目錄

项目简介

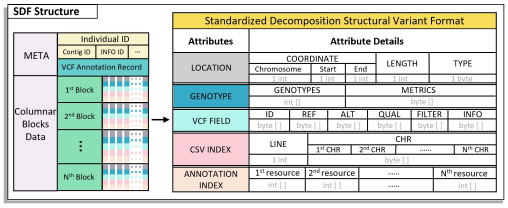
使用手册

SDFA是什么

SDFA是为大规模结构变异（Structural Variation， SV）分析设计的高效分析工具，它基于一种新的SV存储格式并构建了配套的工具集。具体而言，它首先设计了一种SV的标准化分解格式(Standardized Decomposition Format, SDF)，通过分解SV来高效地表征、存储和检索任意类型的SV数据。基于SDF文件，SDFA在考虑大规模样本下的性能下对已有的SV分析算法进行设计或优化。

SDF是什么

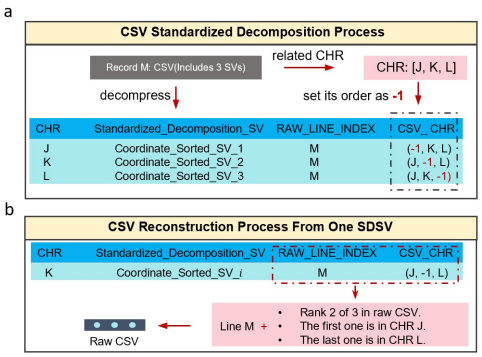
SDF的全称是标准拆分格式 (Standardized Decomposition Format, SDF)，是一种拆分、存储、压缩SV数据的文件格式:



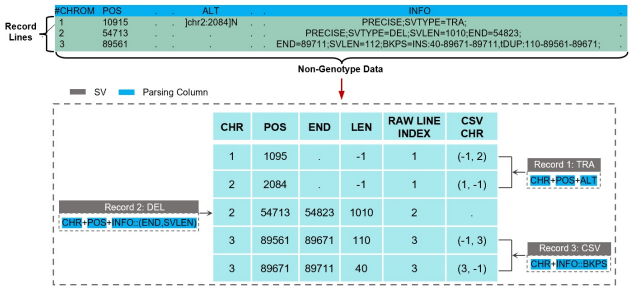
我们对上图的属性进行详细的解释：

字段组	字段	值类型	描述
`LOCATION`	`coordinate`	int[3]	当前结构变异（SV）所在染色体的起始和终止位置
`LOCATION`	`length`	int	当前结构变异（SV）的长度 (PS：对于插入变异，仅依靠坐标字段值无法确定其长度)
`LOCATION`	`type`	int	当前SV类型
`GENOTYPE`	`genotypes`	bytecode	该SV的样本基因型
`GENOTYPE`	`metrics`	bytecodeList	该SV所有基因型的质控信息
`VCF Field`	`id`	bytecode	原始 VCF 文件中当前 SV 的 ID 信息
`VCF Field`	`ref`	bytecode	原始 VCF 文件中当前 SV 的 REF 信息
`VCF Field`	`alt`	bytecode	原始 VCF 文件中当前 SV 的 ALT 信息
`VCF Field`	`qual`	bytecode	原始 VCF 文件中当前 SV 的 QUAL 信息
`VCF Field`	`filter`	bytecode	原始 VCF 文件中当前 SV 的 FILTER 信息
`VCF Field`	`info`	bytecodeList	原始 VCF 文件中当前 SV 的 INFO 信息
`CSV INDEX`	`line`	int	原始 VCF 文件中当前结构变异（SV）的行号
`CSV INDEX`	`chr`	int[N]	如果当前的结构变异（SV）是一个复杂的结构变异，记录所有拆分后的结构变异所在的染色体
`ANNOTATION INDEX`	`indexes`	int[N]	记录与当前结构变异（SV）及各种注释资源相关的行区间

在上面的描述中我们强调”拆分“和”拼装“的概念，因为我们将所有SV拆分为多个单区间，拆分后的每个单区间SV我们称之为标准拆分SV(Standardized Decomposition SV, SDSV)。具体的拆分和重构原理如下：



