

联合概率压缩的实验报告

钟家杰 22009290034



2024-4-24

西安电子科技大学

西安市长安区西沣路兴隆段266号

# 联合概率编码的实验尝试

目录

[**1.** **联合概率编码的功能描述** 1](#_Toc164864781)

[2. 联合概率编码问题的提出背景 1](#_Toc164864782)

[3. 联合概率编码的算法实现 2](#_Toc164864783)

[4. 仿真实验结果与分析 8](#_Toc164864784)

[5. 算法和创新点总结 9](#_Toc164864785)

[6. 过程的心得和体会以及一些错误的尝试 9](#_Toc164864786)

[7. 附录 17](#_Toc164864787)

## **联合概率编码的功能描述**

本实验实现的是对基因压缩里面的压缩进行改进。在原来的AVS-g框架下，对碱基和质量分数是分开进行压缩的，也就是说，对于碱基和质量分数的压缩参考的上下文是割裂开来的。由于在测序的fastq文件里面，碱基和质量分数具有十分密切的依赖关系，基于这个理论，本实验对基因压缩里面的碱基和质量分数的压缩进行了联合概率的编码，即在原来的对碱基和质量分数基础上，将质量分数和碱基的上下文结合起来，在增大上下文的同时，对质量分数在31~38之间的部分和碱基进行结合，从而达到提高压缩率和时间的目的。

******

# 联合概率编码问题的提出背景

基因压缩是指通过各种技术手段对基因组进行压缩，以减少基因组的大小，同时尽可能地保留其功能和信息。这一技术的发展主要受到以下几个因素的驱动：

A. 数据存储和处理需求增长：随着基因组学和生物信息学的发展，大量的基因组数据被生成和积累。存储和处理这些海量数据需要庞大的计算资源和存储空间，因此压缩基因组数据可以显著减少存储和处理成本。

B.基因组编辑和合成需求：在基因编辑和合成生物学领域，研究人员需要处理大量的基因组数据，包括设计、编辑和合成新的基因组。压缩基因组数据可以提高这些操作的效率和速度。

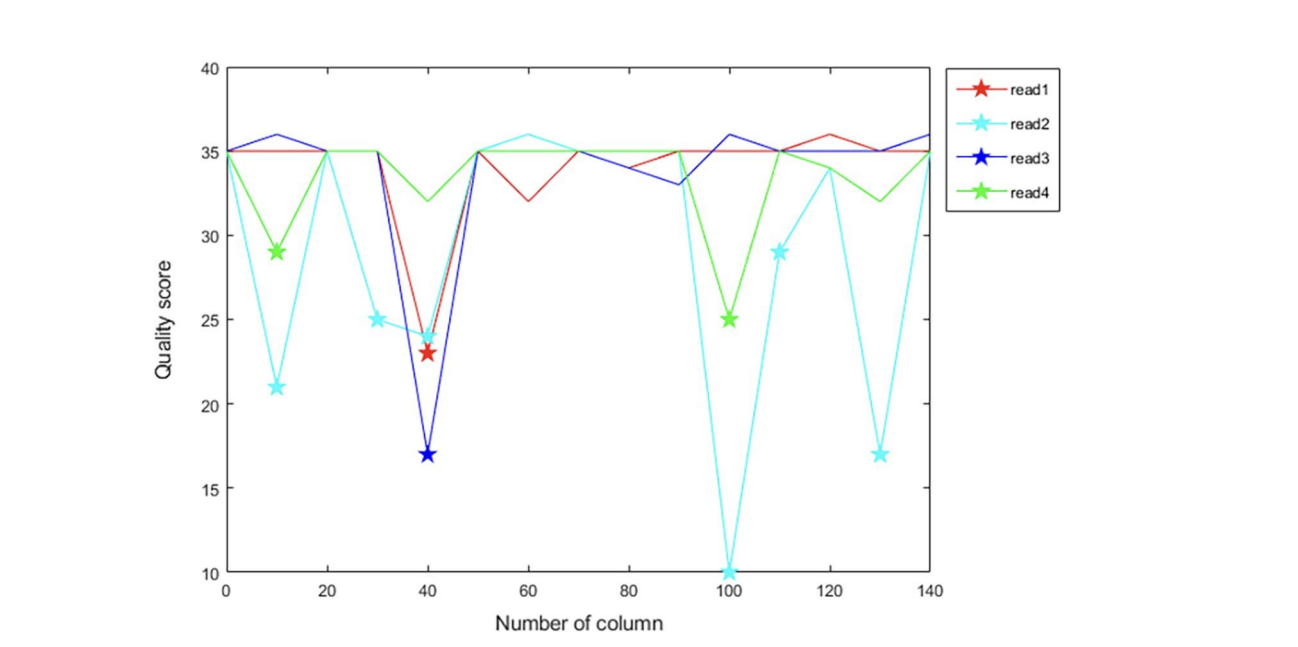
C. 基因组信息挖掘：对基因组数据进行分析和挖掘可以揭示生物学系统的结构和功能。压缩基因组数据可以加快这些分析过程，提高数据挖掘的效率。

