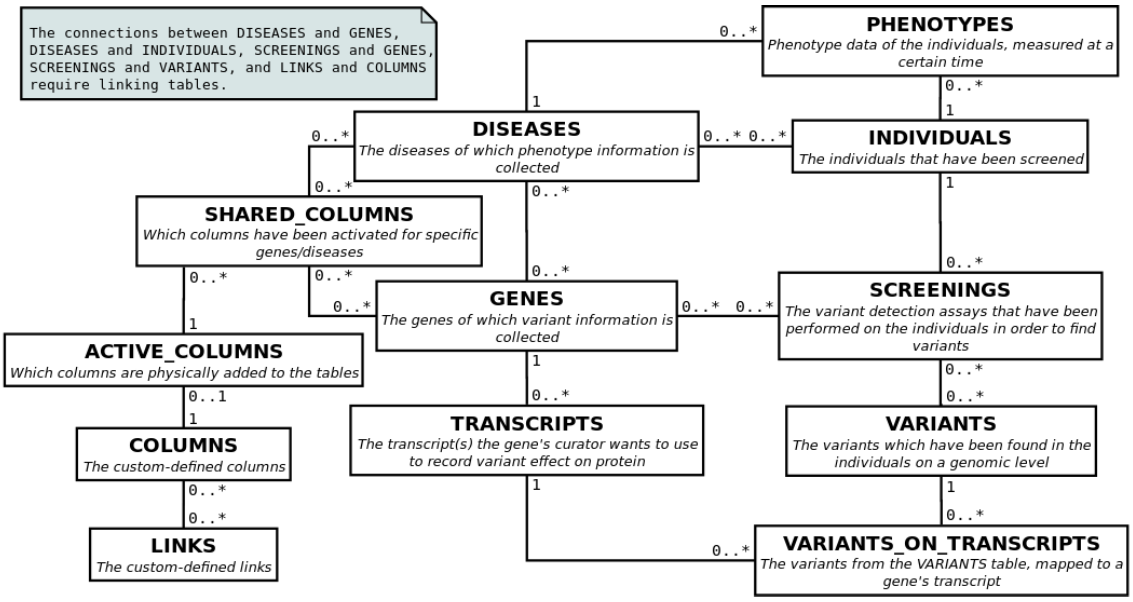

The connections between DISEASES and GENES, DISEASES and INDIVIDUALS, SCREENINGS and GENES, SCREENINGS and VARIANTS, and LINKS and COLUMNS require linking tables.
PHENOTYPES
Phenotype data of the individuals, measured at a certain time
DISEASES
The diseases of which phenotype information is collected
INDIVIDUALS
The individuals that have been screened
SHARED_COLUMNS
Which columns have been activated for specific genes/diseases
GENES
The genes of which variant information is collected
SCREENINGS
The variant detection assays that have been performed on the individuals in order to find variants
ACTIVE_COLUMNS
Which columns are physically added to the tables
COLUMNS
The custom-defined columns
TRANSCRIPTS
The transcript(s) the gene's curator wants to use to record variant effect on protein
VARIANTS
The variants which have been found in the individuals on a genomic level
LINKS
The custom-defined links
VARIANTS_ON_TRANSCRIPTS
The variants from the VARIANTS table, mapped to a gene's transcript
0..*
1
0..1

Figure 1: An overview of the relevant tables in the LOVD 3.0 data model.

1.2.1 Columns and links

Besides from the more 'biological' tables, LOVD also has a number of database tables to support flexible storage of the data itself.

LOVD allows database managers to add additional data columns or edit some of the default columns in a feature called 'Custom columns'. Custom columns can be added to the 'variants' (also called 'VariantsOnGenome' in the context of custom columns), 'variants_on_transcripts', 'individuals', 'screenings', and 'phenotypes' tables. The column settings (name, description on submission forms, allowed values, etc.) are stored in the 'columns' table. When a column is physically added to the database table, another entry is included in the 'active_columns' table. This allows LOVD to display tables that include custom columns without having to inspect the structure of all relevant database tables in advance.

Furthermore, LOVD 3.0 allows 'variants_on_transcripts' custom columns to be activated on a per-gene basis. Similarly, 'phenotypes' custom columns can be activated on a per-disease basis. When having several diseases in LOVD, this saves submitters from having to fill out long forms with information that is not related to their specific disease or gene of interest. The 'shared_columns' table contains the links between these columns and the gene or disease entries they are active for, along with settings specific to that gene or disease.

Another feature, called 'Custom links', allows data to be linked to external resources, like dbSNP.[6] Custom links specify a 'place holder' that is replaced with the actual link when the data is viewed. For example, the dbSNP custom link that comes with LOVD by default, replaces the place holder `{dbSNP:rs12345}` with a link to dbSNP entry rs12345. Such custom links are stored in the 'links' table. They can be activated for each custom column separately. The 'columns_to_links' table keeps track of which custom links are active for which custom columns.

The custom links feature has some advantages over storing raw URLs. The most important advantage is that the custom links feature provides improved security. Without custom links, LOVD would need to allow the use of HTML to enable users to include links to other databases. This is a security risk, because users can include links or scripts to malicious websites, allowing for web forgery.

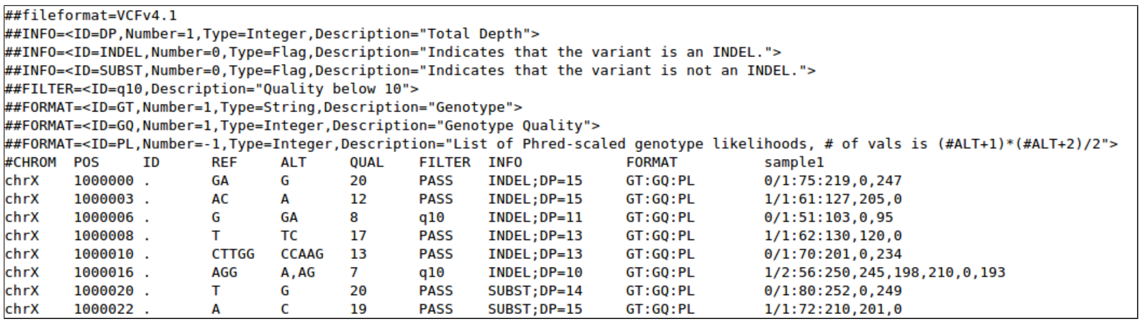
Another advantage is that all links can be changed easily in case an external resource moves to a different location. Furthermore, users do not need to know the exact URL format for a certain resource. The place holder can be inserted with a single mouse click, leaving only the ID to be entered by the user. And finally, instead of showing complete URLs in data tables, only the database name they refer to is displayed, which saves a lot of space and also results in a much cleaner look.

2 Materials and Methods

2.1 Variant Call Format

The Variant Call Format[2] was the first file format for which importing data into an LOVD 3.0 database was implemented. In this tab-delimited file format, each line in the file corresponds to a single location in the genome. VCF files do not provide annotation of the variants on transcripts.

A small, fictional VCF file may be found in Figure 2.



```
##fileformat=VCFv4.1
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=INDEL,Number=0,Type=Flag,Description="Indicates that the variant is an INDEL.">
##INFO=<ID=SUBST,Number=0,Type=Flag,Description="Indicates that the variant is not an INDEL.">
##FILTER=<ID=q10,Description="Quality below 10">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=PL,Number=-1,Type=Integer,Description="List of Phred-scaled genotype likelihoods, # of vals is (#ALT+1)*(#ALT+2)/2">
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO            FORMAT          sample1
chrX    1000000 .       GA      G       20      PASS    INDEL;DP=15     GT:GQ:PL        0/1:75:219,0,247
chrX    1000003 .       AC      A       12      PASS    INDEL;DP=15     GT:GQ:PL        1/1:61:127,205,0
chrX    1000006 .       G       GA      8       q10     INDEL;DP=11     GT:GQ:PL        0/1:51:103,0,95
chrX    1000008 .       T       TC      17      PASS    INDEL;DP=13     GT:GQ:PL        1/1:62:130,120,0
chrX    1000010 .       CTTGG   CCAAG   13      PASS    INDEL;DP=13     GT:GQ:PL        0/1:70:201,0,234
chrX    1000016 .       AGG     A,AG    7       q10     INDEL;DP=10     GT:GQ:PL        1/2:56:250,245,198,210,0,193
chrX    1000020 .       T       G       20      PASS    SUBST;DP=14     GT:GQ:PL        0/1:80:252,0,249
chrX    1000022 .       A       C       19      PASS    SUBST;DP=15     GT:GQ:PL        1/1:72:210,201,0
```

Figure 2: A simple VCF file that shows the basics of the format. The data shown here is fabricated.