基于上述原理我们给出了一个示例，从原始的VCF文件中进行拆分：



为什么要用**SDFA**

相较于现有工具，SDFA具有如下优势：

- SDFA提供了一个从根本上解决大规模SV基础分析问题的系统性方案。
- 它能够高效地处理复杂的SV类型,如嵌套SV,而其他工具往往无法正确解析这些复杂SV。
- SDFA在速度和效率方面显著优于现有工具,特别是在大规模数据集上。
- SDFA可以联合Plink等工具进行SV- based GWAS研究，进行群体层面SV的挖掘。

SDFA有哪些功能和突出特点

- 高效的SV数据存储和检索:通过SDF格式实现。
- 一致且稳健的SV合并算法:能处理大规模样本数据。
- 快速且内存高效的SV注释:使用索引滑动窗口算法。
- 新颖精确的基因特征注释:使用数值化基因特征注释(NAGF)方法。
- 卓越的性能:在SV合并速度上至少快17.64倍,在注释速度上至少快120.93倍。
- 能够解析和注释复杂SV:是唯一能正确处理嵌套复杂SV的工具。
- 可扩展性强:成功处理了英国生物银行数据集中150,119个个体的895,054个SV,而其他方法失败。
- 并行处理能力:可以利用多线程提高处理速度。
- 灵活的自定义功能:如用户定义的过滤条件和注释资源。

下载与安装

SDFA 是基于 Oracle JDK 8 开发的应用程序，任何兼容支持或兼容 Oracle JDK 8 的计算机设备都可以使用我们的软件。用户需要先下载安装 Oracle JDK 或 Open JDK。Apple Silicon 设备可以使用 zulu JDK 作为替代。

资源类型	路径
软件包	https://github.com/Overinterested/SDFA/blob/master/SDFA.jar
源代码	https://github.com/Overinterested/SDFA/tree/master/src
API 文档	https://pmglab.top/SDFA/api
示例文件	**1. VCF**测试文件：1100 VCF files **2. VCF**家系文件：http://data.schatz-lab.org/jasmine/HG002Trio/UnmergedVCFs/ **3.** 注释资源文件： https://zenodo.org/records/13293672

下载软件包

```
# 下载 sdfa 软件包
wget https://github.com/Overinterested/SDFA/blob/master/SDFA.jar

# 运行 sdfa 软件包
java -jar SDFA.jar
```

更新日志

> [!UPDATE|label:2024/5/14]
>
> - 发布 SDFA 的第2个版本，版本号 2.0，Github 仓库地址：TODO

从 **VCF** 构建 **SDF** 存档

SDFA为标准的二倍体物种提供了从 VCF (Variant Calling Format)文件到 SDF(Standardized Decomposition Format) 文件的转化，以实现 SV 数据的高效存储、SV 及其相关属性的快速访问、定位和VCF 文件的压缩等目的。

快速入门

在命令行中，使用如下指令为基因组 VCF 文件构建 SDF 存档：

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

当然上述命令行支持把VCF的GZ、BGZ压缩文件转为SDF文件，我们提供几个简单的例子：

- 指定文件作为输入

> [!NOTE|label:Example 1]
>
> 使用示例文件 `HG01258_HiFi_aligned_GRCh38_winnowmap.sniffles.vcf` 构建存档：
>
> ```
> java -jar sdfa.jar vcf2sdf -f ./HG01258_HiFi_aligned_GRCh38_winnowmap.sniffles.vcf -o ./
> ```

- 指定文件夹作为输入

> [!NOTE|label:Example 2]
>
> 使用文件夹作为输入构建存档：
>
> ```
> java -jar sdfa.jar vcf2sdf -d ./data -o ./
> ```

- 指定输入数据的Calling类型

> [!NOTE|label:Example 3]
>
> 不同于SNP的VCF文件，不同Calling工具的SV VCF格式存在差异(主要体现在 `INFO` 字段)，并且这些差异会影响对SV坐标位置的提取。因此我们实现了对13种主流的SV Calling工具的SV VCF提取:
>
> CuteSV2、CuteSV、Debreak、Delly、NanoSV、Nanovar、Pbsv、Picky、Sniffles2、Sniffle、Svim、Svision、Ukbb
>
> 可以通过 `--calling-type` 或者 `-ct` 指定所选择的Calling工具类型。值得注意的是，默认情况下或者指定未知 `ct` 情况下会按照标准SV VCF4.3进行解析。
>
> ```
> java -jar sdfa.jar vcf2sdf -ct cutesv -d ./data -o ./
> ```