D.基因组传输和分享：在基因组学研究和医学应用中，需要共享和传输基因组数据。通过压缩基因组数据可以降低数据传输的成本和时间。

而目前，基因压缩技术主要分为两类：有损压缩和无损压缩。有损压缩会删除一些基因组数据以降低文件大小，但可能会导致信息损失。无损压缩则尽量保留所有信息，但压缩率通常较低。研究人员正在不断改进这些技术，以实现更高的压缩率和更好的信息保留效果。因此，不断地对无损压缩来进行改进就显得尤其重要。在此时，我们提出联合概率进行编码，对碱基和质量分数的上下文进行结合，通过重新调整上下文分布以及联合压缩的方式来尝试进行压缩率的提高。

# 联合概率编码的算法实现

在原来的AVS-g的编码里面，无参考的压缩对碱基的压缩采用的是进行字典树的建立来进行参照，在有参压缩里面，对碱基的压缩是采用的应用参考序列进行压缩，而对质量分数的压缩是采用蛇形遍历的方法来参考前面的四个质量分数来建立上下文进行压缩。在我们的联合概率编码里面，我们把碱基和质量分数的上下文进行合并，即在原来acocompress的基础上进行改进，引入碱基的序列作为上下文进行处理,这里我们考虑到质量分数分布的特殊性，即由于利用光学特性进行测序的原因，碱基会出现质量分数偏低的可能性存在（如下图）：



所以我们分别计算model\_idx\_seq和model\_idx\_qual，在计算model\_idx\_seq的时候，对于第一列压缩的碱基，由于前面没有上下文的存在，所以我们可以预设一个上下文模型，这里设置为255（在随后的结果中对压缩率有一定的提高），在计算model\_idx\_qual的时候参考前面的质量分数的时候，采用质量分数偏大的那个来进行参考上下文的构建，最后我们把两者相加，得到model\_idx\_combine的压缩模型来调用算术编码器对其进行压缩：





接下来是对在31~38之间的，以及等于0的质量分数和其对应的碱基进行联合压缩，我们采用匹配的算法，即为31~38之间的质量分数和碱基分配一个特定的数字，例如：当碱基为A，质量分数为0时，我们给予一个0的编码；对碱基为A，质量分数为32的，我们给予一个6的编码。其他的类似推导下去，这样我们就把这部分的碱基和质量分数组合成为一个整体，从而使得这个赋予的进行编码的数字就可以代表一个碱基和一个质量分数的信息，而对于其他质量分数的碱基，我们根据5类不同的碱基来进行编码分配，即碱基为A时，分配46的编码，接着再进行二次压缩，把其对应的质量分数进行压缩。

代码如下：

这部分是利用map以及make\_pair算法来建立映射表，即把每类碱基和特定的质量分数进行分配编码：



接下来这部分是进行压缩：

 //================================combine compress====================================//

            if(quality\_block[offset]==0||(quality\_block[offset]>=31&&quality\_block[offset]<=38))

            {

                model\_combine[model\_idx\_combine].encodeSymbol(&rc,mapp[make\_pair(int(table[base\_block[offset]]),int(quality\_block[offset]))]);

            }

            else

            {

                int J=table[base\_block[offset]];

                switch(J)

                {

                case 0:

                    model\_combine[model\_idx\_combine].encodeSymbol(&rc, 45);

                    model\_combine[model\_idx\_qual].encodeSymbol(&rc, quality\_block[offset]);

                    break;

                case 1:

                    model\_combine[model\_idx\_combine].encodeSymbol(&rc,46);

                    model\_combine[model\_idx\_qual].encodeSymbol(&rc, quality\_block[offset]);

                    break;

                case 2:

                    model\_combine[model\_idx\_combine].encodeSymbol(&rc, 47);

                    model\_combine[model\_idx\_qual].encodeSymbol(&rc, quality\_block[offset]);

                    break;

                case 3:

                    model\_combine[model\_idx\_combine].encodeSymbol(&rc, 48);

                    model\_combine[model\_idx\_qual].encodeSymbol(&rc, quality\_block[offset]);

                    break;

                case 4:

                    model\_combine[model\_idx\_combine].encodeSymbol(&rc, 49);

                    model\_combine[model\_idx\_qual].encodeSymbol(&rc, quality\_block[offset]);

                    break;

                default:

                    break;