2.1.1 VCF file structure

VCF files begin with a number of header lines that include metadata such as the file format version and the name of the program that generated the file. These lines are easily distinguished since they begin with two hash characters (`##`). The header lines also contain descriptions of any filters that have been applied and the various bits of data that are provided for each position in the file. These descriptions specify an ID of the filter or data field (usually a two-letter abbreviation), the number of values a data field can take, the type of the values (`integer`, `float`, `string` (text) or `flag` (a data field with no value, the presence of which indicates that a certain condition is met)) and a short textual description of the data field or filter.

Variant data

The last header line begins with a single hash character and contains the column headers. VCF files contain 8 fixed columns and one optional column which is only present when the VCF file includes sample data; genotype-related information for a specific individual, tissue, or cell line that was sequenced. These are summarised in Table 3. If the VCF file contains sample data, the `FORMAT` column is present and it is followed by one extra column for each sample containing the actual data.

Column	Description	Content
`CHROM`	The chromosome number.	`chr15`, `chrX`, etc.
`POS`	The nucleotide position.	Positive integers
`ID`	A dbSNP ID.	`rs12345`
`REF`	Bases in the reference sequence.	`A`, `ATG`, etc.
`ALT`	Alternate alleles found for this site, separated by commas.	`C`, `AT,AG`, etc.
`QUAL`	Quality of the data for this site.	Positive decimal numbers
`FILTER`	Whether the data passes the applied filters.	`PASS`, or the IDs of the failed filters
`INFO`	Detailed data about this site, separated by semicolons.	Pairs of IDs and values like `DP=14;AA=T`
`FORMAT`	Which sample data is included for this site (if applicable), separated by colons.	IDs of the included fields, like `GT:GQ:DP`
(Samples)	The data specific for this sample. A separate column for each sample.	Values, like `0/1:48:8`

Table 3: Columns in the Variant Call Format.

The variant data begins after the line containing the column headers. Each line in the file corresponds to the chromosome and position given on that line. The VCF specification prescribes that the sites are ordered by position and grouped by chromosome. If certain data, for example a dbSNP ID, is missing or unknown, a '`.`' is used in place.

For all variants that are not substitutions of a single base by another, the VCF file includes extra bases around the actual variant. For example, the base before a deletion is always included in the file, so for a deletion of the 'T' in 'AGTC', the `REF` column contains 'GT' and the `ALT` column contains 'G'. Similarly, VCF files may also contain a few bases after the variant under some circumstances.

If the file contains sample data, the VCF specification prescribes that a `GT` field is included as the first field for each sample, containing the genotype for that sample. A value like `0/1` means that the sample contains a `REF` allele (0) and the first `ALT` allele (1). Similarly, `1/2` denotes a case of compound heterozygosity: both the first and second `ALT` alleles are present. For phased data, where it is known that certain variants are present together on the same chromosome, the slash is replaced by a pipe character (i.e. `1|1`).

Genotype data may also be specified using 'Phred-scaled likelihoods' in the optional `PL` field. This is a comma-separated list of integers between 0 and 255, where 0 is the most probable. Each likelihood score refers to a specific possible genotype, in the order `0/0`, `0/1`, `1/1`, `0/2`, `1/2`, etc. Thus, a `PL` value of `255,220,234,190,0,201` specifies a genotype of `1/2`.

2.1.2 Reading VCF files

The VCF file importer that was written for LOVD first skips the header lines, because they contain no data relevant to LOVD, and then reads the variants from the file one by one. The file is not read into memory all at once, because PHP scripts may only consume a limited amount of memory to prevent issues on shared web servers that run several distinct websites.

If a line in the VCF file contains more than one alternate allele, a separate entry for each allele is inserted into the database. Variants that can't be stored, which may be the case if the bases contain uncertainties or the chromosome number is invalid, are counted and the first 100 are printed at the end of the import so the user can see what may be going wrong.

If sample data is present, variants that are not included in the genotype of any of those samples are not imported.

Making HGVS descriptions

In LOVD, the variants need to be described according to the HGVS nomenclature guidelines. The VCF import script can recognise all variant types listed in Table 2.

The variant types are detected as follows. First, the variant's startPosition is defined as the position in the `POS` column of the VCF file, and the endPosition is the startPosition minus one plus the number of given `REF` bases. Because VCF files may contain flanking bases that are not part of the variant, identical bases at the beginning of the reference and alternate allele sequences are removed and the number of removed bases is added to startPosition. Identical bases are removed from the ends of the sequences in a similar fashion, lowering endPosition by the number of removed bases. These simple steps isolate the difference between the `REF` and `ALT` sequences. Then, the remaining sequences are compared.

- If `ALT` is empty, the variant is a deletion of `REF` and the HGVS notation is `g.`$startPosition\_endPosition$`del`.
- If `REF` is empty, and the original `ALT` (before truncating) contained twice the truncated `ALT`, it's a duplication. In this case, the startPosition must be lowered by the number of

duplicated bases. The HGVS notation becomes `g.`*startPosition*_*endPosition*`dup`.

- If `REF` is empty, and it was not a duplication, it's an insertion. In this case, due to how the truncation steps changed the positions, startPosition and endPosition must be exchanged first. Then, the HGVS notation becomes `g.`*startPosition*_*endPosition*`ins`*ALT*.
- If `REF` and `ALT` are both one base, it's a substitution and the HGVS notation is `g.`*startPositionREF*>*ALT*.
- If `ALT` is the reverse complement of `REF`, it's an inversion and the HGVS notation is `g.`*startPosition*_*endPosition*`inv`
- Finally, any other case is considered to be a deletion followed by an insertion and the HGVS notation becomes `g.`*startPosition*_*endPosition*`delins`*ALT*.

For variants on mitochondrial DNA, `g.` is replaced by `m.` to reflect this. Also, for deletions and duplications of a single base, endPosition is omitted.

Once the HGVS description has been constructed, the variant is ready to be inserted into the database. If the user submitted the VCF file as part of a mutation screening, the variants are linked to the screening in the 'screenings_to_variants' table.

2.2 Mapping the variants to transcripts

When a user enters a single variant entry, they get the option to enter information about that variant on specific transcripts too. However, when someone imports tens of thousands of variants from a VCF file, this process must be automated instead. Therefore, a second major new feature in LOVD 3.0 is automatic variant mapping.

Because the process of mapping variants to transcripts relies on a number of web services, it generates quite some slow queries over the internet. This makes automatic mapping a slow process that can't be executed 'on-the-fly' while importing the variants. Instead, all variants are loaded into the database including only the genomic data that can be found in the VCF file. The variants are then processed in small groups to map them to known genes and transcripts using services on the internet without requiring any user intervention. The script runs in the background while users can continue browsing the database.

2.2.1 Keeping the script running

If there are any variants in the database that await automatic mapping, the web browser sends a request to the web server, which will run the mapping script. The mechanism by which the browser requests additional data from the server without reloading the web page is an example of LOVD 3.0's use of AJAX (Asynchronous JavaScript And XML). This is implemented by a tiny script embedded in the footer that is included on every page in LOVD.

After mapping a number of variants to transcripts, the server reports back with some feedback on the progress. The reported data is visualised by means of a small circular progress meter in the footer of the page. Depending on whether there are any more variants to map, the browser automatically sends a new AJAX request for the script to the server. This way, the automatic mapping process is kept active as long as anyone has a browser window with LOVD opened.

When all variants are mapped, the time is saved in the user's session data and the browser will not automatically request the mapping script for 24 hours.

2.2.2 Overview of the mapping script

The mapping script performs a number of steps each time it is called.

1. Select a group of variants to map during this call.
2. Get a list of genes and transcripts that are present within the range of the variants and find out which of those genes and transcripts each variant hits.
3. If the user requested so, attempt to create gene and transcript entries for genes that are not yet in LOVD.
4. Retrieve mapping information about the variant on the transcripts in the database and insert this information into the database.

1. Selecting a group of variants
When development of the mapping script began, there were no means to store mapping progress information in the database. Any variants that did not have 'variants_on_transcripts' entries in the database would be considered 'unmapped' and would be picked up by the script. The script kept a counter of the number of variants that it could not map on each chromosome (for example because the variant did not hit any transcript) in the user's session data.