API工具

将 VCF 文件转换为 SDF 文件的 API 工具是 SDSVManager，使用示例如下：

```
// 对某个文件夹下的所有VCF及其GZ、BGZ压缩文件进行转换
int thread = 4;
SDSVManager.of("inputDir")
  .setOutput("outputDir")
  .setCallingType("cuteSV")
  .run(thread);
```

从其他文件格式构建SDF存档

目前SV主要的记录和存储格式为VCF文件，因此在 SDFA 中不提供直接由其他文件格式构建SDF存档的命令行工具。但是基于广泛的SV文件兼容性和后续开发的SV分析工具，SDFA 提供了一个 API 已构建 SDF 存档。

SDFWriter 构建 SDF 存档

`SDFWriter` 是一个专门用于通过构造 `SDFWriterRecord` 直接构建 SDF 的类，其本质就是 `获得记录-设置记录-写入记录-重置记录-结束记录` 的过程。所以用户需要思考如何将一个输入的文件格式转为一条 `SDFWriterRecord` 记录。

下面我们给出一个 `SDFWriter` 的使用实例：

> [!NOTE|label:Example 1]
>
> SDFWriterRecord的本质是一个SV记录，因此我们需要构建一个SV记录并手动写入文件中。

```
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("ACGAGGGCCCCAA"))
                .setChrName("chr1")
                .setType(SVTypeSign.getByName("DEL"))
                .setID(new Bytes("ID_0"))
                .setRef(new Bytes("ACGAGGGCCCCAA"))
                .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();
    }
```

上述实例主要构建 SV 并写入 SDF 文件中，因此对于一个任意格式的输入文件，用户可以通过 `line -> SV` 的形式来进行文件转化。

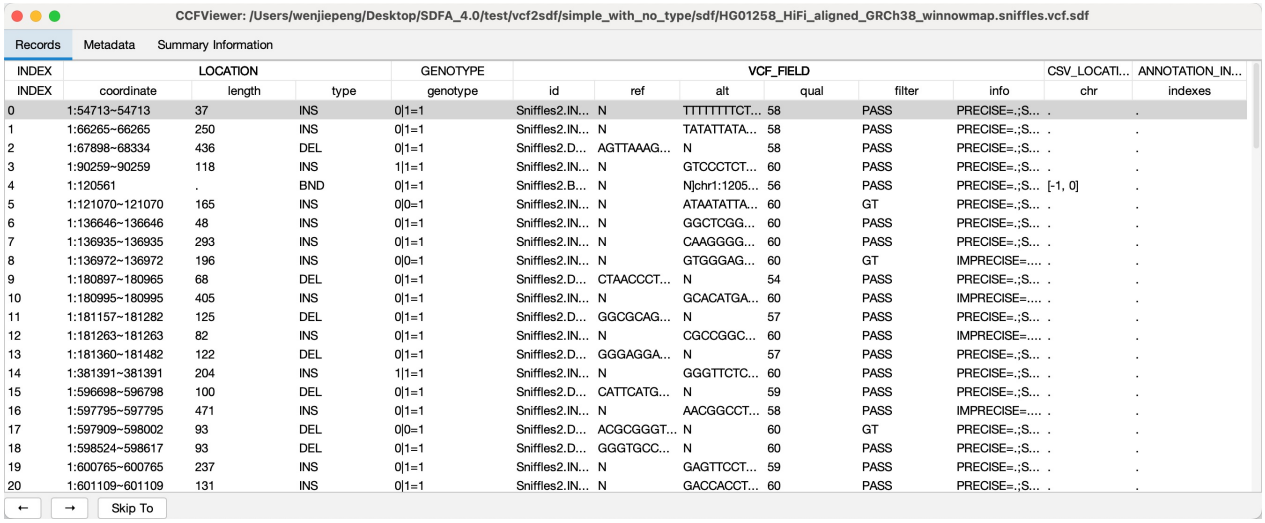
API文档

具体可以查看 `SDFWriter` 类已构建SDF存档

图形化界面

为了方便查看 SDF 文件，SDFA设计了一个图形界面，充分利用了 SDF 文件的块和列特性，具有以下特点：

- 本地扫描：仅扫描正在查看的内容，利用块和列特性。
- 页面导航：支持低内存使用的快速导航。
- 自定义显示：当用户有自定义编码方法时，允许设置查看方法。
- 完整信息：显示VCF头信息、SV信息和文件大小的源信息。



