
Table of Contents

Introduction

About SDFA	1.1
Download and Install	1.2
API Document	1.3

Usage Document

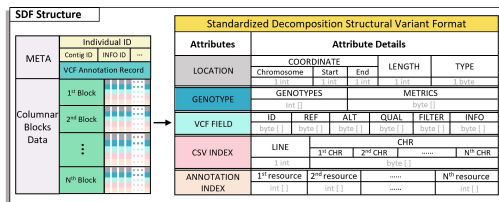
Build SDF Archives	2.1
Build SDF from VCF	2.1.1
Build SDF from other files	2.1.2
SDF Operation Toolkit	2.2
SDF GUI	2.2.1
Filter Mode	2.2.2
Sample Extraction	2.2.3
Concatenate Multiple SDF	2.2.4
SDFA Toolkit	2.3
Sample Merge	2.3.1
Functional Annotation	2.3.2
Numeric Gene Feature Annotation	2.3.3
SV-based GWAS	2.3.4

What is SDFA

SDFA is an efficient analysis tool designed for large-scale structural variation (SV) analysis. It is based on a new SV storage format and constructs a supporting toolset. Specifically, it first designs a standardized decomposition format (SDF) for SV, which efficiently represents, stores, and retrieves any type of SV data by decomposing SV. Based on the SDF file, SDFA designs or optimizes existing SV analysis algorithms considering the performance in large-scale samples.

What is SDF

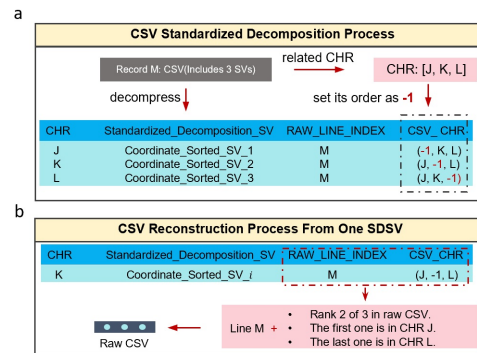
The full name of SDF is Standardized Decomposition Format (SDF). It is a file format for splitting, storing, and compressing SV data:



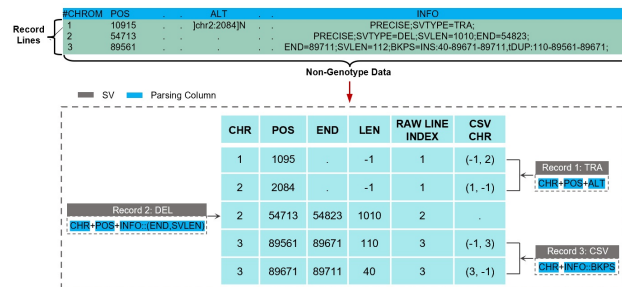
We provide a detailed explanation of the attributes in the above figure as follows:

Group	Field	Value Type	Description
LOCATION	coordinate	int[3]	The start and end positions of the chromosome where the current SV is located
LOCATION	length	int	The length of the current SV (for example, for an insertion variation, it is impossible to determine its length only relying on the <code>coordinate</code> field value)
LOCATION	type	int	Type of the current SV
GENOTYPE	genotypes	bytecode	The genotype of the current sample under this SV
GENOTYPE	metrics	bytecodeList	Quality metrics information of the current genotype
VCF Field	id	bytecode	The ID information of the current SV in the original VCF file
VCF Field	ref	bytecode	The REF information of the current SV in the original VCF file
VCF Field	alt	bytecode	The ALT information of the current SV in the original VCF file
VCF Field	qual	bytecode	The QUAL information of the current SV in the original VCF file
VCF Field	filter	bytecode	The FILTER information of the current SV in the original VCF file
VCF Field	info	bytecodeList	The INFO information of the current SV in the original VCF file
CSV INDEX	line	int	The line number of the current SV in the original VCF file
CSV INDEX	chr	int[N]	If the current SV is a complex SV, record the chromosomes where all the split SVs are located
ANNOTATION INDEX	indexes	int[N]	Record the intervals of lines related to the current SV and various annotations

In the above description, we emphasize the concepts of "splitting" and "assembling" because we split all SVs into multiple single intervals. Each single - interval SV after splitting is called a Standardized Decomposition SV (SDSV). The specific principles of splitting and reconstruction are as follows:



Based on the above principles, we provide an example of splitting from the original VCF file:



Why use SDFa

Compared with existing tools, SDFa has the following advantages:

- SDFa provides a systematic solution to fundamentally solve the problems of large-scale SV basic analysis.
- It can efficiently handle complex SV types, such as nested SVs, while other tools often fail to correctly parse these complex SVs.
- SDFa significantly outperforms existing tools in terms of speed and efficiency, especially on large - scale datasets.
- SDFa can collaborate with tools such as Plink to conduct SV-based GWAS research and explore SV at the population level.