The mapping script first selected a random chromosome with unmapped variants (variants without 'variants_on_transcripts' entries) on it at the beginning of each call. It would then select the first 5 unmapped variants on that chromosome, ordered by position, skipping the number of variants that it could not map previously. This was later changed so it selects one variant plus at most 24 others within a range of 100,000 bases downstream of that. In case these values need to be changed this can be done easily, because these values are defined at the beginning of the script. This prevents the script from requesting irrelevant transcript data for a large portion of the chromosome between two distant, yet consecutive, variants. The limit of 25 variants per call is implemented to make sure the script will not attempt to map many variants in one go, which may make the call very slow.

However, this approach had three important drawbacks. First, the counters in the user's session data would reset when a user logs on or off, which caused the script to re-attempt all failed variants. Second, if the user would add a variant at the beginning of the chromosome, the script may skip over it thinking that it failed mapping previously. And finally, two instances of the script would process the same variants if they select the same chromosome (either by chance or because there is just one chromosome with unmapped variants left).

This issue was later resolved by adding a new 'mapping_flags' data column to the 'variants' database table. This single-byte column allowed for a number of refinements to the mapping script. An overview of the different flags that are stored in this column, can be found in Table 4.

First of all, instead of automatically mapping *all* variants in the database, users can now select whether a variant *should* be mapped. Additionally, the user may select whether the

Flag name	Description
`ALLOW`	Flags whether the user requested automatic mapping for this variant.
`ALLOW_CREATE_GENES`	Flags whether the mapping script may create a gene entry if this variant can be mapped to genes that are not in the database.
`IN_PROGRESS`	Flags that the mapping script is currently working on this variant.
`NOT_RECOGNIZED`	Is set when Mutalyzer could not parse the variant description, usually because it is not valid HGVS syntax.
`ERROR`	Is set when the mapping script failed to connect to Mutalyzer while working on this variant.
`DONE`	Is set when the mapping script has successfully processed the variant, regardless of whether the variant actually got mapped to anything.

Table 4: Mapping flags.

mapping script should attempt to create a gene entry if it can't map the variant otherwise. The option to turn on automatic mapping is currently only present on the VCF file upload form, but the LOVD development team may decide to add automatic mapping controls elsewhere in LOVD in the future.

Next, concurrency is improved because the mapping script can now flag the variants it is working on. Other concurrent calls to the script will ignore those variants which prevents two instances of the script from doing the same work.

Because the flags can be used to isolate which variants need to be mapped, the variant selection step at the beginning of the process has been improved in two ways. The script can now map variants that are already mapped to other transcripts because it no longer identifies unmapped variants by the absence of a 'variants_on_transcripts' entry. And because there is no need to count unmappable variants, the variants are no longer processed in order of position which improves efficiency when running multiple instances of the script concurrently. Instead, the script now selects at most 25 variants within a range of 100,000 bases starting with a *randomly chosen* unmapped variant. This also eliminated the need for selecting a random chromosome first.

And finally, the mapping script is automatically suspended for one hour in case the connection to Mutalyzer fails. An error log entry is created when this happens, too. Users can still click the progress indication icon to resume the mapping process immediately.

The script was later extended to be able to map one specific variant on request. This allows users to retry mapping the variant in case it failed, or to map the variant to newly added transcripts in one click.

2. Getting the relevant transcript data

Once the script has selected a number of variants, it will request a list of all transcripts in the range of the variants using the web services provided by Mutalyzer. For each transcript, the genomic start and end positions, the gene symbol and the NCBI accession number (usually starting with `NM_`) including its version number are requested. Because Mutalyzer did not have a web service that includes this information in one call, the script originally needed to do 3 extra calls for each transcript to get the relevant information.

1. Retrieve a list of relevant transcripts by asking Mutalyzer for the `c.` HGVS descriptions for a deletion of the range of interest (e.g. `chrX:g.123000_125000del`).
2. For each transcript, ask Mutalyzer about positions of the transcript's features (a list of exons, but also the begin and end positions of the transcript itself), which are given relative to the start codon in the transcript.
3. Ask Mutalyzer for the `g.` HGVS description for a deletion of the entire transcript (e.g. `NM_001234.5:c.-35_*280del`) to get its genomic start and end positions.
4. Finally, request the gene symbol associated to the transcript.

In weekly work discussions between the LOVD team and the Mutalyzer team, the issue of getting the transcript position data from Mutalyzer was communicated to the Mutalyzer development team. They agreed that this was difficult to do with their current web services. A few weeks later, a new version of Mutalyzer was released which included a web service to provide all information about all transcripts in a given chromosomal range. With the implementation of the new web service, the mapping script made over 60% less calls to Mutalyzer's web services which sped the script up a lot.

Now that a list of transcripts has been built, the script starts looping the selected variants. The script first checks with which transcripts each variant has some overlap. If the variant hits any genes that are already present in the LOVD database, the script will map it onto all transcripts for all those genes that are already in the database.

3. Adding genes to the database

If the uploader of the variants opted to create new gene entries as needed, the script will add all genes which are not in LOVD but are hit by the variant so the variant can be mapped onto all of those.

Most gene-specific information in LOVD, which is stored in the 'genes' table in the database, comes from the HUGO Gene Nomenclature Committee (HGNC)[7]. The HGNC's most important task is to approve and assign standardised gene symbols. The code that retrieves information from the HGNC database was partly modified for the mapping script, so that it can find the current gene symbol in case the HGNC has deprecated a specific entry. The retrieved information includes the current gene symbol, gene name, chromosomal position (number and band), IDs for the OMIM, Entrez and HGNC databases, and a transcript accession number. The transcript accession number retrieved from the HGNC is either the most studied transcript, or otherwise the longest for that gene.

Besides this, LOVD stores an `LRG_` or `NG_` accession number, which identifies the genomic reference sequence of the gene much like `NM_` accessions identify transcript sequences. If no `LRG_` or `NG_` accession number is available for this gene, the `NC_` accession number of the chromosome is used instead. This genomic accession number is retrieved from a file on the LOVD website, which mirrors data available at the NCBI. Additionally, for communications with Mutalyzer, LOVD needs to query Mutalyzer for a special `UD_` identifier that refers to a portion of the chromosome covering the gene and its flanking regions.

With this `UD_` identifier, a detailed list of all the gene's transcripts is requested from Mutalyzer. The script finds the transcript provided by the HGNC in this list and inserts the associated data into the 'transcripts' table in the database.

4. Mapping the variants to the transcripts
Finally, the script calls three other Mutalyzer web services to get the HGVS descriptions and the positions of the variant on this specific transcript and its protein product.

1. Mutalyzer's `mappingInfo` web service provides the positions of the variant relative to the transcript reference sequence.
2. Then, the `numberConversion` web service converts the `g.` HGVS description to a `c.` description on the transcript level.
3. Finally, the `runMutalyzer` web service is used to retrieve a `p.` HGVS description on the protein level for the `c.` description that was provided by the `numberConversion` web service.

Once the script has fetched this additional information, it creates a new entry in the 'variants_on_transcripts' table to store it in the database.

2.3 SeattleSeq Annotation files

After the VCF import support and the ability to map imported variants to transcripts, the SeattleSeq Annotation file format was the second NGS data format to be supported by LOVD.[3]

2.3.1 SeattleSeq Annotation file format

Like VCF, SeattleSeq files are also tab-delimited text files. SeattleSeq files can have metadata (or commentary) lines at the beginning and at the end of the file, which need to be skipped. They can be detected easily because they begin with a single hash character (`#`). The line containing the column headers also begins with a hash character; it is the last metadata line at the beginning of the file.

After that, each line in the file represents a specific location on a chromosome, much like it is done in VCF files. However, if a variant hits more than one transcript, the line is duplicated for each transcript and only the transcript-specific columns are different on the duplicated lines.

Missing values are substituted by `none`, `unknown`, or `NA` (for 'Not Available') — whichever is considered to be most appropriate in each case.

Most of the columns in the file are optional and can be selected as output options on the SeattleSeq Annotation submission form. Some of the default columns include `chromosome`, `position`, `rsID` (for dbSNP), `referenceBase`, `sampleGenotype`, (transcript) `accession`, `aminoAcids`, and `functionGVS` and `functionDBSNP` for functional effects.