过滤模式

相比于单核苷酸多态性(single-nucleotide polymorphism，SNP)而言，SV的过滤筛选往往缺乏统一标准，同时筛选的属性也更加多元(可能包含INFO、QUAL和GT字段等)。

为进行更全面的SV过滤，SDFA内置了丰富的过滤功能。具体而言，主要覆盖两方面的过滤：

- `Genotype level`：设置质控信息对基因型过滤——不满足条件的设置为`./.`
- `SV level`：设置自定义函数对当前SV记录的`CHR`、`ID`等多个VCF字段进行过滤

具体的指令为

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

Genotype 过滤

使用下述指令进行过滤：

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

上述命令行对指定的`format_attr`进行过滤，而`function`是一个返回`boolean`值的表达式。

为方便用户直接对每一个基因型的质控属性的值进行操作，我们在function中直接使用value代表单个基因型的值，下面我们举一个例子：

> [!NOTE|label:Example 1]
>
> 我们对`./data`文件夹下的所有SDF文件的基因型进行过滤，过滤质控属性为`GQ`，过滤条件为`(int)value>20`，输出到`./`文件夹中：
>
> ```
> java -jar sdfa.jar filter -d ./data -o ./ --filter-gty GQ (int)value>20
> ```
>
> 如果SDF文件为多样本文件，依次对每个样本的基因型进行上述条件过滤。

为简化转化逻辑同时减少`metric`数据的存储空间，我们将常见的质控属性封装特定类型以方便用户进行直接操作，具体如下：

抽象类型	存储类型	获取值类型	质控属性	操作示例	操作解释
SingleIntValueBox	`int`数组	单个`int`值	`DP`，`DR`，`DV` `GQ`，`MD`，`PP`	`--filter-gty DP (int)values>10`	将DP属性值<=10的基因型为`./.`
TwoIntValueBox	`int`数组	2个`int`值组成的IntList类	`AD`，`RA`	`--filter-gty AD ((IntList)values).get(0)>10`	将AD属性值的第一个int值<=10的基因型为`./.`
ThreeIntValueBox	`int`数组	3个`int`值组成的IntList类	`PL`	`--filter-gty PL ((IntList)values).get(0)>10`	将PL属性值的第一个int值<=10的基因型

					为 ./.
SingleStringValueBox	string 数组	单个 string 值	FT others	--filter-qty FT (string)values.equals(\"TRUE\")	将FT属性不为TURE的基因型设置为 ./.

SV 过滤

与基因型的指令类似，SV过滤的指令如下：

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

上述命令行对指定的 sv_attr 进行过滤，而 function 是一个返回 boolean 值的JAVA表达式，下面举个例子：

> [!NOTE|label:Example 2]
>
> 我们对INFO字段中的 IMPRECISE 进行过滤，当存在该字段时我们过滤该SV：
>
> ```
> java -jar sdfa.jar -d ./data -o ./ --filter-sv IMPRECISE value!=null
> ```

上述例子中的 sv_attr 为IMPRECISE，在SDFA设计中 sv_attr 的定位和获取是由一定顺序的，下面是全面的过滤顺序：

INDEX	匹配字段	获取值类型	操作解释
1	CHR	string	获取SV的染色体名称
2	ID	Bytes	获取SV的ID值
3	REF	Bytes	获取SV的REF值
4	ALT	Bytes	获取SV的ALT值
5	QUAL	Bytes	获取SV的QUAL值
6	FILTER	Bytes	获取SV的FILTER值
7	INFO	List<Bytes>	获取SV的所有INFO字段值
8	FORMAT	List<Bytes>	获取SV的所有基因型的质控值
9	GT	CacheGenotypes	获取SV下所有样本的基因型
10	LEN	int	获取SV的LEN值
11	SDF_FIELD_NAME	Object	获取SV在 SDF_FIELD_NAME 的值
12	INFO_ATTR	Bytes	获取SV在INFO中KEY为 INFO_ATTR 的值
13	FORMAT_ATTR	Bytes	获取SV在FORMAT中KEY为 FORMAT_ATTR 的值
14	UNKNOWN	ERROR	.