Functions and Outstanding features of SDFa?

- Efficient SV data storage and retrieval: Achieved through the SDF format.
- Consistent and robust SV merging algorithm: Capable of handling large-scale sample data.
- Fast and memory-efficient SV annotation: Using the indexed sliding window algorithm.
- Novel and precise gene feature annotation: Using the Numerical Annotation of Gene Features (NAGF) method.
- Excellent performance: At least 17.64 times faster in SV merging speed and at least 120.93 times faster in annotation speed.
- Ability to parse and annotate complex SVs: The only tool that can correctly handle nested complex SVs.
- High scalability: Successfully processed 895,054 SVs from 150,119 individuals in the UK Biobank dataset, while other methods failed.
- Parallel processing capability: Can utilize multi-threading to improve processing speed.
- Flexible customization features: Such as user-defined filtering conditions and annotation resources.

Download and Install

SDFA is an application developed based on Oracle JDK 8. Our software can be used on any computer device that supports or is compatible with Oracle JDK 8. Users need to download and install [Oracle JDK](#) or [Open JDK](#) first. For Apple Silicon devices, [zulu JDK](#) can be used as an alternative.

Resource Type	Path
Software	https://github.com/Overinterested/SDFA/blob/master/SDFA.jar
Source Code	https://github.com/Overinterested/SDFA/tree/master/src
API Document	https://pmglab.top/SDFA/api
Example Files	1. VCF test files: 1100 VCF files 2. VCF pedigree files: http://data.schatz-lab.org/jasmine/HG002Trio/UnmergedVCFs/ 3. Annotation resource files: https://zenodo.org/records/13293672

Install Software

```
# install SDFA software
wget https://github.com/Overinterested/SDFA/blob/master/SDFA.jar

# run SDFA
java -jar SDFA.jar
```

Update Log

[!UPDATE|label:2024/05/20]

Release the first version of SDFA, version number 1.0, Github repository address: TODO

Build SDF from VCF

SDFA provides the conversion from VCF (Variant Calling Format) files to SDF (Standardized Decomposition Format) files for standard diploid species, aiming to achieve efficient storage of SV data, rapid access and location of SV and its related attributes, and compression of VCF files.

Quick Start

In the command line, use the following command to build an SDF archive for the genomic VCF file:

```
java -jar sdfa.jar vcf2sdf [options]
```

Of course, the above command line supports converting GZ and BGZ compressed VCF files into SDF files. Here are a few simple examples:

- Build SDF from single VCF file

[!NOTE|label:Example 1]

Build an archive using the example file `HG01258_HiFi_aligned_GRCh38_winnowmap.sniffles.vcf` :

```
java -jar sdfa.jar vcf2sdf -f ./HG01258_HiFi_aligned_GRCh38_winnowmap.sniffles.vcf -o ./
```

- Build SDF from the fold

[!NOTE|label:Example 2]

Build an archive using a folder as input:

```
java -jar sdfa.jar vcf2sdf -d ./data -o ./
```

- Build SDF by specifying the Calling Type of VCF files

[!NOTE|label:Example 3]

Unlike the VCF files of SNPs, the SV VCF formats of different calling tools vary (mainly reflected in the `INFO` field), and these differences can affect the extraction of SV coordinate positions. Therefore, we have implemented the extraction of SV VCFs from 13 mainstream SV calling tools:

CuteSV2、CuteSV、Debreak、Delly、NanoSV、Nanovar、Pbsv、Picky、Sniffles2、Sniffle、Svim、Svision、Ukbb

In SDFA, the type of the selected Calling tool can be specified through `--calling-type` or `-ct` . It is worth noting that, by default or when an unknown `ct` is specified, the parsing will be carried out according to the [standard SV VCF4.3](#).

```
java -jar sdfa.jar vcf2sdf -ct cutesv -d ./data -o ./
```

API Document

The API tool for converting VCF files to SDF files is SDSVManager. The usage examples are as follows:

```
// Convert all VCF files and their GZ and BGZ compressed files in a certain folder
int thread = 4;
SDSVManager.of("inputDir")
    .setOutput("outputDir")
    .setCallingType("cuteSV")
    .run(thread);
```


Build SDF From other file type

Currently, the main recording and storage format of SV is the VCF file. Therefore, in SDFA, there is no command - line tool to directly build an SDF archive from other file formats. However, based on the extensive compatibility of SV files and the SV analysis tools developed subsequently, SDFA provides an API to build an SDF archive.