SNP files

SeattleSeq Annotation files come in two flavours: SNP files (Single Nucleotide Polymorphism) which contain only single-base substitutions, and indel files (insertion/deletion) which can contain all other genetic variations. Because the exact format of the indel files depends largely on the original input that was sent to SeattleSeq, the import of indel files was added later; earlier versions of the SeattleSeq import script only supported SNP files.

For SNP files, the `sampleGenotype` column may contain an 'M', 'R', 'W', 'S', 'Y', or 'K' besides the usual bases which represent heterozygous cases. See Table 5 for their meanings.

Letter	Represents	Meaning
R	A and G	puRine
Y	C and T	pYrimidine
M	A and C	aMino
K	G and T	Keto
S	C and G	Strong chemical interaction
W	A and T	Weak chemical interaction

Table 5: Bases representing heterozygosity in SeattleSeq files.

Indel files

Adding indel file support was difficult because SeattleSeq's indel output format depends on the exact input when the file was submitted to the SeattleSeq Annotation server.

For example, SeattleSeq files based on BED files have radically different values in the `sampleGenotype` column. The column contains `D1` for a deletion of one base and `I1` for an insertion of one base. For deletions, the `referenceBase` column contains the deleted bases. However, for insertions, the `referenceBase` column contains only the flanking bases, separated by a hyphen (`-`). The inserted bases may be found in the optional `sampleAlleles` column because SeattleSeq copies the raw BED data to that column if it is present. If the column is missing, LOVD will produce an HGVS description like `g.12345_12346ins3` for an insertion of 3 bases, since the SeattleSeq file doesn't specify *which* bases were inserted. If the column is present, the inserted bases are included in the HGVS description, and single-base duplications are described as such.

For SeattleSeq files created from VCF files with only indels, SeattleSeq provides the option to return either 'SeattleSeq Annotation original allele columns' or 'VCF-like allele columns'. In both cases, the `sampleGenotype` column contains the exact alleles (e.g. `CT/C` for a heterozygous deletion of T). The latter option simply copies the `REF` and `ALT` columns from the VCF file to the `referenceBase` and `sampleAlleles` (if present) columns.

With ‘SeattleSeq Annotation original allele columns’ however, SeattleSeq modifies the data in the `referenceBase` column to be similar to what it looks like for SeattleSeq files created from BED files. The `sampleAlleles` column, if present, contains codes similar to the `sampleGenotype` column for BED files. In any case that is more than a simple deletion or insertion (delins variants and compound heterozygous cases are common examples), the `referenceBase` column contains only the *second* base from the `REF` column. The `sampleAlleles` column contains `CX` (for ‘complex’) or sometimes even `unknown` in these cases. This is a very problematic format, since it makes deletions, insertions and substitutions indistinguishable from one another.

Furthermore, the two allele column formats can only be distinguished from each other if the `sampleAlleles` column is provided or an insertion is present in the file. The SeattleSeq import script will try to detect which SeattleSeq format is used. If the script runs into an insertion that is annotated in the ‘SeattleSeq Annotation original allele columns’ format, the whole import is reverted and a warning message is displayed.

For files with ‘VCF-like allele columns’, the script identifies the alternate alleles from the `sampleGenotype` column and then generates HGVS descriptions using the same code as the VCF import.

2.3.2 Importing SeattleSeq files

To be able to store the transcript-specific variant data that is included in SeattleSeq Annotation files, the SeattleSeq file importer must map the variants ‘on-the-fly’ while the import is running. This may make importing SeattleSeq files a slow process, especially if many gene and transcript entries need to be created at the same time. Therefore, users are given the option not to create any genes and/or transcripts during the import, if their genes of interest are already in the database.

A special function was written to process the file. This function returns an array of key-value pairs for one *genomic* variant on each call. The lines of data in the transcript-specific columns are grouped into sub-arrays. This way, the rest of the script can easily access data fields using the column names.

The function that reads the variants detects the header line on the first call and remembers it internally (using a ‘static function variable’ that is preserved across function calls) for subsequent calls. Besides that, the function also saves the last-read line in a static variable. This is needed because it must read one line *too much* in order to know that it has read all lines that belong to the same genomic variant before returning the data.

After the script has read a variant, it first constructs a description in HGVS format. If the `sampleGenotype` is one of the six letters in Table 5, and the `referenceBase` is not one of the two bases represented by the `sampleGenotype`, both alleles have changed differently so the sample must be compound heterozygous. In this case, the script will split up the variant into two entries, one for each allele.

Then, the script will loop through the transcript accession numbers for the variant. For each accession number, the corresponding gene symbol is requested from Mutalyzer. These calls are cached, which means that the same accession number is not queried a second time, to speed up the script.

If this gene is not yet in the database, it must be added to be able to store the data about this variant on the gene's transcripts. Therefore, the script will prepare the gene's data in a format that is ready to be inserted into the database. The actual insertion will only be executed once the script is going to insert a variant mapped to this gene for sure; if for whatever reason it can't get sufficient data to map the variant, for example because Mutalyzer does not recognise the transcript, the gene won't be inserted either. The SeattleSeq import script gets the gene data from the HGNC database and a `UD_` identifier from Mutalyzer similarly to the automatic mapping script.

The next step is to get the transcript in the database, if it isn't already in there. Detailed data about all the gene's transcripts is retrieved from Mutalyzer using the `UD_` identifier of the gene. The data for each transcript is parsed into a format that is ready to be inserted into the database. The Mutalyzer call does not need to be made again for the same gene because a single call provides data for all of the gene's transcripts.

To prepare the 'variants_on_transcripts' entry, the script constructs the HGVS `p.` description of the variant on the protein using the data from the SeattleSeq file. The `c.` description is retrieved from Mutalyzer, along with transcript-based positions.

Finally, the variant entries that were constructed (usually one, but two in compound heterozygous cases) are inserted into the database. If the user was submitting the SeattleSeq file as part of a mutation screening, a 'screenings_to_variants' entry is created too. The script loops the 'variants_on_transcripts' entries to insert them into the database, also inserting any genes and transcripts that they rely on.

After this, the script is ready to read the next variant from the file.

2.4 Finding the same variant in other LOVDs

The last new feature described in this report is the 'Global Search' service, which allows users to search for a specific variant in *any* public LOVD installation worldwide. This allows users to find out quickly and easily whether a given variant has been reported and studied before.

The service makes use of a database hosted on LOVD.nl, which contains an index of 36,000 variants[1]. This database is updated weekly by querying the APIs of all known LOVDs to see if they have updated information.

The list of known LOVD installations is kept as follows. When LOVDs query LOVD.nl for software updates, they send some information about themselves to the website. This information includes the URL for the LOVD, their current software version, a unique signature, and various statistical data like the number of genes and variants in the database. The script that updates the central variant database makes use of this list of LOVDs to be able to find them.

Although this central variant database already existed, it was not used for anything in LOVD 2.0. Thus, a script was yet to be made to expose the information in the database in a simple API so LOVD 3.0 can access the data. The API can be called with a number of parameters as listed in Table 6 to narrow down the search. Since the `position` parameter is

[1] As of 15 March 2012

mandatory, it is not possible to extract bulk data from the database by calling the API without parameters.

Name	Values	Description
build	hg18 or hg19	Return results in the given human genome build. Defaults to hg19.
symbol	A gene symbol	Return only variants that are found within the given gene.
position	[chrX:][g.]*start*[_*end*]	Mandatory field. Return only results in a specific position. The chromosome may be omitted if a gene symbol is specified.
position_match	exact, partial, or exclusive	Selects whether the positions must be matched exactly, must overlap partially, or must overlap entirely, respectively. Defaults to exact.

Table 6: Search options for the Global Search service.

However, the API will never return more than 50 variants and it is designed to provide only minimal information. Instead, users are directed to the originating LOVD to find additional data. These two measures prevent abuse of the service and maintain traffic to the original data source, respectively.