详细获取值可以查看 SDFRecordWrapper 的 getV 函数

> [!TIP|label:过滤顺序]
>
> 值得注意的是，在实际代码执行中会首先进行 SV Level 的过滤，然后再进行 Genotype Level 的过滤。

样本提取

在大型数据集中往往需要提取固定的样本进行下游分析，例如UKB的 Whole genome GraphTyper SV data [interim 150k release] 是已经被合并好的150k 合并的数据集，此时我们需要某些疾病的case-control样本时就需要进行提取操作。

当该VCF文件被转为SDF文件后，使用下述指令对SDF进行提取：

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

这里我们使用PED文件作为输入文件以从群体SDF文件中提取所需的样本，同时SDFA为提取样本后的SV提供了一些基本的SV筛选函数。

> [!NOTE|label:Example 1]
>
> 我们对输入文件夹的所有SDF文件进行提取：
>
> ```
> java -jar sdfa.jar extract -d ./data -ped ./ped.ped -o ./
> ```

程序参数

```
语法：extract -d input_dir -o output_dir -ped ped_path
Java-API: edu.sysu.pmglab.sdfa.toolkit.SDFExtract
关于：提取多个 SDF 文件中的PED中的样本
参数：
    *--output, -o          设置输出文件夹.
                                        格式：-o <dir>
    *--dir, -d                 设置输入文件夹.
                                        格式：-d <dir>
    *--ped-file, -ped     设置PED文件
                                        格式：ped <file>
    --thread, -t.             设置线程数
    --max-maf                     设置提取样本中含有基因型的最大比例
    --min-maf                     设置提取样本中含有基因型的最小比例
```

