PeptoVar USER’S MANUAL

OVERVIEW

PeptoVar (**Pept**ides **o**f **Var**iations) is a pure Python3 program for personalization of protein coding genes, and population-wide peptidome generation. PeptoVar can be used for *in-silico* prediction of new minor histocompatibility antigen (MiHa), simulation of stem sells transplantation or as peptide database generator for shotgun proteomic (Fig. 1).

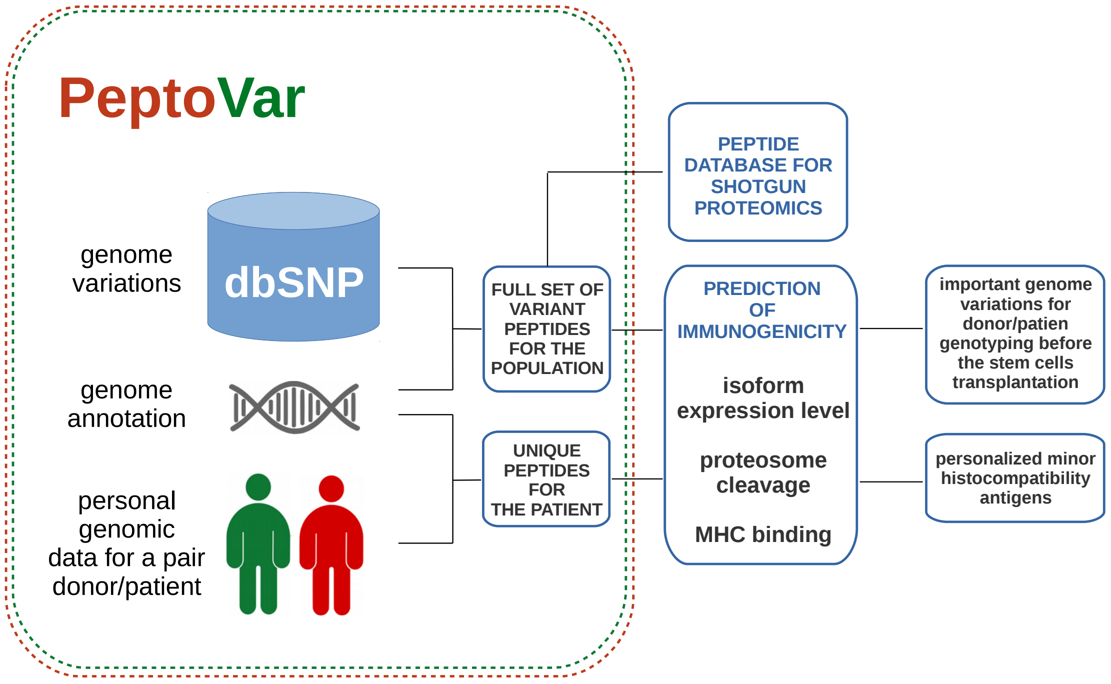
PeptoVar
genome variations
dbSNP
genome annotation
personal genomic data for a pair donor/patient
FULL SET OF VARIANT PEPTIDES FOR THE POPULATION
UNIQUE PEPTIDES FOR THE PATIENT
PEPTIDE DATABASE FOR SHOTGUN PROTEOMICS
PREDICTION OF IMMUNOGENICITY
isoform expression level
proteosome cleavage
MHC binding
important genome variations for donor/patien genotyping before the stem cells transplantation
personalized minor histocompatibility antigens

Fig. 1 The variant of a pipeline for peptidome generation and MiHa prediction.

Features:

- generate peptides for combinations of genome variations in the population
- make personalized peptidomes for selected samples (useful for data like 1000 Genome project)
- select unique peptides for the pair of samples (useful for simulation of the stem cell transplantation)
- determine variation translation dependencies
- classify variations (synonymous / non-synonymous) according to translation dependencies
- treat unphased variation data

Input data:

- genome annotation
- set of genomic variations

REQUIREMENTS

Platforms:

Linux, Mac OS

Dependencies:

python >= 3.5

pysam module >= 0.11.2.2

INSTALLATION

1) install pysam module with pip:

```
pip3 install pysam
```

2) download latest stable PeptoVar build from the project release page:

   https://github.com/DMalko/PeptoVar/releases/latest

   and unzip the archive

or clone PeptoVar using git:

```
git clone https://github.com/DMalko/PeptoVar
```

3) add resulting folder to your ``PATH`` variable

or add symbolic link for ``PeptoVar`` script to your ``bin`` folder

or use PeptoVar directly by specifying full path to the executable script

USAGE

```
PeptoVar.py [-h] [-gff file.gff] [-fasta file.fasta] [-vcf file.vcf.gz]
            [-tmpdir dirpath] [-samples sample_name1 [sample_name2]]
            [-minaf THRESHOLD] [-var all | used] [-nopt]
            [-peptlen LENGTH1 [LENGTH2 ...]] [-outdir dirpath]
            [-indir dirpath] [-trnlist transcriptID [transcriptID ...]] [-trnfile transcriptID.txt]
```

-h   show this help message and exit

-gff   GFF input file

-fasta   FASTA input file (use if GFF file has no sequences)

-vcf   VCF input file (requirements see in INPUT FILE FORMAT paragraph)

-tmpdir   TEMP directory

-samples   a sample name or a pair of names in VCF file; for two samples (donor/patient) only unique peptides will be represented

-minaf   allele frequency (AF) threshold; alleles with AF < THRESHOLD will be ignored (AF=0 will be set for alleles with no AF data) **NOTE: very low value of THRESHOLD or ignoring -minaf at all can cause high memory usage and increasing computational time**

-var (all | used)   save translated polymorphisms (all or only used to make peptides)

-nopt   do not use optimization, i.e. all alleles of synonymous variations will be used (may cause high CPU load and memory usage)

-peptlen   lengths of peptides (0 - full-length proteins)

-outdir   output directory (will be created if not exists, default: ./output)

-indir   input directory for files *.vcf.gz, *.vcf.gz.tbi, *.gff and *.fasta (if no sequences in GFF file); the files MUST have the same name for each locus (chromosome)

-trnlist   list of transcriptID for processing

-trnfile   one column text file with the transcriptID list for processing

INPUT FILE FORMATS

PeptoVar uses genome annotation data in GFF3 format (http://gmod.org/wiki/GFF3). Note that only records with CDS type are processed and grouped by Parent identifiers.

VCF files with genetic variation data must be compressed with bgzip (block compression/decompression utility) and indexed with tabix (generic indexer for TAB-delimited genome position files). The both utilities are parts of Samtools - a suite of programs for interacting with high-throughput sequencing data (http://www.htslib.org).

OUTPUT FILE FORMATS

PeptoVar creates four files in CSV format (TAB-delimited) with a header in the output directory: the file for peptides (if -peptlen option has a value more than 0), the file for proteins (if -peptlen option has 0 value), the file for polymorphisms (if -var option is utilized) and the file for warnings.

Data fields in the file for peptides (sample_name.pept.csv):
chrom – chromosome number (or locus name)
transcript_id – transcript ID
sample – sample name
sample_allele1 – sample allele 1 (‘0’ - not in allele 1; ‘1’ - in allele 1)
sample_allele2 – sample allele 2 (‘0’ - not in allele 2; ‘1’ - in allele 2)
beg – start genome position of the peptide (peptide shadow position, see Fig.4)
end – end genome position of the peptide (peptide shadow position, see Fig.4)
upstream_fshifts – frame shifts in upstream of the peptide
variations(positions_in_matrix) - polymorphisms inside the peptide with nucleotide positions
peptide – peptide sequence
matrix – genome sequence of translation to the peptide

Data fields in the file for proteins (sample_name.prot.csv):
chrom – chromosome number (or locus name)
transcript_id – transcript ID
sample – sample name
sample_allele1 – sample allele 1 (‘0’ - not in allele 1; ‘1’ - in allele 1)
sample_allele2 – sample allele 2 (‘0’ - not in allele 2; ‘1’ - in allele 2)
beg – start genome position of the protein
end – end genome position of the protein
variations(positions_in_matrix) – polymorphisms inside the proteins with nucleotide positions
protein – protein sequence
matrix – genome sequence of translation to the protein

