**FAQ of sver**

1. What is sver?

sver is a software aimed at discovering structural variants in genome through massive parallel sequencing.

1. What kind of structural variants sver can discover?

In principle, sver supports discovery of 5 classes of structural variants: Inversion, Deletion, Duplication, Insertion, Translocation.

In practice, as long as a structural variant event has at least 2 split reads supporting or 2 discordant paired-end reads supports, sver will discover this event.

1. What is the size limitation of structural variants sver can discover?

The minimum length of Inversion, Deletion and Duplication sver can discover is 50bp;

The minimum length of Insertion sver can discover is 10bp;

Translocation have not any length limits as they are across chromosomes;

For all structural variants, the maximum length is unlimited as long as there are proper reads alignment result supports.

1. How is sver implemented?

sver is implemented totally in C++, it does not need any third part scripts or software to assist its discovery of structural events. After compilation, it is a standalone binary executable program and can work on its own. During compilation, it only depends on the excellent gorgeous htslib.

1. What kind of sample can feed to sver?

From whole genome to little panel, whether it is paired end library or single ended library, sver can handle all of them as long as they have FASTQ of massive parallel sequencing reads. At the moment it does not support RNA samples.

1. What result formats does sver support?

sver can output bcf and tsv format results at the moment, all the records in bcf or tsv are sorted by chromosome and breakpoint coordinates naturally.

In the long run, I hope it will support vivid HTML format output with sequence and positions embedded.

1. Is sver fast? How much resources will it exhaust in one run?

In my test about 30 bam files ranging from 0.6G to 5G, sver will finish in 6 to 30 minutes. The memory peak has never reached above 3G. The threads are 8 by default. In principle the more threads it consumed, the faster it will be, as long as thread number are smaller than number of chromosomes containing structural variants.

1. How does sver work in real samples?

I have tested all the standard samples which have some predefined structural variants and recall them all precisely.

I have also tested some samples which Fusionmap didn’t discovery some events, all the missing events have been recalled by sver again except one event which has only one split read support and not any discordant pair read supports(In my opinion it is the problem of sequencing)

1. Has the result of sver been filtered under some limitations?

No, not too much, you can filter it against various limitations as you like, or you can propose these issues to me as I am planning to write a filter program to refine the results.

1. How to run sver?

Firstly, get your FASTQ and do some QC filter such as cut adapter and drop low quality reads.

Secondly, alignment you FASTQ with bwa to reference genome in `bwa mem` mode and keep the supplementary alignments as they are(in short, run bwa mem in its default argument setting)

Thirdly, sort alignment result bam by coordinates with samtools and mark duplicates with duplexer or picard tool (please do not use gencore as it will drop all the supplementary alignments) and sort the bam by coordinates once more.

Last, feed the sorted bam with dup marked to sver

1. How wo set sver arguments?

-b the input bam file

-g the reference genome used in previous alignment of FASTQ

-a annotation database file, which I have prepared it for you

-r bed region in which to discover structural variants. If provided, sver will only

find structural variants in these provided regions. If not provided, sver will find

structural in the whole genome.

-o output bcf file name

-t output tsv file name

-s structural variant types to discover, if not provided, all kinds of structural variants

will be discovered

-n number of threads use

1. What reference genome version does sver use?

Both hg19 and hg38 will work as long as the annotation database file is the same version.

1. How to prepare annotation database file?

Firstly, get refgene tsv file from UCSC;

Secondly, run a lite program written by myself to extract information from the tsv.

Thirdly, sort and compress with bgzip of extracted result.

Last, use tabix to index the result file for sver usage

1. What strategy does sver used?

Breifly:

Firstly, it finds all split reads and cluster them by breakpoint positions and structural variant type they support;

Secondly, it uses MSA to generate a consensus sequence of all reads supporting each structural variant event;

Thirdly, it uses global alignment the consensus sequence against the reference constructed from reference genome depending on the structural variant type and breakpoint positions to refine the breakpoints;

What’s more, it also finds all the discordant pairs and cluster them by breakpoint positions and structural variant type they support;

And then, it merges structural events discovered by split reads and discordant read pairs.

After that, it will compute the reference and variant reads count at each breakpoint to calculate AF value as well as genotype likely hood.

Finally, it will annotate all structural variants and output the result.

1. How to read the sver result?

For bcf file, use bcftools view command can open the bcf file. The bcf file is just the binary format of vcf which consumes much less space and more friendly for programers. The bcf file totally follows the standards of vcf formats version4.2, all the tags are explanained in the headers, if you do not understand them, just refer the explanations in headers.

For tsv file, I will explanain each column in the following paragraphs

**SVType**: structural variant type, they are abbreviations defined in BCF Formats4.2

**CatType**: catenation type, can be 5to5, 3to3, 5to3, 3to5

**Chr1**: the larger chromosome participating this structural variant

**Breakpoint1**: the breakpoint position on the larger chromosome

**Gene1**: the gene name the breakpoint position ‘Breakpoint1’ belongs to

**Chr2**: the little chromosome participating this structural variant

**Breakpoint2**: the breakpoint position on the little chromosome

**Gene2**: the gene name the breakpoint position ‘Breakpoint2’ belongs to

**SRSupports**: number of split reads supporting this structural variant

**DPSupports**: number of discordant pairs of reads supporting this structural variant

**RefCount**: number of read/pairs supporting reference type genotype

**AltCount**: number of read/pairs supporting this structural variant

**AltFreq**: structural variant frequency

**Strand1**: strands of all the transcripts spanning Breakpoint1

**Transcripts1**: transcripts and their functional parts spanning Breakpoint1

**Strand2**: strands of all the transcripts spanning Breakpoint2

**Transcripts2**: transcripts and their functional parts spanning Breakpoint2

**SVID**: SV ID, just a counting number for programming internal usage

PS: if a structural variant occurred on the same chr, Chr1 and Chr2 will be the same

1. What does 5to5, 3to3, 5to3 and 3to5 means?

For structural variant on the same chromosome.

5to5 stands for the leftmost breakpoint catenation manner of inversion.

3to3 stands for the rightmost breakpoint catenation manner of inversion.

5to3 stands for the breakpoint catenation manner of deletion.

3to5 stands for the breakpoint catenation manner of duplication.

NtoN stands for the breakpoint catenation manner of insertion.

For structural variant across chromosomes(translocations)

5to5 stands for 5’end of two chromosome catenated together.

3to3 stands for 3’ends of two chromosome catenated together.

5to3 stands for 5’end of larger chr catenated with 3’end of little chr

3to5 stands for 3’end of larger chr catenated with 5’ene of little chr

These slogans may seem daunting at the first time, while if you get used to them and think through, you will appreciate their meaning hopefully.

1. How to do the programming of all the strategies and algorithms in sver?

Eh…Here I will omit about 30 pages of PPTX and 5 thousand lines of C++ codes. If you’re interested in it, I might have a talk about this sometime later~, this is really the most exciting part anyway~

1. How can I get the source code of sver?

It’s hosted on GitHub: <https://github.com/vanNul/sver>, you can do whatever you want with it

**PS: if you find this software useful, please report any bugs you fond to me and I will kill the bugs to provide you with a much better user experience then~**