3 Results

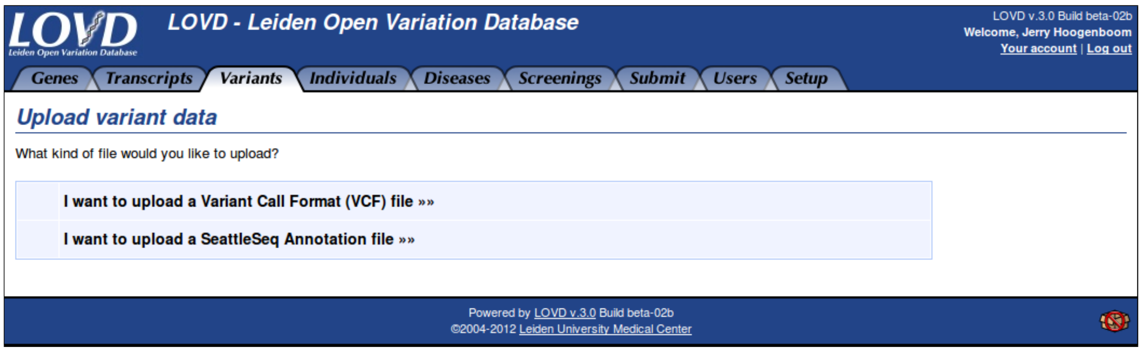
LOVD 3.0 has a user-friendly submission process that guides the user through the process of entering new information about the various data entities (individuals, phenotypes, screenings, variants, etc.) and links everything together. The process keeps track of the choices and data additions the user has made and sends a summary to the database curators and managers by email when the submission is finished.

3.1 Importing files

The VCF and SeattleSeq file upload options appear in the submission process only for managers, because they are the only users with the permissions to add large-scale data. Where other users only get the options to enter a variant on genomic level or genomic and transcript level, managers also see the option to upload a file with genome-scale information. After clicking the new option, they are taken to the file format selection as seen in Figure 3.

3.1.1 VCF file uploads

After clicking the Variant Call Format option, users are taken to the VCF upload form (Figure 4).


LOVD
Leiden Open Variation Database
LOVD - Leiden Open Variation Database
LOVD v.3.0 Build beta-02b
Welcome, Jerry Hoogenboom
Your account | Log out
Genes
Transcripts
Variants
Individuals
Diseases
Screenings
Submit
Users
Setup

Upload variant data

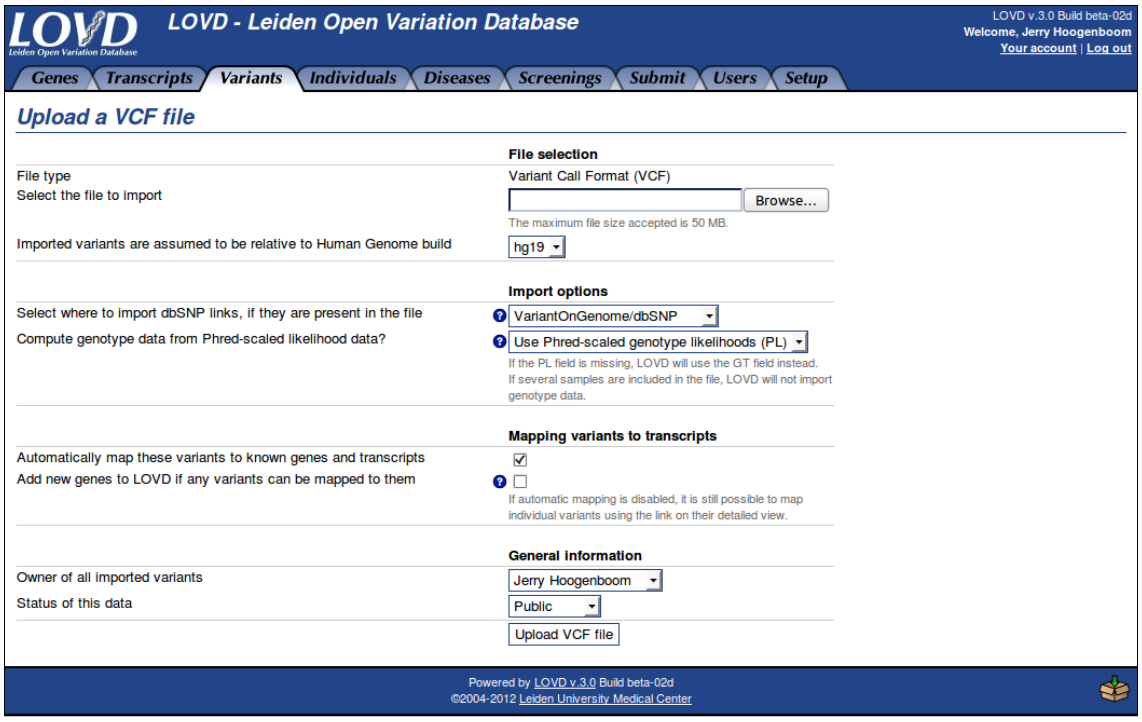
What kind of file would you like to upload?

I want to upload a Variant Call Format (VCF) file »»

I want to upload a SeattleSeq Annotation file »»

Powered by LOVD v.3.0 Build beta-02b
©2004-2012 Leiden University Medical Center

Figure 3: File format selection in the LOVD 3.0 submission process.


LOVD
Leiden Open Variation Database
LOVD - Leiden Open Variation Database
LOVD v.3.0 Build beta-02d
Welcome, Jerry Hoogenboom
Your account | Log out
Genes
Transcripts
Variants
Individuals
Diseases
Screenings
Submit
Users
Setup

Upload a VCF file

	File selection
File type	Variant Call Format (VCF)
Select the file to import	Browse... The maximum file size accepted is 50 MB.
Imported variants are assumed to be relative to Human Genome build	hg19
	Import options
Select where to import dbSNP links, if they are present in the file	VariantOnGenome/dbSNP
Compute genotype data from Phred-scaled likelihood data?	Use Phred-scaled genotype likelihoods (PL) If the PL field is missing, LOVD will use the GT field instead. If several samples are included in the file, LOVD will not import genotype data.
	Mapping variants to transcripts
Automatically map these variants to known genes and transcripts	☑
Add new genes to LOVD if any variants can be mapped to them	☐ If automatic mapping is disabled, it is still possible to map individual variants using the link on their detailed view.
	General information
Owner of all imported variants	Jerry Hoogenboom
Status of this data	Public
	Upload VCF file

Powered by LOVD v.3.0 Build beta-02d
©2004-2012 Leiden University Medical Center

Figure 4: The Variant Call Format upload form.

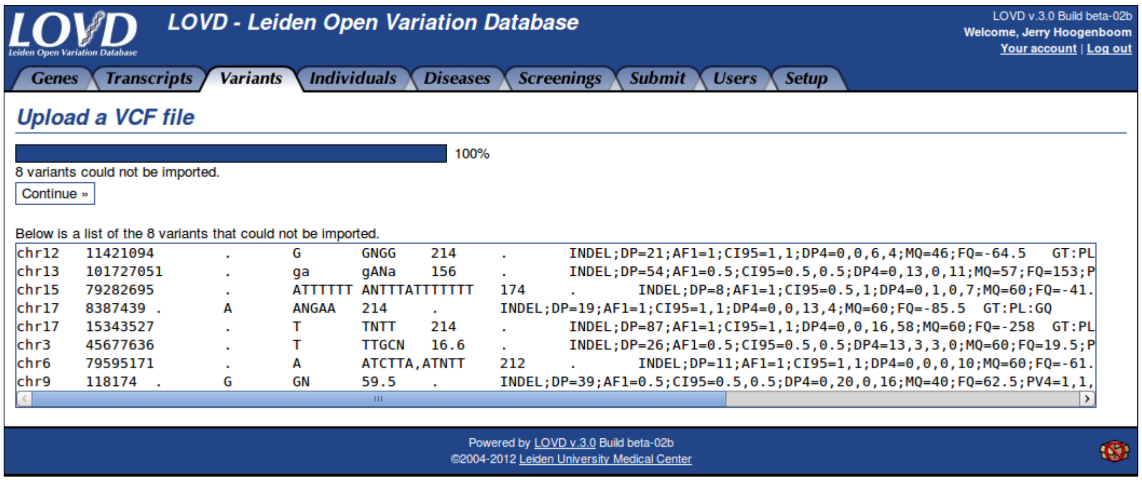
The form reports the maximum possible file size that can be uploaded. This depends on the settings of the web server; PHP's settings `post_max_size` and `upload_max_filesize` are checked and an additional maximum size of 50 MB is specified in LOVD's source code.

Below the file selection field, the user is reminded about the database's human genome build configuration. This is done because the import script and the automatic mapping both blindly interpret all positions to be relative to that build of the genome since there is no standardised way to mention the build in VCF and SeattleSeq files. Because of this, uploading a VCF file that was generated with respect to a different build of the genome would flood the database with nonsense. The human genome build is presented as a selection list with only a single option, because this was the best-looking way to place this note in the form without bulky message boxes.