SDFWriter build SDF Archives

The `SDFWriter` is a class specifically designed to directly build an SDF by constructing `SDFWriterRecord`. In essence, it is a process of obtaining a record, setting the record, writing the record, resetting the record and ending the record. Therefore, users need to consider how to convert an input file format into an `SDFWriterRecord` record.

Now, let's give an example of the use of `SDFWriter`:

[!NOTE|label:Example 1]

The essence of `SDFWriterRecord` is an SV record. Therefore, we need to construct an SV record and manually write it into a file.

```
public static void main(String[] args) throws IOException, InterruptedException {
    String[] names = new String[1000];
    for (int i = 0; i < 1000; i++) {
        names[i] = String.valueOf(i);
    }
    SDFWriter writer1 = SDFWriter.SDFWriterBuild.of(new File("/Users/wenjiepeng/Desktop/tmp/yg/1.sdf"))
        .addFormat("GT")
        .addFormat("AD")
        .addInfoKeys("PRECISE", "READS_SUPPORT")
        .addIndividuals(names)
        .build();
    SDFWriter.SDFWriterRecord item = writer1.getTemplateSV();
    item.setInfo("PRECISE", "true")
        .setInfo("READS_SUPPORT", "3")
        .setAlt(new Bytes("ACGAGGGCCCAA"))
        .setChrName("chr1")
        .setType(SVTypeSign.getByNames("DEL"))
        .setID(new Bytes("ID_0"))
        .setRef(new Bytes("ACGAGGGCCCAA"))
        .setPos(1000)
        .setEnd(2000)
        .setLength(1000)
        .setQuality(new Bytes("."))
        .setFilter(new Bytes("PASS"));
    for (int i = 0; i < 1000; i++) {
        item.addInitFormatAttrs(i, "1/1;3,2");
    }
    writer1.write(item);
    writer1.write(item.setPos(999));
    writer1.write(item.setPos(1001));
    writer1.close();
}
```

The above example is to construct an SV and write it. Therefore, for an input file in any format, what the user needs to construct is the process of converting Line → Record.

API Document

Specifically, users can check the SDF archive built by the `SDFWriter` class.

Graphical User Interface For SDF

To facilitate the viewing of SDF files, SDFA has designed a graphical interface that makes full use of the block and column features of SDF files and has the following characteristics:

- Local Scan: Only scan the content being viewed, making use of block and column features.
- Page Navigation: Support fast navigation with low memory usage.
- Custom Display: Allow setting the viewing method when the user has a custom encoding method.
- Complete Information: Display the source information of VCF header information, SV information, and file size.

CCFViewer: /Users/wenjiepeng/Desktop/SDFA_4.0/test/vcf2sdf/simple_with_no_type/sdf/HG01258_HiFi_aligned_GRCh38_winnomap.sniffles.vcf.sdf