                }

至于解码的部分，我们采用编码部分同样的idx的计算，找到编码时用的idx，即可以从算术编码器里面解码出对应的字符，然后用特定的指针来进行存储便于进行对原fastq文件的复原。

代码如下：





# 仿真实验结果与分析

在仿真实验结果分析里面，我们对比三种压缩方法，即联合概率压缩，字典树压缩以及直接进行碱基和质量分数的压缩，我们分别从时间和压缩率的角度对三种压缩进行分析：

结果如下：

先是进行压缩率的比较：

|  |  |  |  |
| --- | --- | --- | --- |
| Fastq文件大小 | 联合概率压缩的压缩比 | 字典树的压缩比 | 直接分开压缩的压缩比 |
| 198MB | 43.91 | 42.71 | 39.05 |
| 326MB | 6.89 | 11.48 | 6.37 |
| 526MB | 8.15 | 10.30 | 7.93 |
| 60.7MB | 6.41 | 6.83 | 6.12 |
| 402MB | 12.73 | 20.45 | 12.51 |
| 126MB | 11.99 | 16.51 | 12.00 |

接着是运行速度的比较：

|  |  |  |  |
| --- | --- | --- | --- |
| Fastq文件大小 | 联合概率压缩的速度 | 字典树的速度 | 直接分开压缩的速度 |
| 198MB | 21s | 18s | 8s |
| 326MB | 17s | 345s | 16s |
| 526MB | 51s | 494s | 19s |
| 60.7MB | 8s | 18s | 3s |
| 402MB | 28s | 44s | 11s |
| 126MB | 8s | 13s | 5s |

解码时间比较：

|  |  |  |  |
| --- | --- | --- | --- |
| Fastq文件大小 | 联合压缩的解码时间 | 字典树的解码时间 | 直接分开压缩的解码时间 |
| 198MB | 8s | 17s | 10s |
| 326MB | 20s | 26s | 20s |
| 526MB | 28s | 32s | 19s |
| 60.7MB | 5s | 8s | 3s |
| 402MB | 15s | 16s | 14s |
| 126MB | 5s | 6s | 5s |

从以上的数据比对可以发现，从压缩比来看，字典树的压缩由于采用了建立字典树的比对操作，即对于比对上的部分只需要压缩一个首尾的位置，即和有参考压缩一样，因此建立字典树的压缩比是最高的，其次就是联合概率的压缩，最后是分开的压缩。由此我们也可以发现，联合概率的分布对于上下文的预测分布会比单独的各自参考更加准确，即联合概率更能够把相同一类的碱基和质量分数放在一起进行压缩，从而用了更少的码字来编码序列，从而使得压缩率得到提高。

从运行速度来看，字典树的运行速度是最慢的。因为字典树在编码的时候有一个建立字典树以及比对参考的过程，类似于有参压缩。因此，字典树的时间大多用于建立字典树和参考比对的过程，从而花费的时间大大增加。联合概率编码在对质量分数位于31~38之间的进行了码字的赋予过程，在编码的时候也有这样的查找比对过程，所以会略慢于直接分开进行压缩，但是压缩比会有所提高，但是其速度也是比字典树有了极大的提高。

# 算法和创新点总结

此实验运用的算法和创新点如下：

1. map和make\_pair的结合来建立编码映射表，即用映射表来解决碱基和质量分数的组合问题，便于解码能够解码出正确的码字，同时也实现了组合压缩；
2. 对碱基和质量分数的上下文进行了结合，即把两者进行结合，对其分布进行了重新调整，以此来提高压缩率；
3. 利用条件概率来进行算术编码；即我们利用上下文来计算当前要压缩的碱基和质量分数，把相同的一类放到同一个算术编码器里面进行压缩，在同一个算术编码器里面，对出现概率高的分配更少的码字，从而实现利用更少的码字实现更多的压缩。