Users can select in which data column they want to import dbSNP identifiers, if they are present in the VCF file. The drop-down list contains all custom columns that accept dbSNP custom links (the columns and links can be set up in the LOVD setup area). This way, LOVD will automatically generate clickable links to dbSNP when the data is viewed.

In the 'Mapping variants to transcripts' section of the upload form, users may select whether they want to make use of the automatic mapping functions. The first check box sets the `ALLOW` flag for all imported variants, whereas the second sets the `ALLOW_CREATE_GENES` flag (see Table 4 on page 11).

After importing the VCF file, LOVD guides the user automatically to the next step in the submission process. However, if some variants could not be imported, the user is informed about this and the lines that could not be imported are shown on-screen. This can be seen in Figure 5, where the uploaded VCF file contained 8 lines with 'N' bases in the `ALT` column. Because these variants are not clear, these lines were rejected from the import.


LOVD
Leiden Open Variation Database
LOVD - Leiden Open Variation Database
LOVD v.3.0 Build beta-02b
Welcome, Jerry Hoogenboom
Your account | Log out
Genes
Transcripts
Variants
Individuals
Diseases
Screenings
Submit
Users
Setup
Continue »

Upload a VCF file

100%

8 variants could not be imported.

Below is a list of the 8 variants that could not be imported.

```
chr12   11421094        .       G       GNGG    214     .       INDEL;DP=21;AF1=1;CI95=1,1;DP4=0,0,6,4;MQ=46;FQ=-64.5   GT:PL
chr13   101727051       .       ga      gANa    156     .       INDEL;DP=54;AF1=0.5;CI95=0.5,0.5;DP4=0,13,0,11;MQ=57;FQ=153;P
chr15   79282695        .       ATTTTTT ANTTTATTTTTTT   174     .       INDEL;DP=8;AF1=1;CI95=0.5,1;DP4=0,1,0,7;MQ=60;FQ=-41.
chr17   8387439 .       A       ANGAA   214     .       INDEL;DP=19;AF1=1;CI95=1,1;DP4=0,0,13,4;MQ=60;FQ=-85.5  GT:PL:GQ
chr17   15343527        .       T       TNTT    214     .       INDEL;DP=87;AF1=1;CI95=1,1;DP4=0,0,16,58;MQ=60;FQ=-258  GT:PL
chr3    45677636        .       T       TTGCN   16.6    .       INDEL;DP=26;AF1=0.5;CI95=0.5,0.5;DP4=13,3,3,0;MQ=60;FQ=19.5;P
chr6    79595171        .       A       ATCTTA,ATNTT    212     .       INDEL;DP=11;AF1=1;CI95=1,1;DP4=0,0,0,10;MQ=60;FQ=-61.
chr9    118174  .       G       GN      59.5    .       INDEL;DP=39;AF1=0.5;CI95=0.5,0.5;DP4=0,20,0,16;MQ=40;FQ=62.5;PV4=1,1,
```

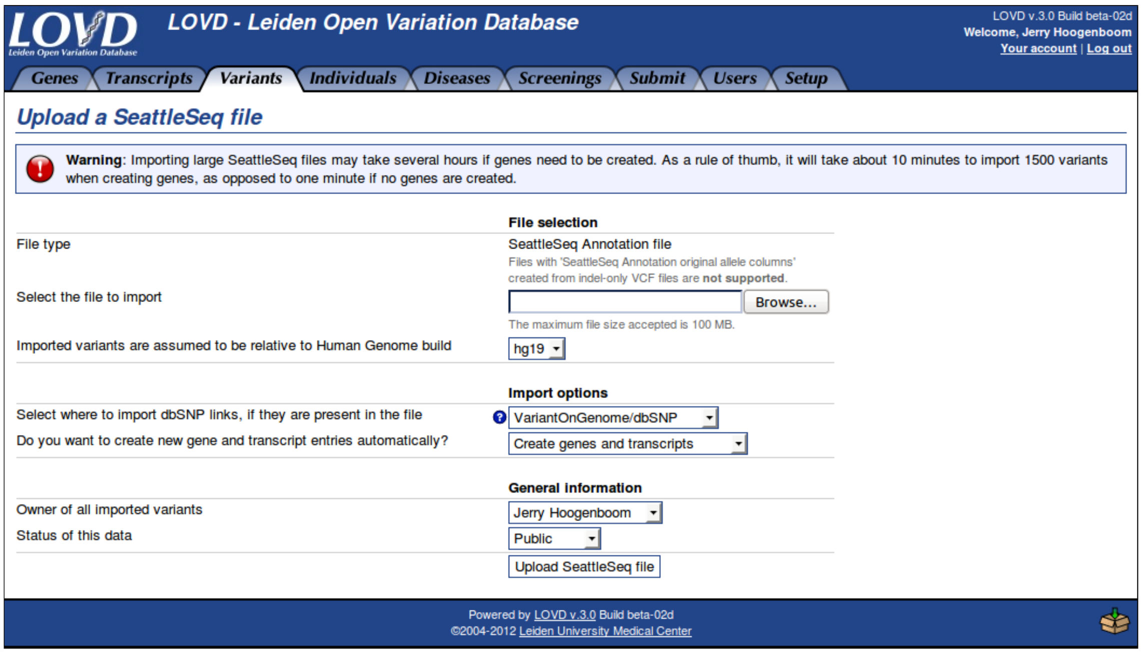
Powered by LOVD v.3.0 Build beta-02b
©2004-2012 Leiden University Medical Center

Figure 5: The user is informed of the fact that some lines in a VCF file could not be processed.

It is also worth to note that the line 'Below is a list of . . . ' is made entirely context-dependent which keeps this 'bad news message' somewhat more friendly. For example, the text changes to 'Below is the variant that could not be imported' if just one variant could not be imported. On the other hand, if many variants could not be imported, the text changes to 'Below is a list of the first 100 of . . . ' to reflect that not all of them are shown. In LOVD, phrases like '1 variant(s)' are always avoided.

3.1.2 SeattleSeq file uploads

The SeattleSeq file upload form shares most of its code with the VCF file upload form, but it shows slightly different options and adds a number of new things. The hard-coded maximum file size for SeattleSeq Annotation files is 350 MB, although this is still limited to the server's settings. A screenshot of the SeattleSeq upload form can be seen in Figure 6.



Upload a SeattleSeq file

Warning: Importing large SeattleSeq files may take several hours if genes need to be created. As a rule of thumb, it will take about 10 minutes to import 1500 variants when creating genes, as opposed to one minute if no genes are created.

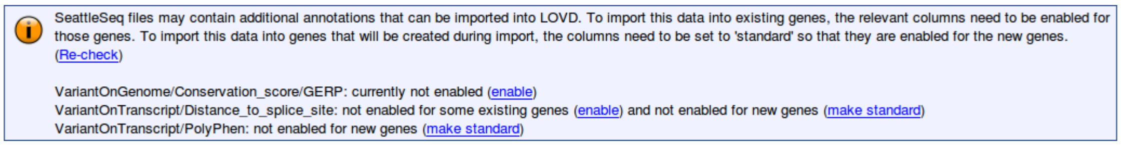
	File selection
File type	SeattleSeq Annotation file Files with 'SeattleSeq Annotation original allele columns' created from indel-only VCF files are **not supported**.
Select the file to import	Browse... The maximum file size accepted is 100 MB.
Imported variants are assumed to be relative to Human Genome build	hg19
	Import options
Select where to import dbSNP links, if they are present in the file	VariantOnGenome/dbSNP
Do you want to create new gene and transcript entries automatically?	Create genes and transcripts
	General information
Owner of all imported variants	Jerry Hoogenboom
Status of this data	Public
	Upload SeattleSeq file

Powered by LOVD v.3.0 Build beta-02d
©2004-2012 Leiden University Medical Center

Figure 6: The SeattleSeq Annotation file upload form.

Users may select whether they want the importer to 'Create genes and transcripts', 'Create transcripts only', or 'Do not create genes or transcripts' at all. A warning at the top of the upload form informs users of the fact that creating genes may significantly slow down the process.

Special care had to be taken of the additional information that SeattleSeq files provide. For this, LOVD 3.0 comes with a number of preconfigured custom data columns that database managers can activate and modify. The data in the SeattleSeq file can't be imported if the columns are not enabled. Therefore, the SeattleSeq import script checks for their existence and displays any columns that may be disabled in an extra notification box on the form, as seen in Figure 7. The notification box also provides some simple tools to set up the columns so the user doesn't have to go to the setup area to set them up correctly.