Records		Metadata	Summary Information											CSV_LOCATI...		ANNOTATION_IN...
INDEX	INDEX	LOCATION		type	GENOTYPE		VCF_FIELD						info	chr	indexes	
		coordinate	length		genotype	id	ref	alt	qual	filter						
0		1:54713-54713	37	INS	0 1=1	Sniffles2.IN...	N	TTTTTTTCT...	58	PASS	PRECISE=;S...					
1		1:66265-66265	250	INS	0 1=1	Sniffles2.IN...	N	TATATTATA...	58	PASS	PRECISE=;S...					
2		1:67898-68334	436	DEL	0 1=1	Sniffles2.D...	AGTTAAAG...	N	58	PASS	PRECISE=;S...					
3		1:90259-90259	118	INS	1 1=1	Sniffles2.IN...	N	GTCCCTCT...	60	PASS	PRECISE=;S...					
4		1:120561	.	BND	0 1=1	Sniffles2.B...	N	Njchr1:1205...	56	PASS	PRECISE=;S...	[-1, 0]				
5		1:121070-121070	165	INS	0 0=1	Sniffles2.IN...	N	ATAATATTA...	60	GT	PRECISE=;S...					
6		1:136646-136646	48	INS	0 1=1	Sniffles2.IN...	N	GGCTCGG...	60	PASS	PRECISE=;S...					
7		1:136935-136935	293	INS	0 1=1	Sniffles2.IN...	N	CAAGGGG...	60	PASS	PRECISE=;S...					
8		1:136972-136972	196	INS	0 0=1	Sniffles2.IN...	N	GTGGGAG...	60	GT	IMPRECISE=...					
9		1:180897-180965	68	DEL	0 1=1	Sniffles2.D...	CTAACCCT...	N	54	PASS	PRECISE=;S...					
10		1:180995-180995	405	INS	0 1=1	Sniffles2.IN...	N	GCACATGA...	60	PASS	IMPRECISE=...					
11		1:181157-181282	125	DEL	0 1=1	Sniffles2.D...	GGCGCAG...	N	57	PASS	PRECISE=;S...					
12		1:181263-181263	82	INS	0 1=1	Sniffles2.IN...	N	CGCCGGC...	60	PASS	IMPRECISE=...					
13		1:181360-181482	122	DEL	0 1=1	Sniffles2.D...	GGGAGGA...	N	57	PASS	PRECISE=;S...					
14		1:381391-381391	204	INS	1 1=1	Sniffles2.IN...	N	GGGTTCTC...	60	PASS	PRECISE=;S...					
15		1:596698-596798	100	DEL	0 1=1	Sniffles2.D...	CATTGATG...	N	59	PASS	PRECISE=;S...					
16		1:597795-597795	471	INS	0 1=1	Sniffles2.IN...	N	AACGGCCT...	58	PASS	IMPRECISE=...					
17		1:597909-598002	93	DEL	0 0=1	Sniffles2.D...	ACGCGGGT...	N	60	GT	PRECISE=;S...					
18		1:598524-598617	93	DEL	0 1=1	Sniffles2.D...	GGGTGCC...	N	60	PASS	PRECISE=;S...					
19		1:600765-600765	237	INS	0 1=1	Sniffles2.IN...	N	GAGTTCCT...	59	PASS	PRECISE=;S...					
20		1:601109-601109	131	INS	0 1=1	Sniffles2.IN...	N	GACCACCT...	60	PASS	PRECISE=;S...					

←

→

Skip To

Filter Mode

Compared with single - nucleotide polymorphism (SNP), the filtering and screening of SV often lack a unified standard, and the screening attributes are more diverse (possibly including INFO, QUAL, GT fields, etc.).

To perform more comprehensive SV filtering, Sdfa has a rich set of built - in filtering functions. Specifically, it mainly covers two aspects of filtering:

- `Genotype level` : Set quality control information for genotype filtering - set those that do not meet the conditions as `./.`
- `SV level` : Set a custom function to filter multiple VCF fields such as `CHR` and `ID` of the current SV record

The specific instructions are

```
java -jar sdfa.jar filter -d [input_dir] -o [output_dir] [options]
```

Genotype Filter

Use the following instructions for filtering:

```
--filter-gty <format_attr> <function>
```

The above command line filters the specified `format_attr` , and `function` is an expression that returns a `boolean` value.

To facilitate users to directly operate on the values of the quality control attributes of each genotype, we directly use `value` in the function to represent the value of a single genotype. Now, let's take an example:

[!NOTE|label:Example 1]

We filter the genotypes of all SDF files in the `./data` folder. The filtering quality control attribute is `GQ` , and the filtering condition is `(int)value>20` . The output is saved to the `./folder` :

java -jar sdfa.jar filter -d ./data -o ./ --filter-gty GQ (int)value>20

If the SDF file is a multi-sample file, perform the above-mentioned conditional filtering on the genotypes of each sample in sequence.

To simplify the conversion logic and reduce the storage space of `metric` field data, we encapsulate common quality control attributes into specific types to facilitate users to operate directly. The details are as follows:

Storage Class	Storage Type	Value Type	Quality control attributes	Example	Example Explanation
SingleIntValueBox	int List	Single int value	DP , DR , DV GQ , MD , PP	--filter-gty DP (int)values>10	Set the genotype to./ when the DP attribute value is <= 10.
TwoIntValueBox	int List	IntList instance composed of 2 int values	AD , RA	--filter-gty AD ((IntList)values).get(0)>10	Set the genotype to./ when the first int value of the AD attribute value is <= 10.