# 过程的心得和体会以及一些错误的尝试

通过这次的对AVS-g的编程练习，我的算法编程能力，大工程处理能力，以及代码架构能力得到了极大的提高。

在此之前，我从未有过接触大工程代码的机会，从大二上学期开始，在导师和师兄的引导下，我开始看AVS-g这个大工程代码。一开始觉得十分的抽象，一个大工程代码里面的结构错综复杂却又相互紧密关联，牵一发而动全身。因此，在大二上学期我花费了大量时间来学习AVS-g的代码结构和思想，在此过程中也真正开始对这些企业级别的代码有了初步的了解。在大二期末考试结束，在和导师和师兄的交谈中，我开始尝试进行联合压缩，于是有了下面的几次错误尝试：

1. 考虑用多重继承来对seqprocess和qualprocess来进行合并：

因为考虑到了联合压缩，所以自然想到对原来的质量分数和碱基压缩进行合并，所以考虑到了利用多重继承或者直接进行合并的方法来进行实现，相关代码如下：









后来在与导师和师兄的交流中得知，实现联合概率的压缩不仅仅是简单的进行合并，还需要考虑重新调整上下文以及组合一起编码的问题，因此在重新明确了方向之后，我再次学习了AVS-g结构里面的acocompress的压缩，阅读了相关的文献以及作了大量的思考，最后仿照了qualprocess以及acocompress的压缩方法，自行写了combinecompress的压缩，并且把其加入了cmakelist进行编译，再次进行尝试的时候，写出来了如下的错误尝试代码：

B．错误尝试2：







上面的代码里面，我对其算出seq和qual的idx后，用了两次压缩，即把碱基和质量分数按照先后顺序进行了压缩，这样导致的结果是编号小的碱基和编号大的质量分数放在了同一个算术编码器模型里面进行了压缩，这样就导致编码器里面的概率分布不再是斜分布，导致出现的概率平均了不少，因此导致了编码率极大的降低，这里给出一次比对：

|  |  |  |
| --- | --- | --- |
| Fastq文件大小 | 联合概率编码的压缩比 | 此次错误做法编码的压缩比 |
| 60.7MB | 6.33 | 4.08 |

所以可以看出，由于概率分布的不理想以及二次压缩，导致了压缩比的大量降低，因此也说明了算术编码器的概率斜分布对其压缩率影响之大。

接下来就是在进行编码解码过程中遇到的一些问题以及解决问题的一些心得：

（1）.段错误：根据多次的尝试，发现基本段错误是由于栈溢出导致的，导致栈溢出，大多是由于数组越界，即当前使用的数组下标超出了数组的最大分配的空间，或者使用指针读取到了空的内容（可以是空指针也可以是空的文件）；解决办法也很简单，只需要检查代码里面的分配的内存空间是否够大，或者检查是否使用的sym超出了已经分配的最大范围来进行调整即可；

（2）浮点异常：出现这个问题的原因，基本都是在于model\_idx计算过程中出现了问题，即在解码过程中，我们利用前面已经解码出来的上下文来进行计算的时候，有一些碱基或者质量分数的还原出现了问题，就会导致编码和解码使用的model\_idx不相同，就会导致解码错误。在我进行处理的过程中，我采用了输出中间文件的方式来进行处理，即我们把编码和解码生成的idx输出到一个txt文件里面，我们在运行代码之后进行文件比对，从而找出是在哪一部分的的idx计算出现错误，最后根据我的结果，我发现是在解码的时候对碱基的处理，没有进行统一化的处理导致的，解决方法如下图的代码：



即我对算术编码器编码出来的碱基部分进行一个处理，使得解码能够得到正确的上下文，从而解码成功。

（3）seq MD5 check fail：这个问题的出现就是在编解码完成的时候，进行比对，看编解码出来的文件是否相同，不相同就产生如下结果。解决办法就是输出此时解码的文件，同样的进行文件比对，找出错误的碱基，然后返回比对编解码的代码，看是否出错即可。

（4）编译时候的错误：处理基本的语法错误之外，还有一种是在cmake编译过程中显示的未定义的错误，此时就涉及到了cmake的编译技巧，下面是cmakelist的部分代码：



我们需要在此段代码里面加入需要编译的.cpp文件，才能进行编译，同时对cmakelist的研究可以发现，在编译之后产生的文件，大多都是根据cmakelist来进行生成，典型的有：



这里要求了cmake的版本格式，以及生成的是AVS\_api的一个可执行文件

# 附录

参考文献：【[1](acocompress.pdf)】【[2](2304.01031.pdf)】

联合概率压缩的部分代码：【[3](code)】