Data fields in the file for polymorphisms (variations.csv):
transcript_id – transcript ID
variation_id – polymorphism ID
beg – start genome position of the polymorphism

end – end genome position of the polymorphism

allele_id – allele ID (polymorphism_ID:polymorphism_sequence=[allele_fequency](ref|alt), where ref - reference sequence, alt - alternative sequence)

sample – sample name

sample_allele1 – sample allele 1 (‘0’ - not in allele 1; ‘1’ - in allele 1)

sample_allele2 – sample allele 2 (‘0’ - not in allele 2; ‘1’ - in allele 2)

synonymous – ‘1’ if the polymorphisms is synonymous, else ‘0’

upstream_fshifts – frame shifts in upstream of the polymorphism

prefix_alleles – polymorphisms in the polymorphism prefix

prefix – sequence of the polymorphism prefix (the polymorphism sequence addition to get the first complete codon)

allele – sequence of the polymorphism

suffix – sequence of the polymorphism suffix (the polymorphism sequence addition to get the last complete codon)

suffix_alleles – polymorphisms in the polymorphism suffix

translation – translation of the polymorphism with the siffix and the prefix

***NOTE**: ID of synonymous polymorphism alleles are enclosed in square brackets*

File with warnings (warnings.csv):

PeptoVar inspect records in the VCF and the GFF files for collisions.

The most important ones:

- multiple polymorphisms with the same ID
- multiple polymorphisms with the same genome positions but different ID
- intersection of a polymorphism and the exon border (in this case we don’t know where is a site of splicing)
- no data for the sample in the VCF file
- no genome sequence in the exon position range

EXAMPLES

Personalized peptides for a sample (sample mode):

```
     PeptoVar –samples SAMPLE01 –peptlen 9 –var used –gff
./testdata/test.gff –vcf ./testdata/test.vcf.gz
```

or

```
     PeptoVar -samples SAMPLE01 -peptlen 9 -var used -indir
./testdata
```

Unique peptides for a pair of samples (transplantation mode):

```
     PeptoVar -samples SAMPLE01 SAMPLE02 -peptlen 9 -var used -gff
./testdata/test.gff -vcf ./testdata/test.vcf.gz
```

or

```
     PeptoVar -samples SAMPLE01 SAMPLE02 -peptlen 9 -var used
-indir ./testdata
```

Generation of peptides using all possible variations in the set (virtual sample mode):

```
     PeptoVar -peptlen 9 -var used -gff ./testdata/test.gff
-vcf ./testdata/test.vcf.gz
```

or

```
     PeptoVar -peptlen 9 -var used -indir ./testdata
```

UNDER THE HOOD

The main idea of PeptoVar is decrease computational complexity of translation nucleotide sequence with a generous amount of polymorphisms and determine their translation dependencies (allele combinations of frame shifts and closely located polymorphisms which are diversity factors for translation products).

The base steps of PeptoVar workflow:

1) reading input data

2) building graph for nucleotide sequence

3) insertion genomic polymorphisms to the graph as extra paths

4) attaching prefixes and suffixes to branching paths according to the reading frames (including frames induced from upstream frame shifts)

5) removing synonymous branches

6) translation the graph to peptides

7) substitution the peptide sets (for a pair of samples)

8) writing the results

The known public databases of polymorphisms have information about which genomic variations are non-synonymous and must be taken into consideration for population-wide peptidome construction. But the data is not comprehensive. Depending on combination of upstream frame shifts the same variation can be synonymous and non-synonymous (Fig. 2). More over, two polymorphisms in the same codon can cause translation dependency. For example, in the polymorphic codon (C,T)T(A,T) the third position is synonymous if the first position is ‘C’ and non-synonymous if the first position is ‘T’ (Fig. 3). The PeptoVar program save the dependency data into an own file. (Сказать, что это важно для поиска значимых для поиска MiHa и типирования на них?) In case lots of variations occur consecutively PeptoVar optimizes combinatorial task to reduce computational time and memory usage (Fig. 4). If a translated sequence starts and ends in genomic insertions its genome coordinates is not determined. PeptoVar use “peptide shadow” to set start and end genomic peptide positions (Fig. 5).