ThreeIntValueBox	<code>int</code> List	IntList instance composed of 3 <code>int</code> values	PL	<code>--filter-qty PL ((IntList)values).get(0)>10</code>	Set the genotype to./ when the first int value of the PL attribute value is <= 10.
SingleStringValueBox	<code>string</code> List	Single string value	FT others	<code>--filter-qty FT (string)values.equals(\"TRUE\")</code>	Set the genotype with the FT attribute not being TR

SV Filter

Similar to the instructions for genotypes, the instructions for SV filtering are as follows:

```
--filter-sv <sv_attr> <function>
```

The above command line filters the specified `sv_attr` , and `function` is a JAVA expression that returns a `boolean` value. Now, let's take an example:

[!NOTE|label:Example 2]

We filter for `IMPRECISE` in the INFO field. When this field exists, we filter the SV:

```
java -jar sdfa.jar -d ./data -o ./ --filter-sv IMPRECISE value!=null
```

In the above example, the `sv_attr` is `IMPRECISE`. In the SDFa design, there is a certain order for the positioning and acquisition of `sv_attr` . The following is the comprehensive filtering order:

INDEX	Match Field	Value Type	Value Description
1	CHR	<code>string</code>	Obtain the chromosome name of SV
2	ID	<code>Bytes</code>	Obtain the ID value of SV
3	REF	<code>Bytes</code>	Obtain the REF value of SV
4	ALT	<code>Bytes</code>	Obtain the ALT value of SV
5	QUAL	<code>Bytes</code>	Obtain the QUAL value of SV
6	FILTER	<code>Bytes</code>	Obtain the FILTER value of SV
7	INFO	<code>List<Bytes></code>	Obtain all INFO field values of SV
8	FORMAT	<code>List<Bytes></code>	Obtain the quality control values of all genotypes of SV
9	GT	<code>CacheGenotypes</code>	Obtain the genotypes of all samples under SV
10	LEN	<code>int</code>	Obtain the LEN value of SV
11	SDF_FIELD_NAME	<code>object</code>	Obtain the value of SV in <code>SDF_FIELD_NAME</code> (for details, please refer to the SDF structure)
12	INFO_ATTR	<code>Bytes</code>	Obtain the value of SV in <code>INFO</code> where the KEY is <code>INFO_ATTR</code>
13	FORMAT_ATTR	<code>Bytes</code>	Obtain the value of SV in <code>FORMAT</code> where the KEY is <code>FORMAT_ATTR</code>
14	UNKNOWN	ERROR	.

For detailed value retrieval, you can refer to the `getV` function of `SDFRecordWrapper` .

[!TIP|label:Filter Order]

It is worth noting that in the actual code execution, the SV Level filtering will be carried out first, and then the Genotype Level filtering.

Sample Extraction

In large datasets, it is often necessary to extract fixed samples for downstream analysis. For example, the UKB's Whole genome GraphTyper SV data [interim 150k release] is a merged 150k dataset. When we need case - control samples for certain diseases, we need to perform extraction operations.

After the VCF file is converted to an SDF file, use the following command to extract the SDF:

```
java -jar sdfa.jar extract [options]
```

Here we use the PED file as the input file to extract the required samples from the population SDF file. At the same time, SDFA provides some basic SV screening functions for the SV after sample extraction.

[!NOTE|label:Example 1]

We extract all SDF files in the input folder:

```
java -jar sdfa.jar extract -d ./data -ped ./ped.ped -o ./
```

Program parameters

```
Grammar: extract -d input_dir -o output_dir -ped ped_path
Java-API: edu.sysu.pmglab.sdfa.toolkit.SDFExtract
About: Extract samples from PED in multiple SDF files
Parameters:
  *--output, -o          Set the output folder.
                        Format: -o <dir>
  *--dir, -d             Set the input folder.
                        Format: -d <dir>
  *--ped-file, -ped      Set the PED file.
                        Format: ped <file>
  --thread, -t           Set the thread numbers.
  --max-maf              Set the maximum proportion of genotypes in the extracted samples
  --min-maf              Set the minimum proportion of genotypes in the extracted samples
```

API Document

The API tool for extracting SDF files is SDFExtract, and the usage examples are as follows:

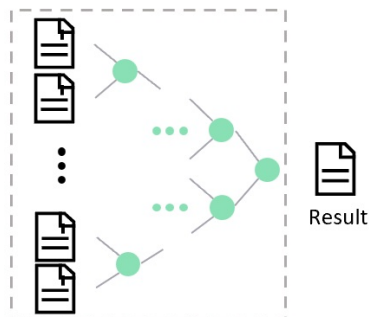
```
SDFExtract.of(file.toString(),
    sdfExtractProgram.pedFile,
    FileUtils.getSubFile(sdfExtractProgram.outputSDFDir, file.getName())
)
    .setMaxMAF(sdfExtractProgram.maxMaf)
    .setMinMAF(sdfExtractProgram.minMaf)
    .submit();
```