SeattleSeq files may contain additional annotations that can be imported into LOVD. To import this data into existing genes, the relevant columns need to be enabled for those genes. To import this data into genes that will be created during import, the columns need to be set to 'standard' so that they are enabled for the new genes. (Re-check)

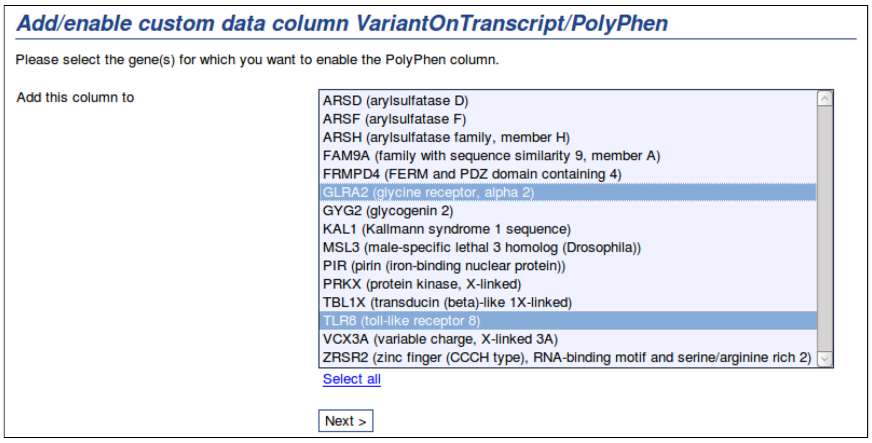
VariantOnGenome/Conservation_score/GERP: currently not enabled (enable)
VariantOnTranscript/Distance_to_splice_site: not enabled for some existing genes (enable) and not enabled for new genes (make standard)
VariantOnTranscript/PolyPhen: not enabled for new genes (make standard)

Figure 7: The user is informed of a number of SeattleSeq data columns that need to be activated.

The notification is generated by an AJAX script that checks the columns. The 'Re-check' link simply refreshes the notification box by requesting a new one using AJAX.

The 'Make standard' links are also implemented using AJAX, since they only require a small change in the 'columns' table. When the user clicks a 'Make standard' link, the browser does an AJAX request for a script that makes this change. The server sets the column to standard and reports back to the browser with a status code that specifies whether the change was successful. If it was, the notification box is refreshed, otherwise the user is prompted with the message that the column could not be edited. With normal use of LOVD, this can only happen if the user has been logged out because they haven't done anything for a long period of time after opening the upload form.

Because enabling columns may require the database table to be changed physically, the 'Enable' links open a small pop-up window asking for the user's password for extra authorisation. Additionally, in the case of 'VariantOnTranscript' columns, the user is also asked to select the genes for which the column is to be activated. LOVD will create an entry in the 'shared_columns' table for each gene that is selected, to enable the column for that gene. The pop-up, as seen in Figure 8, shows the same interface as what is used in the column set-up in the LOVD setup area, albeit with minimal headers and footers. If the database contains a lot of information, the user is notified that the operation may take more than a few seconds. In this time, the database will be locked (and thus unusable for anyone) because its structure needs to be changed to add the column. After the column is enabled, the pop-up closes itself and refreshes the notification box using AJAX.


Add/enable custom data column VariantOnTranscript/PolyPhen

Please select the gene(s) for which you want to enable the PolyPhen column.

Add this column to

ARSD (arylsulfatase D)
ARSF (arylsulfatase F)
ARSH (arylsulfatase family, member H)
FAM9A (family with sequence similarity 9, member A)
FRMPD4 (FERM and PDZ domain containing 4)
GLRA2 (glycine receptor, alpha 2)
GYG2 (glycogenin 2)
KAL1 (Kallmann syndrome 1 sequence)
MSL3 (male-specific lethal 3 homolog (Drosophila))
PIR (pirin (iron-binding nuclear protein))
PRKX (protein kinase, X-linked)
TBL1X (transducin (beta)-like 1X-linked)
TLR8 (toll-like receptor 8)
VCX3A (variable charge, X-linked 3A)
ZRSR2 (zinc finger (CCCH type), RNA-binding motif and serine/arginine rich 2)

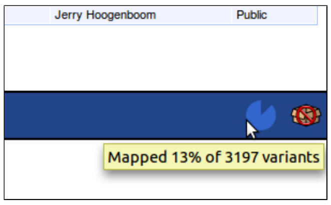
Figure 8: This pop-up allows users to quickly enable columns before they upload a SeattleSeq file.

During the import, the user sees a progress bar similar to the one in Figure 5. However, although the VCF import progress bar only shows 'Loading variant data from the file . . . ', the SeattleSeq progress bar also shows the number of variants that have been processed. When the SeattleSeq import is busy adding genes or transcripts, an extra line like 'Loading gene information for DMD . . . ' provides additional detail. This is done to make it more evident that the script is actually working, because the progress bar fills quite slowly for large imports that create many gene entries.

At the end of the import, the user sees a screen quite similar to Figure 5.

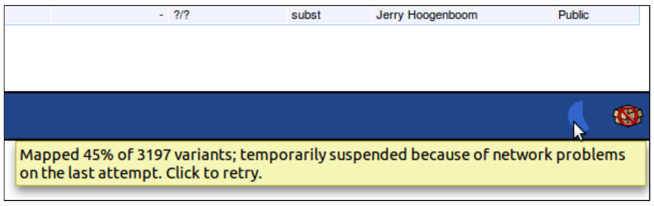
3.2 Automatic mapping

Because the automatic mapping functionality is running in the background, progress information is communicated to the user by means of a small circular progress meter in the footer of every page. Additional textual progress information is shown when the user moves the mouse pointer over the progress meter, as illustrated in Figure 9.



Mapped 13% of 3197 variants

Figure 9: The automatic mapping progress meter.

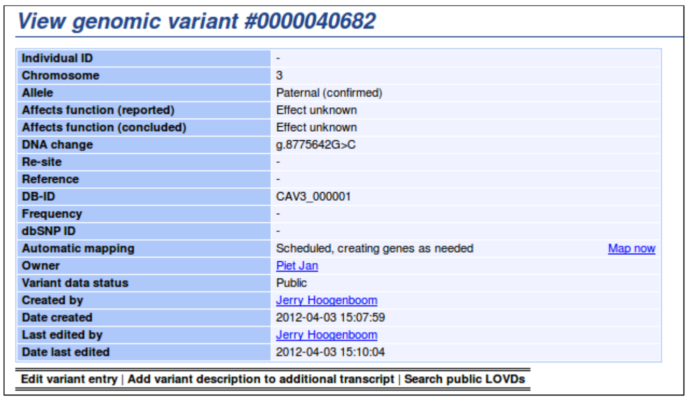


Mapped 45% of 3197 variants; temporarily suspended because of network problems on the last attempt. Click to retry.

Figure 10: Automatic mapping can be suspended temporarily.

In case automatic mapping fails to connect to Mutalyzer, it is suspended for one hour. When this happens, the progress information looks like Figure 10.

Additionally, a single variant can also be mapped on demand at the variant's entry page in LOVD. The variant entry page lists the automatic mapping settings for that particular variant (usually 'Off', 'Scheduled', or 'Done'; possibly also 'creating genes as needed'). A link is provided to map the variant immediately, which calls the mapping script using AJAX and passes the variant ID that needs to be mapped. When the server reports back, the page is reloaded to reflect the changes. A screenshot of a variant entry with mapping information can be seen in Figure 11.



View genomic variant #0000040682

Individual ID	-
Chromosome	3
Allele	Paternal (confirmed)
Affects function (reported)	Effect unknown
Affects function (concluded)	Effect unknown
DNA change	g.8775642G>C
Re-site	-
Reference	-
DB-ID	CAV3_000001
Frequency	-
dbSNP ID	-
Automatic mapping	Scheduled, creating genes as needed Map now
Owner	Piet Jan
Variant data status	Public
Created by	Jerry Hoogenboom
Date created	2012-04-03 15:07:59
Last edited by	Jerry Hoogenboom
Date last edited	2012-04-03 15:10:04

Edit variant entry | Add variant description to additional transcript | Search public LOVDs

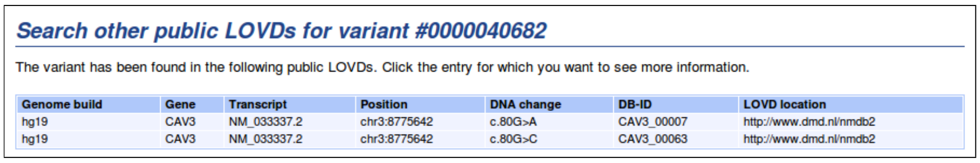
Figure 11: Mapping information displayed on a variant entry page.