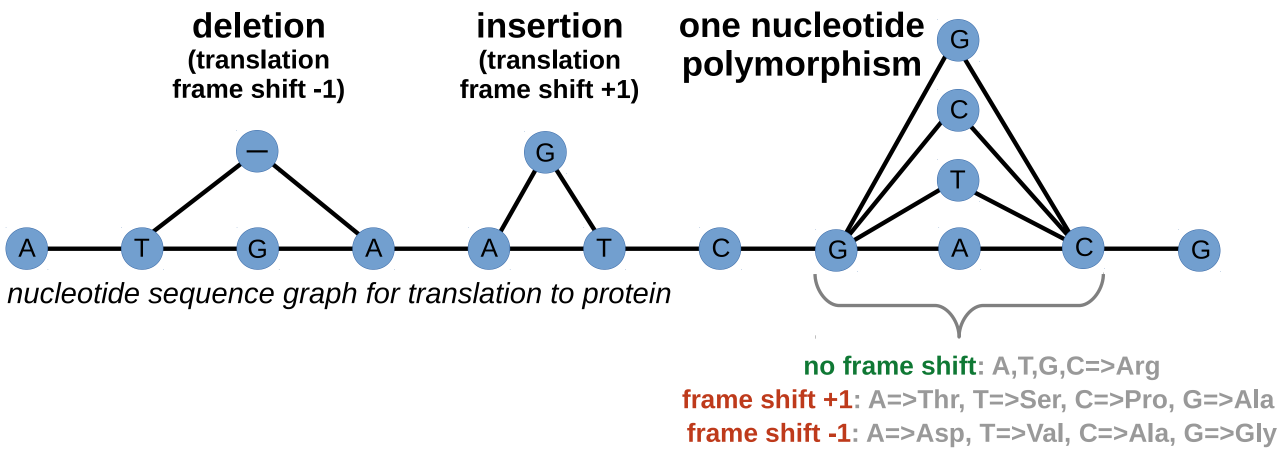
deletion
(translation
frame shift -1)
insertion
(translation
frame shift +1)
one nucleotide
polymorphism
A
T
G
A
A
T
C
G
A
C
G
–
G
G
C
T
nucleotide sequence graph for translation to protein
no frame shift: A,T,G,C=>Arg
frame shift +1: A=>Thr, T=>Ser, C=>Pro, G=>Ala
frame shift -1: A=>Asp, T=>Val, C=>Ala, G=>Gly

Fig. 2 The variation type (synonymous or non-synonymous) depends on frame shifts in upstream. PeptoVar considers full set of frame shift combinations in upstream to classify each polymorphism properly.

(C,T)T(A,T)

CT(A,T) => Leu (L)

TT(A,T) => Leu (L), Phe (F)

Fig. 3 Polymorphisms translation dependency in the polymorphic codon.