API工具

对 SDF 文件进行提取的 API 工具是SDFExtract，使用示例如下：

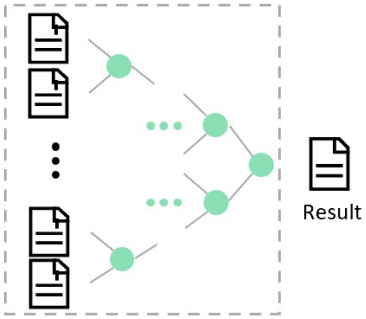
```
SDFExtract.of(file.toString(),
              sdfExtractProgram.pedFile,
              FileUtils.getSubFile(sdfExtractProgram.outputSDFDir, file.getName())
              )
                    .setMaxMAF(sdfExtractProgram.maxMaf)
                    .setMinMAF(sdfExtractProgram.minMaf)
           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

合并多个**SDF**文件

连接即按行添加新的变异位点，最常见的情形是将分散在不同文件（按照染色体、按照变异位点数量、文件大小储存）的同一批受试者的变异位点重新拼接回单独的文件（例如：UKB的 Whole genome GraphTyper SV data [interim 150k release] 数据）。使用以下指令对 SDF 文件进行连接：

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

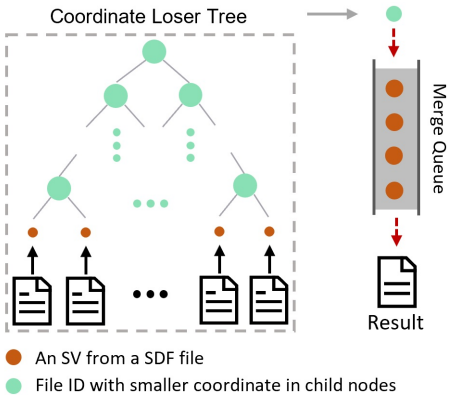
多个文件合并时会检查所有文件的样本名，并在每个线程中使用 2路归并排序 进行合并。合并过程中保证坐标有序并持续更新META信息。



> [!NOTE|label:Example 1]
>
> 下面是对3个单样本文件进行合并（此处假设3个文的样本名一致）：
>
> ```
> java -jar sdfa.jar concat -d ./data -o ./ -t 4
> ```

样本合并

为了进一步开展更大规模的 SV 研究，SDFA 开发了一种 cohort-wide 的合并算法，用于群体规模水平的SV样本合并。该算法基于SDF文件，在有序考虑所有SV数据的基础上进行k路合并：



目前SDFA使用position进行样本间的SV合并，具体合并逻辑如下：

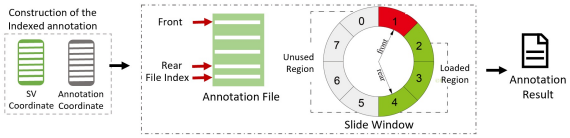
SDFA维护相同类型和位置的sv有序列表，以最小位置依次添加相同类型的sv。当新增SV不满足合并条件时，当前列表中的所有 SV 都会弹出并合并为一个 SV。默认的合并条件如下：

$$|Pos_{first} - Pos_{new}| < Threshold$$
$$|End_{first} - End_{new}| < Threshold$$

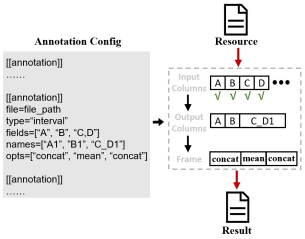
> [!NOTE|label:Example 1]
>
> SDFA将指定文件夹中的VCF文件（包括压缩文件和SDF文件）合并，并将合并结果输出到指定文件夹。命令行如下：
>
> ```
> java -jar sdfa.jar -d ./data -o ./
> ```

功能注释

为了进一步定位易感 SV 并探索其生物学原理，往往需要对 SV 进行功能注释。为快速进行多样本多资源的功能注释，SDFA 设计了下图所示的索引滑动窗口算法加速注释过程。



同时，为进行多资源定制化注释，SDFA 设计了 `post-annotate` 注释方式，通过下图所示的配置文件可以定制输出文件：



配置文件

SDFA 借鉴Vcfanno工具的 `post annotation` 概念，定义了一个用于配置注释资源和输出结果的配置文件，具体解释如下：

参数	参数解释	是否必须 (*代表必须，.代表可选)
`[[annotation]]`	一个新注释资源的开始标识符	*
`file`	注释资源文件的完整路径	*
`type`	注释资源的类型 (当前支持 `gene` 、 `interval` 和 `svdatabase` 三种类型)	*
`names`	输出结果的列名	.
`fields`	输入文件的列名	.
`opts`	对应输出列的函数	需要和`names`大小匹配

对于上述参数，我们举一个例子：

> [!NOTE|label:Example 1]
>
> 下面我们对 `interval_1.txt` 文件进行操作，该文件是一个 `interval` 类型的文件，文件大致为：
>
> **interval_1.txt**
>
> ```
> #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
> ```
>
> 单行之间的元素通过 `\t` 进行分割，现在我们想获取与SV相关的 `COL1` 列，因此此处设置 `fields=["COL1"]` ，同时我们想输出列为 `name` ，因此设置 `names=["name"]` 。最后的配置文件如下：
>
> ```
> [[annotation]]
> ```

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

完成上述配置文件后，我们通过下述指令进行注释：

