[jellyfish][https://bioinformatics.uconn.edu/genome-size-estimation-tutorial/#]

Genome Size Estimation Tutorial

给定长度为L的序列, k-mer长度为k, 总比对k-mer数量为: (L-k)+1:

```
GATCCTACTGATGC ( L = 14 ) on decomposition of k-mers of length k = 8,

Total number of k-mer's generated will be
n = ( L - k ) + 1
  = (14 - 8 ) + 1
  = 7

GATCCTAC,     ATCCTACT,     TCCTACTG,     CCTACTGA,     CTACTGAT,     TACTGATG,
   ACTGATGC
```

针对基因组而言, 该k-mer数量就非常接近于基因组真实长度:

k=18			
Genome Sizes	Total K-mers of k=18	% error in genome estimation	
L	N=(L-K)+1		
100	83	17	
1000	983	1.7	
10000	9983	0.17	
100000	99983	0.017	
1000000	999983	0.0017	**1MB genome size**

如果存在10 拷贝序列(14bp), 总k-mer数目为(k=8), 那么k-mer数量就为:

```
n      = [( L - k ) + 1 ] * C
       = [(14 - 8 ) + 1] * 10
       = 70
```

获得真实基因组长度:

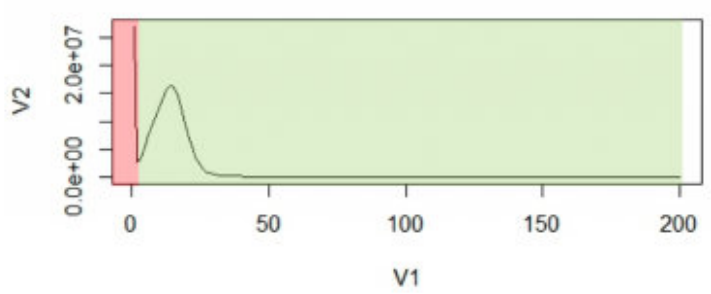
```
       = n / C
       = 70 / 10
       = 7
```

因此, 测序过程中获得了C 个拷贝基因组, 也就是覆盖度为C, 因此获得实际基因组大小: N=n/C

k-mer Distribution of a Typical Real World Genome

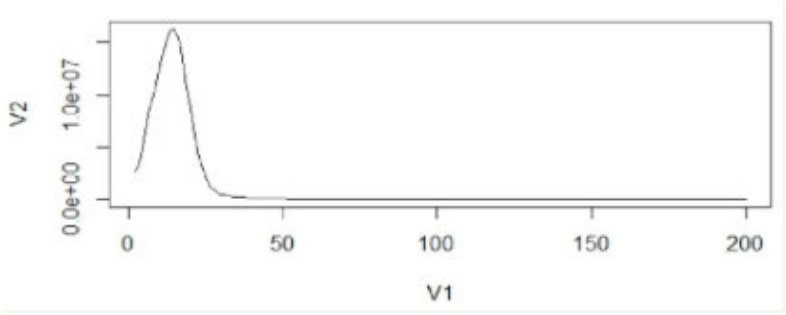
k-mer大小应该足够大实现该k-mer在基因组上唯一比对. 过大的k-mer将会导致计算资源的过度使用.

首先计算k-mer的频率用来判断测序过程中基因组的覆盖度:



x轴v1为指定k-mer在测序数据中出现的次数, y轴v2为给定出现次数下k-mer的总数目

图中第一个峰为reads中的测序错误所致, 忽略.



假设k-mer为唯一匹配到了基因组, 因此应该在基因组中仅出现一次, 这样次数值反应了基因组的覆盖度. 那么为了计算目的, 使用上图的平均覆盖度14, 曲线下面积将代表总的k-mers数目, 那么基因组评估为:

```
N = Total no. of k-mers/Coverage
  = Area under curve /mean coverage(14)
```

---

1. Count k-mer occurrence using Jellyfish

```
jellyfish count -t 8 -C -m 19 -s 5G -o 19mer_out --min-qual-char=?
/common/Tutorial/Genome_estimation/sample_read_1.fastq
/common/Tutorial/Genome_estimation/sample_read_2.fastq
```

-t / -treads=unit32 运行线程

-C -both-strands 计算双链

-m -mer-len=unit32 k-mer长度

-s -size=unit32 Hash size/memory allocation :

```
例如, 测序错误率为e( illumina reads, e ~ 1%), 评估的基因组大小为G, 覆盖度为c, 那么期待的k-mers数目为G+G*c*e*k
不同于jellyfish 1, 该-s参数仅是个估计量. 加入指定的size太小而无法匹配所有的k-mers, 该hash size将自动增加或部分结果将会输出到硬盘且最终自动合并
```