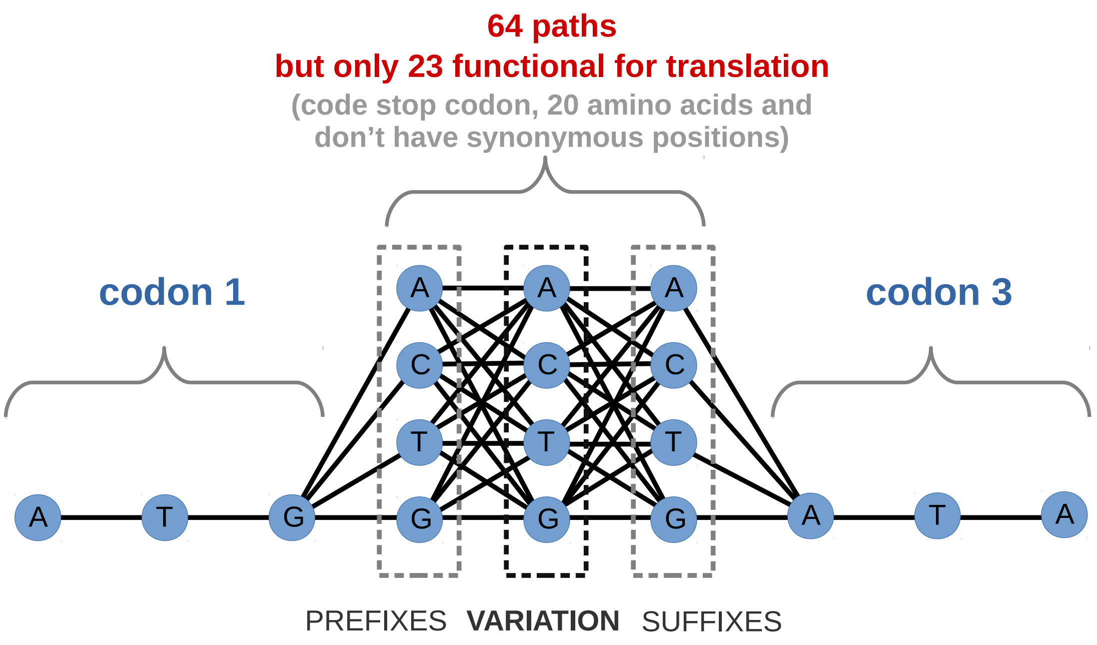
64 paths
but only 23 functional for translation
(code stop codon, 20 amino acids and don't have synonymous positions)
codon 1
codon 3
A T G
A C T G
A C T G
A C T G
A T A
PREFIXES
VARIATION
SUFFIXES

Fig. 4 Combinatorial explosion of translation to protein in case variations occur consecutively. For each variation PeptoVar takes prefixes and suffixes and determines functional paths including translation frames induced from upstream frame shifts.

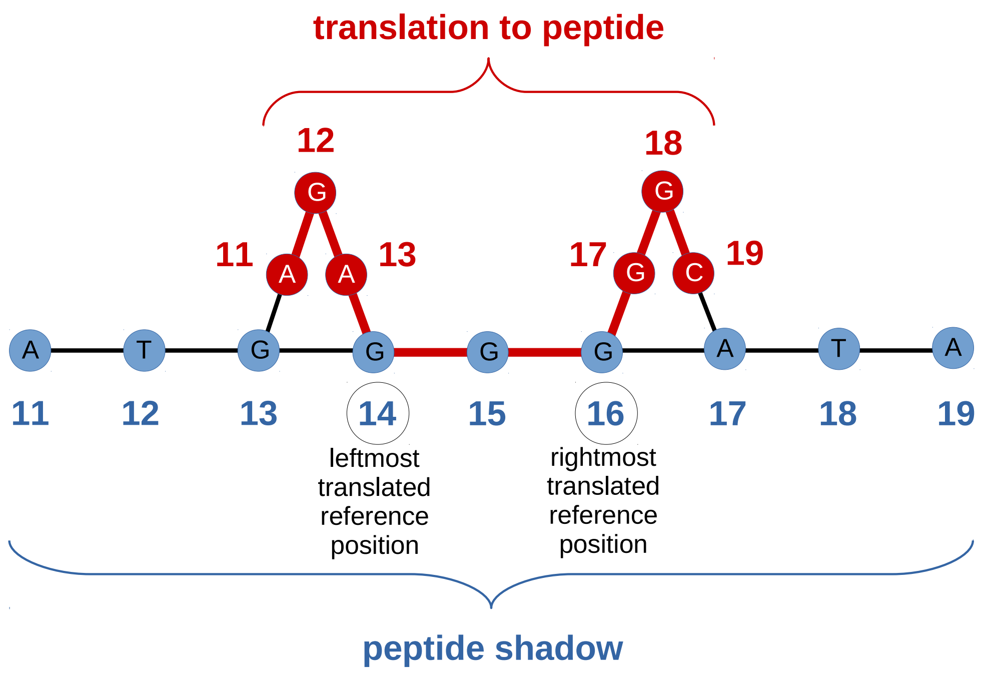

Fig. 5 The problem of peptide positioning on genome sequence. If peptide translation starts and ends on a genome insertion the peptide genome position can be presented as a "peptide shadow":

start = leftmost_translated_reference_position - number_of_translated_insertion_nucleotides;

end = rightmost_translated_reference_position + number_of_translated_insertion_nucleotides;

LICENSE