In case the mapping script fails to connect to Mutalyzer when processing the variant on demand, the ERROR flag is set. The automatic mapping line will then include 'encountered a problem on the last attempt' to make this clear to the user.

Also, the text is made entirely context-dependent. For example, the 'Map now' link changes to 'Map again' on success or 'Retry' on failure.

3.3 Global Search

The ‘Search public LOVDs’ button that is visible in Figure 11 is part of the Global Search service. Clicking it will open a pop-up window which queries the service for this variant and lists the results. The pop-up window is seen in Figure 12. Clicking any of the results opens the variant’s page in that particular LOVD installation in a new window.



Search other public LOVDs for variant #0000040682

The variant has been found in the following public LOVDs. Click the entry for which you want to see more information.

Genome build	Gene	Transcript	Position	DNA change	DB-ID	LOVD location
hg19	CAV3	NM_033337.2	chr3:8775642	c.80G>A	CAV3_00007	http://www.dmd.nl/nmdb2
hg19	CAV3	NM_033337.2	chr3:8775642	c.80G>C	CAV3_00063	http://www.dmd.nl/nmdb2

Figure 12: A pop-up window shows hits from the Global Search service.

4 Discussion

4.1 Relying on web services

Many features new to LOVD 3.0, including automatic variant mapping and SeattleSeq file import, rely on a number of web services. Also for more fundamental tasks, like adding new genes or transcripts, LOVD 3.0 downloads data from over the internet.

The dependency on these web services has become painfully clear in the first days of 2012, as the HGNC database was unreachable for days. Because of this, it was entirely impossible to create new gene entries in LOVD 3.0 installations.

New plans have been made to solve these problems by mirroring the required data on the LOVD.nl website. LOVD already mirrors lists of `NG_` and `LRG_` accession numbers for genes. When the HGNC database is mirrored on LOVD.nl, some code needs to be modified to use the mirror when LOVD detects that the HGNC database is down.

4.2 AJAX and PHP sessions

LOVD uses PHP’s session features to keep temporary data about a user’s session. This data is stored in a temporary file on the web server, which is linked to a specific user by sending them a cookie with a session key. A script can read and modify the data in the session file once the `session_start()` function is called.

To prevent race conditions, where two scripts are writing to the same session file losing each other’s changes, PHP locks the file while a script is using it. Other PHP scripts will block on the `session_start()` call until the lock is removed from the file. Normally, this is not a problem since a user would usually only browse one page at a time and pages load very quickly.

However, this turned out to be a problem for the automatic mapping script, which uses the session file to store progress data. Since a single run of the mapping script can take up to 20 seconds to complete, it would keep the session file locked for a long time. In this time, the user could not load any LOVD page and the searching and sorting features of data tables would become unresponsive too.

It was not easy to find out why PHP's session handling was causing these delays. By printing the execution time of each code block, the delays could be isolated to a single line of code: the `session_start()` call. Since calls to the mapping script are done automatically through AJAX, it took some time to realise that the mapping script would still be running when trying to leave the page that called it.

The problem was solved by using the `session_write_close()` function at the beginning of the mapping script, which closes the session file and removes the lock. At the end of the mapping script, the session is reopened to write the progress data.

The same workaround was later used in the SeattleSeq import script. Since importing SeattleSeq files may take a long time, the session is closed while the script is working to enable the user to browse the database in other tabs.

4.3 Selecting the 'best' transcript for a gene

When mapping variants to a newly inserted gene, the mapping script needs to select a transcript relative to which the variants will be described. The HGNC database provides two transcript accession numbers for most genes – one 'curated' transcript accession which is usually the most studied or otherwise 'most important' transcript, and one 'mapped' accession number which is retrieved from the NCBI automatically and is always the longest transcript for the gene.

The mapping script currently selects the 'curated' transcript accession number if it is available, using the 'mapped' accession number otherwise. In case the HGNC does not provide a transcript accession number in either data field, the gene is currently ignored. Since this is rather bad behaviour, the mapping script can be improved by falling back on another accession in these cases.

5 Future steps

5.1 Mapping script improvements

Although the current version of the mapping script does its job quite well, there are still a few points that can be improved in the future.

5.1.1 Getting RNA descriptions

Automatic mapping currently only maps `g.` HGVS descriptions to `c.` and `p.` descriptions. LOVD provides a field for the `r.` description for variants on transcripts, which is even mandatory on the manual input form, but it remains empty through the automatic mapping script. The same holds true for variants imported from SeattleSeq files.

The `r.` description is not filled in because Mutalyzer does not yet support creating these descriptions from their web services. Once support for this is implemented into Mutalyzer, LOVD can make use of the new service to 'complete' the mapping script.

5.1.2 Controlling the mapping script

Like the `mapping_flags` column in the 'variants' database table, the LOVD system could use a configuration column to store status flags about the mapping script itself. For example, there is currently no way for the database administrator to turn off the mapping script altogether if they wish to do so.

Also, the mapping script still makes use of the user's session file to store information like whether mapping has finished or whether an error occurred. If a user logs off or on, this information is lost and the mapping script is run again. A minor issue with this occurs if the Mutalyzer web service goes down temporarily, in which case the mapping script will create an identical error log entry in LOVD's system logs for *every* user currently online.

5.1.3 Making better use of AJAX for the 'Map now' link

The 'Map now' link on variant entry pages currently calls the mapping script with AJAX. However, once the script has run, the whole page is refreshed to display the new mappings. AJAX can be used to refresh only those parts of the page that need to be changed.

This was not yet implemented because for this a number of complex changes need to be made to LOVD's code to allow these parts of the page to be reloaded by AJAX.

5.2 New features

Now that LOVD 3.0 allows for NGS data import, new powerful tools need to be created to handle the amount of data in LOVD effectively. The mapping script automates a lot of work to provide annotations for the variants, but new tools can be made to make it easier to find those few disease-causing mutations.

Besides simply listing the variants in the database, LOVD can be a great analysis tool with the use of intelligent new views on the data. For example, disease-causing mutations can be detected by sequencing a patient and their parents and then looking at the variants that are only present in the patient. Such an overview can be added to LOVD easily by allowing users to filter a list of variants based on the individuals they are connected to. The overview could be further improved by including variants for which the patient is homozygous, whereas their parents are heterozygous.

To create these overviews, only small changes need to be made to the LOVD 3.0 source code, since filtering and searching variants (or any other data in any overview) is already fully implemented.

6 Acknowledgements

Many thanks go to Ivo Fokkema and Ivar Lugtenburg, the main LOVD developers, for their guidance, input, and support throughout the project. And of course I would also like to thank the Mutalyzer developers, Martijn Vermaat and Jeroen Laros, for their help and work.

References

[1] Fokkema IF, Taschner PE, Schaafsma GC, Celli J, Laros JF, and den Dunnen JT. LOVD v.2.0: the next generation in gene variant databases. *Hum Mutat*, 32(5):557–63, May 2011.

[2] Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST, McVean G, and Durbin R. The variant call format and VCFtools. *Bioinformatics*, 27(15):2156–8, Aug 2011.

[3] SeattleSeq Annotation server.
`http://snp.gs.washington.edu/SeattleSeqAnnotation134/`.

[4] den Dunnen JT and Antonarakis SE. Mutation nomenclature extensions and suggestions to describe complex mutations: A discussion. *Hum Mutat*, 15(1):7–12, Jan 2000.

[5] Wildeman M, van Ophuizen E, den Dunnen JT, and Taschner PE. Improving sequence variant descriptions in mutation databases and literature using the MUTALYZER sequence variation nomenclature checker. *Hum Mutat*, 29(1):6–13, Jan 2008.

[6] Sherry ST, Ward MH, Kholodov M, Baker J, Phan L, Smigielski EM, and Sirotkin K. dbSNP: the NCBI database of genetic variation. *Nucleic Acids Res*, 29(1):308–11, Jan 2001.

[7] Seal RL, Gordon SM, Lush MJ, Wright MW, and Bruford EA. genenames.org: the HGNC resources in 2011. *Nucleic Acids Res*, 39(Database issue):D514–9, Jan 2011.