Concatenate Multiple SDF files

Concatenation means adding new variant sites line by line. The most common scenario is to reassemble the variant sites of the same group of subjects that are scattered in different files (stored according to chromosomes, the number of variant sites, and file size) back into a single file (for example: the UKB's Whole genome GraphTyper SV data [interim 150k release] data). Use the following command to concatenate SDF files:

```
java -jar sdfa.jar concat [options]
```

When merging multiple files, the sample names of all files will be checked, and a two-way merge sort will be used for merging in each thread. During the merging process, the coordinates are ensured to be in order and the META information is continuously updated.



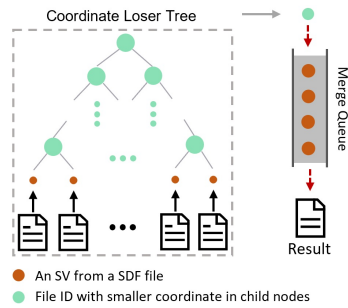
[!NOTE|label:Example 1]

The following is the merging of 3 single - sample files (assuming here that the sample names of the 3 files are the same):

```
java -jar sdfa.jar concat -d ./data -o ./ -t 4
```

Sample Merge

In order to further conduct larger-scale SV research, SDFA has developed a cohort-wide merging algorithm for SV sample merging at the population scale level. This algorithm is based on the SDF file and performs k-way merging on the basis of considering all SV data in an orderly manner:



Currently, SDFA uses position to perform SV merging among samples, as follows:

SDFA maintains an ordered list of SVs of the same type and location, and adds SVs of the same type in ascending order of the minimum position. When a newly added SV (marked as SV_{new}) does not meet the merging conditions, all SVs in the current list will be popped and merged into one SV. The default merging conditions are as follows:

$$\begin{aligned} |Pos_{first} - Pos_{new}| &< Threshold \\ |End_{first} - End_{new}| &< Threshold \end{aligned}$$

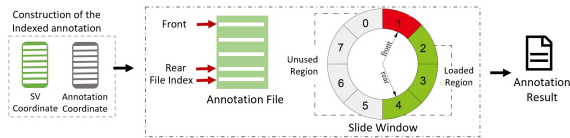
[!NOTE|label:Example 1]

SDFA merges the VCF files (including compressed files and SDF files) in the specified folder and outputs the merged results to the specified folder. The command line is as follows:

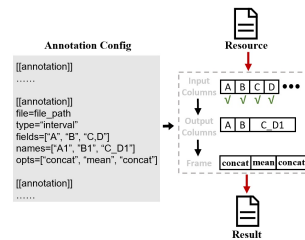
```
java -jar sdfa.jar -d ./data -o ./
```

Functional Annotation

To further locate susceptible SVs and explore their biological mechanisms, it is often necessary to perform functional annotation on SVs. In order to quickly conduct functional annotation for multiple samples and multiple resources, SDFA has designed the index sliding window algorithm shown in the figure below to accelerate the annotation process.



At the same time, in order to perform customized annotation for multiple resources, SDFA has designed the post-annotate annotation method. The output file can be customized through the configuration file shown in the figure below:



Configuration File

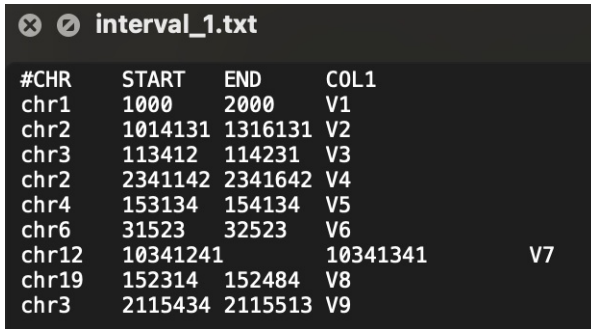
SDFA draws on the post annotation concept of the [Vcfanno](#) tool and defines a configuration file for configuring annotation resources and output results. The specific explanation is as follows:

Parameters	Parameter Explanation	Required or not (* represents required, . represents optional)
<code>[[annotation]]</code>	The start identifier for a new annotation resource	*
<code>file</code>	The full path of the annotation resource file	*
<code>type</code>	Type of annotation resource (3 types are supported: <code>gene</code> , <code>interval</code> , and <code>svdatabase</code>)	*
<code>names</code>	Column names of the output results	.
<code>fields</code>	Column names of the input file	.
<code>opts</code>	Functions corresponding to the output columns	Needs to match the size of names