-o -output=string 输出文件名

--min-quality-char 碱基质量值. 2.2.3版本使用'Phred'值, "?"=30

输出文件为19mer_out, 使用jellyfish histo创建点图:

```
jellyfish histo -o 19mer_out.histo 19mer_out
```

根据输出点图绘制曲线图:

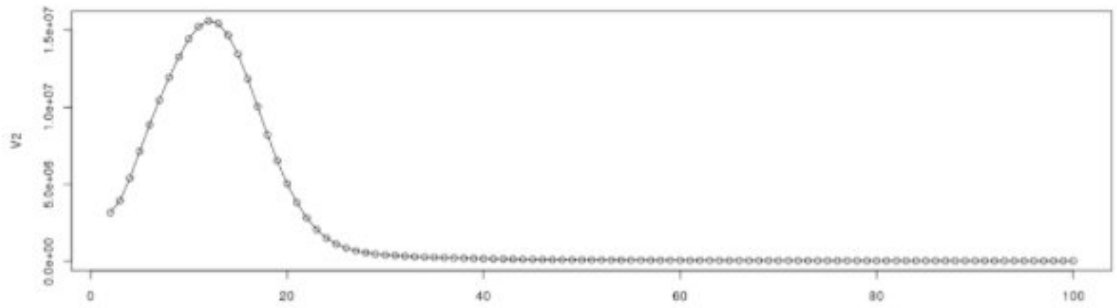
```
dataframe19 <- read.table("19mer_out.histo") #load the data into dataframe19
plot(dataframe19[1:200,], type="l") #plots the data points 1 through 200 in the
dataframe19 using a line
```

去除测序错误带来的偏移:

```
plot(dataframe19[2:200,], type="l")
```

计算单拷贝k-mer区域和总k-mer数目

```
plot(dataframe19[2:100,], type="l") #plot line graph
points(dataframe19[2:100,]) #plot the data points from 2 through 100
```



由上图可知单个拷贝基因组的点图应该位于2到28

假设总的数据点为9325, 那么上图分布中总k-mers数目为:

```
sum(as.numeric(dataframe19[2:9325,1]*dataframe19[2:9325,2]))
```

为: 366790981

计算顶点位置和基因组大小

根据上图数据, 可以直观看到其顶点为k-mers在12的位置, 因此该基因组大小为:

```
sum(as.numeric(dataframe19[2:9325,1]*dataframe19[2:9325,2]))/12
```

为: 305659151 ~ 305Mb

因此其当拷贝基因组区域大小可计算为(2-28):

```
sum(as.numeric(dataframe19[2:28,1]*dataframe19[2:28,2]))/12
```

为: 213956126 ~ 213Mb

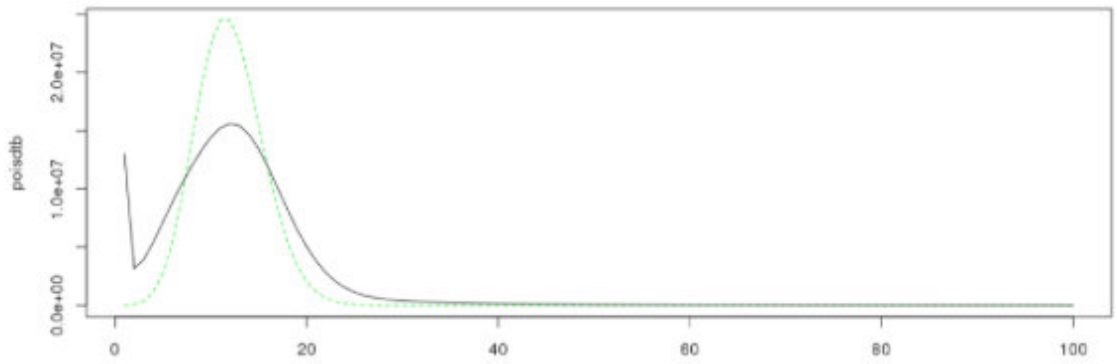
因此单拷贝基因组区域比上总基因组大小为:

```
(sum(as.numeric(dataframe19[2:28,1]*dataframe19[2:28,2]))) /
(sum(as.numeric(dataframe19[2:9325,1]*dataframe19[2:9325,2])))
```