```
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
```

注释后结果如下：

```
#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
```

其中fileID对应的是该文件夹(`/Users/wenjiepeng/Desktop/SDFA_4.0/test/vcf`)下不同的SV文件。

注释资源

SDFA 支持集成外部数据库进行 SV 注释，前提是外部数据文件符合以下基本格式要求：

- 必须使用 tab符（ `\t` ）作为分隔符。
- 注释文件必须包含一个标题行，其列名以数字符号（ `#` ）开头（以 `##` 开头的行将被忽略）。

另外，SDFA 在引入区间注释文件和 SV 数据库文件对已有SV数据进行注释时，必须满足以下具体条件：

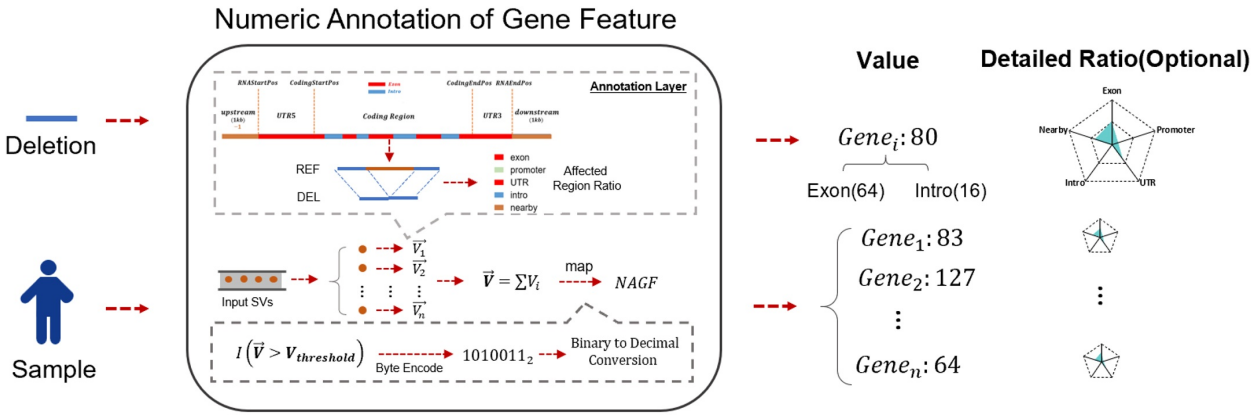
- 对于区间注释文件而言，需满足如下要求：

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

- 对于SV数据库文件，需满足如下要求：

  | 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 | | … | … | … | … |

数值化基因注释



基因特征数值注释（NAGF）是 SDFA 针对 SV 提出的一种新的注释结果。NAGF 用 8 位表示受影响的基因特征区域（如外显子、内含子），用 5 个字节表示基因中单个 SV 内每个基因特征受影响区域的比例。8 位构成特征位，5 个字节构成覆盖范围。下面是关于 NAGF 的详细信息：

- 特征位：该字节充分利用 8 位来表示一个基因的不同功能区。第一位表示该基因是否是蛋白质编码基因。它分别表示受影响的外显子、启动子、UTRs、内含子和附近区域，0 表示未受影响，1 表示受影响。它还表示通过拷贝数变异（CNV）和倒置的完全覆盖。注意，类似的想法可以使用更多的位来表示更微妙的特征，例如 5'UTR 和 3'UTR。
- 覆盖字节数：本研究中的这五个字节代表五个基因特征区域中 SV 的受影响百分比（范围从 0 到 100）：外显子、启动子、UTR、内含子和附近区域。

最终结果如下所示：

> GENE_NAME:Value:[xxxx， xxx，xx，xx，xxx，x]
>
> [!NOTE|label:Example 1]
>
> 使用 SDFA 的 NAGF 对示例文件夹进行注释，并输出到用户主目录下的 tmp 文件夹中：

```
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
```

基于**SV**的**GWAS**分析流程

基于 SV 的 GWA 研究已经出现。由于 PLINK 集成了许多实用的统计测试方法，SDFA 为进一步挖掘易受影响的 SV 提供了基于 SV 的 GWA 工作流程。

以 UKBB 数据库为例，当样本量为海量时，考虑到数据量巨大，将总体 SV 数据拆分为多个 VCF 文件，每个文件存储整个 SV 数据的一部分。同时，由于 PLINK 已经集成了许多 GWA 相关的统计分析工具，SDFA 提供了 SDF2Plink 工具，将 SDF 文件转换为 Plink 输入文件，用于后续基于 SV 的 GWA 分析。

SDFA 已经建立了一个 VCF ⇒ SDF ⇒ Plink 的模式，它主要包含4个过程：

- `VCF2SDF`：将 VCF 转换为 SDF 文件，并执行 SV 过滤、信息提取等操作。
- `SDFConcat`：将多个 SDF 文件集成到一个 SDF 文件中。
- `SDFExtract`：从集成的 SDF 文件中提取部分样本信息。
- `SDF2Plink`：将提取的样本转换为 Plink 文件格式，即 `.fam`、`.bed` 和 `.bim` 文件。

> [!NOTE|lable:Example 1]
>
> 这里我们举一个例子，首先对`./test/resource/gwas`中的多个SDF文件进行集成，接着提取`./test/resource/gwas/sample.ped`中的样本，最后转化为PLINK文件。最后使用PLINK文件进行分析

```
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
```