Here we explain the above parameters through an example:

```
[!NOTE|label:Example 1]
```

Next, we will operate on the `interval_1.txt` file. This file is of the interval type, and its general content is:



#CHR	START	END	COL1
chr1	1000	2000	V1
chr2	1014131	1316131	V2
chr3	113412	114231	V3
chr2	2341142	2341642	V4
chr4	153134	154134	V5
chr6	31523	32523	V6
chr12	10341241	10341341	V7
chr19	152314	152484	V8
chr3	2115434	2115513	V9

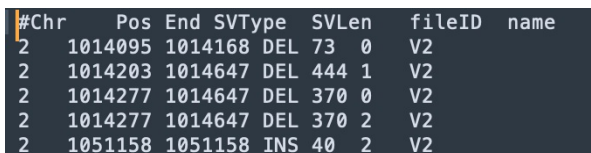
Elements between single lines are separated by `\t`. Now we want to obtain the `COL1` column related to SV, so we set `fields=["COL1"]` here. At the same time, we want the output column to be named `name`, so we set `names=["name"]`. The final configuration file is as follows:

```
[[annotation]]
file=/Users/wenjiepeng/Desktop/SDFA_4.0/test/annotation/data/interval_1.txt
type=interval
names=["name"]
fields=["COL1"]
opts=["concat"]
```

After completing the above configuration file, we perform annotation through the following command:

```
java -jar sdfa.jar annotate \
--config /Users/wenjiepeng/Desktop/SDFA_4.0/test/annotation/data/config.txt \
-t 4 -d /Users/wenjiepeng/Desktop/SDFA_4.0/test/vcf \
-o /Users/wenjiepeng/Desktop/SDFA_4.0/test/annotation/res
```

The results after annotation are as follows:



#Chr	Pos	End	SVType	SVLen	fileID	name
2	1014095	1014168	DEL	73 0	V2	
2	1014203	1014647	DEL	444 1	V2	
2	1014277	1014647	DEL	370 0	V2	
2	1014277	1014647	DEL	370 2	V2	
2	1051158	1051158	INS	40 2	V2	

Among them, the `fileID` corresponds to different SV files in this folder (`/Users/wenjiepeng/Desktop/SDFA_4.0/test/vcf`).

Annotation Resources

SDFA supports integrating external databases for SV annotation, provided that the external data files meet the following basic format requirements:

- The tab character (`\t`) must be used as the delimiter.
- The annotation file must contain a header line, and the column names should start with the number sign (`#`) (lines starting with `##` will be ignored).

In addition, when SDFA introduces interval annotation files and SV database files to annotate existing SV data, the following specific conditions must be met:

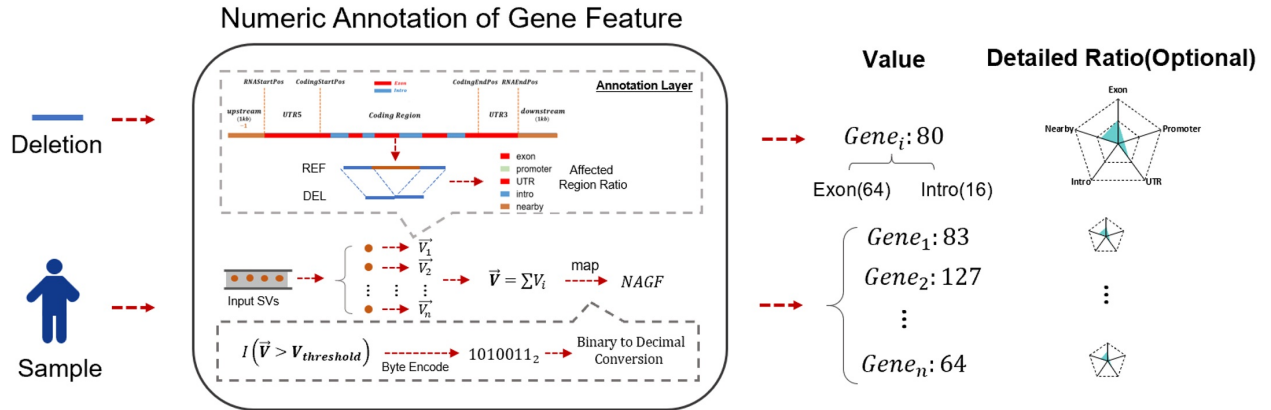
- For interval annotation files, the following requirements must be met:

Column	Name	Type	Example
1	Chromosome	String	chr1
2	Start Position	Integer	1000
3	End Position	Integer	3000
4	[Feature 1 Name]	String	V1

- For SV database files, the following requirements must be met:

Column	Name	Type	Example
1	Chromosome	String	chr1
2	Start Position	Integer	1000
3	End Position	Integer	3000
4	SV Length	Integer	2000
5	SV Type	String	DEL
6	[Feature 1 Name]	String	V1

Numeric Gene Feature Annotation



Numeric Gene Feature Annotation (NAGF) is a new annotation result proposed by SDFA for SV. NAGF uses 8 bits to represent the affected gene feature regions (such as exons, introns), and 5 bytes to represent the proportion of the affected region of each gene feature within a single SV in the gene. The 8 bits form the feature bits, and the 5 bytes form the coverage range. The following is the detailed information about NAGF:

- **Feature Bits:** This byte makes full use of 8 bits to represent different functional regions of a gene. The first bit indicates whether the gene is a protein - coding gene. It represents the affected exons, promoters, UTRs, introns, and nearby regions respectively, where 0 indicates not affected and 1 indicates affected. It also represents complete coverage through copy - number variation (CNV) and inversion. Note that similar ideas can use more bits to represent more subtle features, such as 5'UTR and 3'UTR.
- **Number of Coverage Bytes:** These five bytes in this study represent the percentage of SV - affected areas (ranging from 0 to 100) in five gene - feature regions: exons, promoters, UTRs, introns, and nearby regions.

The final result is as follows:

```
GENE_NAME:Value:[xxxx, xxx, xx, xx, xxx, x]
```

```
[!NOTE|label:Example 1]
```

Use NAGF of SDFA to annotate the example folder and output it to the `tmp` folder in the user's home directory:

```
java -jar /Users/wenjiepeng/projects/sdfa_latest/SDFA.jar ngf \
-dir /Users/wenjiepeng/Desktop/tmp/sdfa_test/sdf_builder/vcf2sdf \
-f /Users/wenjiepeng/projects/sdfa_latest/resource/hg38_refGene.ccf \
-o /Users/wenjiepeng/Desktop/tmp/sdfa_test/sdf-toolkit/sdfa-nagf
```

SV-based GWAS

SV-based GWA studies have emerged. Since PLINK integrates many practical statistical testing methods, SDFA provides an SV-based GWA workflow for further exploring vulnerable SVs.

Taking the UKBB database as an example, when the sample size is massive, considering the huge amount of data, the overall SV data is split into multiple VCF files, with each file storing a part of the entire SV data. At the same time, since PLINK has integrated many GWA-related statistical analysis tools, SDFA provides the SDF2Plink tool to convert SDF files into Plink input files for subsequent SV-based GWA analysis.

SDFA has established a VCF \Rightarrow SDF \Rightarrow Plink pattern, which mainly consists of 4 processes:

- VCF2SDF: Convert VCF to SDF files and perform operations such as SV filtering and information extraction.
- SDFConcat: Integrate multiple SDF files into one SDF file.
- SDFExtract: Extract partial sample information from the integrated SDF file.
- SDF2Plink: Convert the extracted samples into Plink file format, namely `.fam`, `.bed` and `.bim` files.

[!NOTE|label:Example 1]

Here we take an example. First, integrate multiple SDF files in `./test/resource/gwas` folder. Then, extract the samples from `./test/resource/gwas/sample.ped` file. Finally, convert them into PLINK files. And use the PLINK files for analysis.

```
java -jar ./SDFA.jar gwas \  
-dir ./test/resource/gwas \  
-o ./test/resource/gwas/output \  
--ped-file ./test/resource/gwas/sample.ped \  
--concat \  
-t 4  
  
# plink  
## 1. filter  
./test/resource/gwas/plink2 --bfile ./test/resource/gwas/output/ \  
--geno 0.2 \  
--mind 0.8 \  
--hwe 1e-6 \  
--maf 0.05 \  
--make-bed \  
--out ./test/resource/gwas/output/geno_0.2_mind_0.8_maf_0.05  
## 2. association  
./test/resource/gwas/plink2 --bfile ./test/resource/gwas/output/geno_0.2_mind_0.8_maf_0.05 \  
-adjust \  
--glm \  
allow-no-covars \  
--out ./test/resource/gwas/output/geno_0.2_maf_0.05_res
```