为: 0.6999827 ~ 70%

和泊松分布比较峰型

```
singleC <- sum(as.numeric(dataframe19[2:28,1]*dataframe19[2:28,2]))/12
poisdtb <- dpois(1:100,12)*singleC
plot(poisdtb, type='l', lty=2, col="green")
lines(dataframe19[1:100,12] * singleC, type = "l", col=3)#, Ity=2)
lines(dataframe19[1:100,],type= "l")
```



同样迭代使用不同的k-mer长度:

K-mer length	19	21	23	25	27	29	31
total No of fields	9741	9653	9436	9325	9160	8939	8818
Total K-mer count	3.66E+09	4.4E+09	4.53E+09	4.67E+09	4.26E+09	4.44E+09	3.98E+09
Genome size	3.05E+08	2.9E+08	3.02E+08	3.14E+08	3.04E+08	3.41E+08	3.06E+08
single copy region	2.13E+08	2E+08	2.08E+08	2.25E+08	2.21E+08	2.36E+08	2.3E+08
Proportion	0.69998	0.69403	0.688915	0.716151	0.725814	0.690277	0.750801

Practice

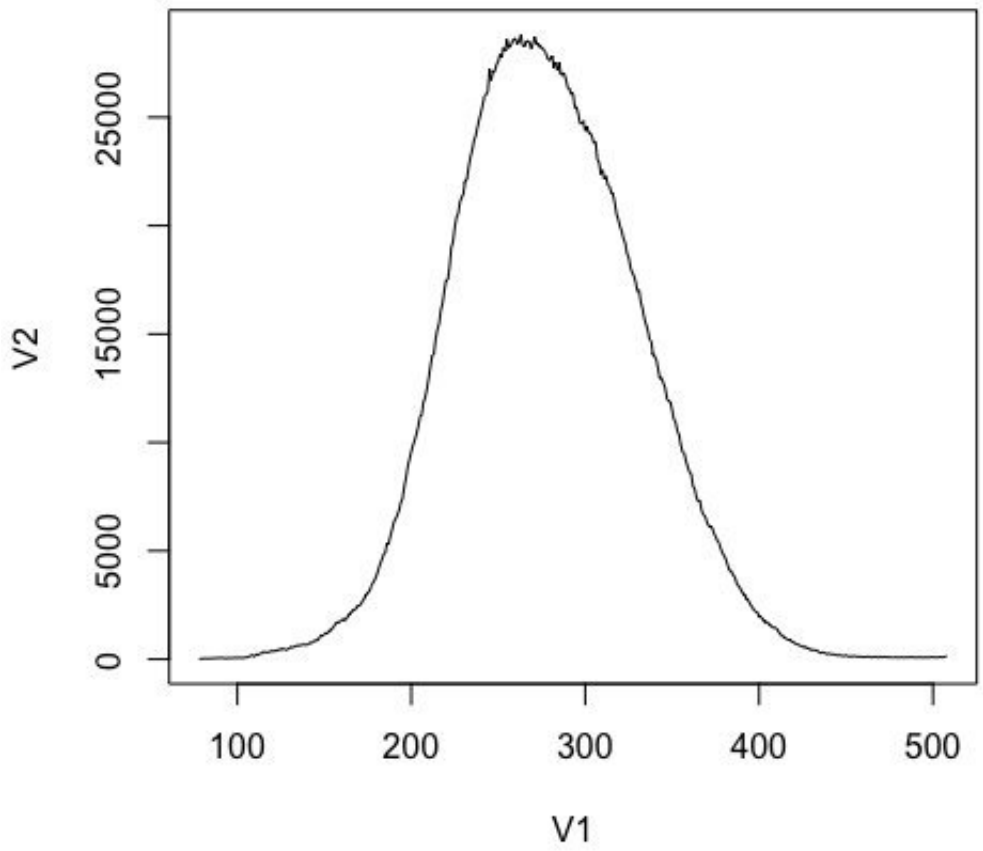
`jellyfish count -C -m 19 -s 100M -o 19mer_out ICU43_galore_R1_trimmed_1P_val_1.fq ICU43_galore_R2_trimmed_2P_val_2.fq 1>jellyfish_19mer.log 2>&1`

`jellyfish histo -o 19mer_out.histo 19mer_out`

查看数据, 删除测序错误带来的峰:

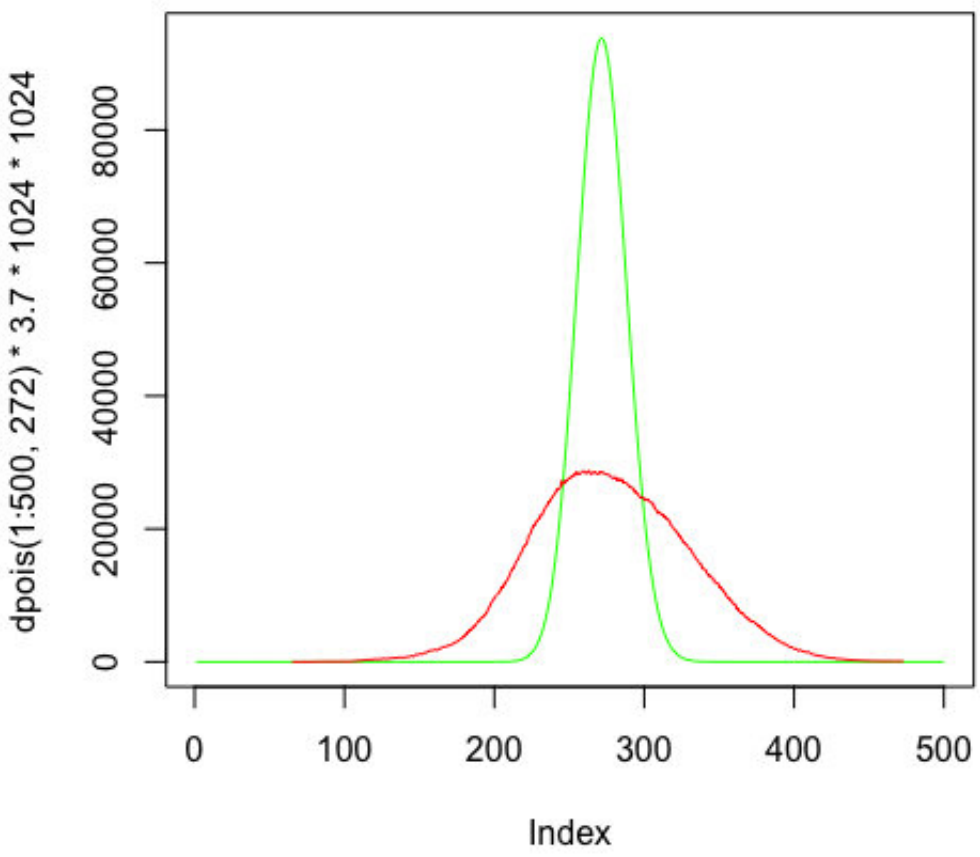
```
> sum(dataframe19[75:500,1]*dataframe19[75:500,2])/(273*1024*1024)
[1] 3.69691
> sum(dataframe19[75:4000,1]*dataframe19[75:4000,2])/(273*1024*1024)
[1] 3.917221
> sum(dataframe19[75:500,1]*dataframe19[75:500,2])/sum(dataframe19[75:4000,1]*dataframe19[75:4000,2])
[1] 0.9437583
>
```

对应峰图:



对应泊松分布:

```
> plot(dpois(1:500,272)*3.7*1024*1024,type="l",col="green")
> lines(dataframe19[60:465,],type="l",col="red")
>
```



2. Counting K-mers in a genome

针对实际的基因组或者组装完成的基因组, k-mer和其反向互补序列是不相等的, 因此使用`-c`是没有意义的. **此外, hash的大小可以直接设置为基因组大小.**

jellyfish提供两种方式计算高频的k-mers(仅包含k-mers数目大于1), 该方法显著减少内存使用. 两种方法都是基于Bloom filters. 第一种为one pass method, 提供部分比例k-mers的大概计数, 第二种方法提供精确计数. 两种方式中, 大部分低比例的k-mers都没有报出.

One pass method

使用`--bf-size`选项, 应为数据集中期待的总共的k-mer数目, `--size argument`应为出现至少一次的k-mers的数目:

```
jellyfish count -m 25 -s 3G --bf-size 100G -t 16 homo_sapiens.fa
```

这里将会适当地使用30x覆盖度的人基因组reads, 计算25-mers数目. The approximate memory usage is 9 bits per k-mer in the Bloom filter.

每个k-mer的计数结果(jellyfish dump/jellyfish query)将比实际少1, 例如, count 2 k-mer将为1

缺点就是一些比例的k-mer不应该报告出来(count为1).

Two pass method

首先使用`jellyfish bc`构建reads的Bloom counter, 然后使用`jellyfish count`命令计算

```
jellyfish bc -m 25 -s 100G -t 16 -o homo_sapiens.bc homo_sapiens.fa
```

```
jellyfish count -m 25 -s 3G -t 16 --bc homo_sapiens.bc homo_sapiens.fa
```

该方法有点是计算所有correct的k-mers计数, 大部分count 1的k-mer不会报告; 缺点就是需要解析所有reads两次.