算法 PJ 报告

马逸君 17300180070

问题描述

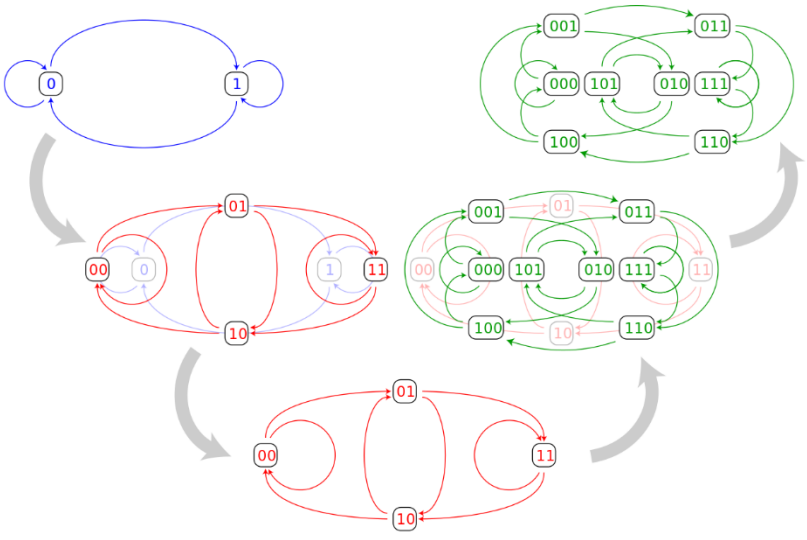
给定若干由参考 DNA 序列（基因组）复制若干次并打碎得到的长度相近的短 DNA 片段、由参考 DNA 序列直接测序得到的（错误率较高的）长 DNA 片段，要求将这些 DNA 片段拼接得到完整的 DNA 序列。

问题分析

在无参考基因组的情况下将短核酸组装为长核酸属于 De novo 序列组装问题。资料显示[1]，两种常用的算法是贪心和 de Bruijn 图。不同算法适用于不同场景，如较小的细菌基因组、较大的真核基因组、转录基因组等。

贪心算法适用于较小的测试数据，因为较大的测试数据中，贪心算法求出的局部最优解较全局最优解差距较大。贪心算法提出较早，早期的组装程序都使用贪心算法。一种著名的贪心算法是 OLC(overlap-layout-consensus)。

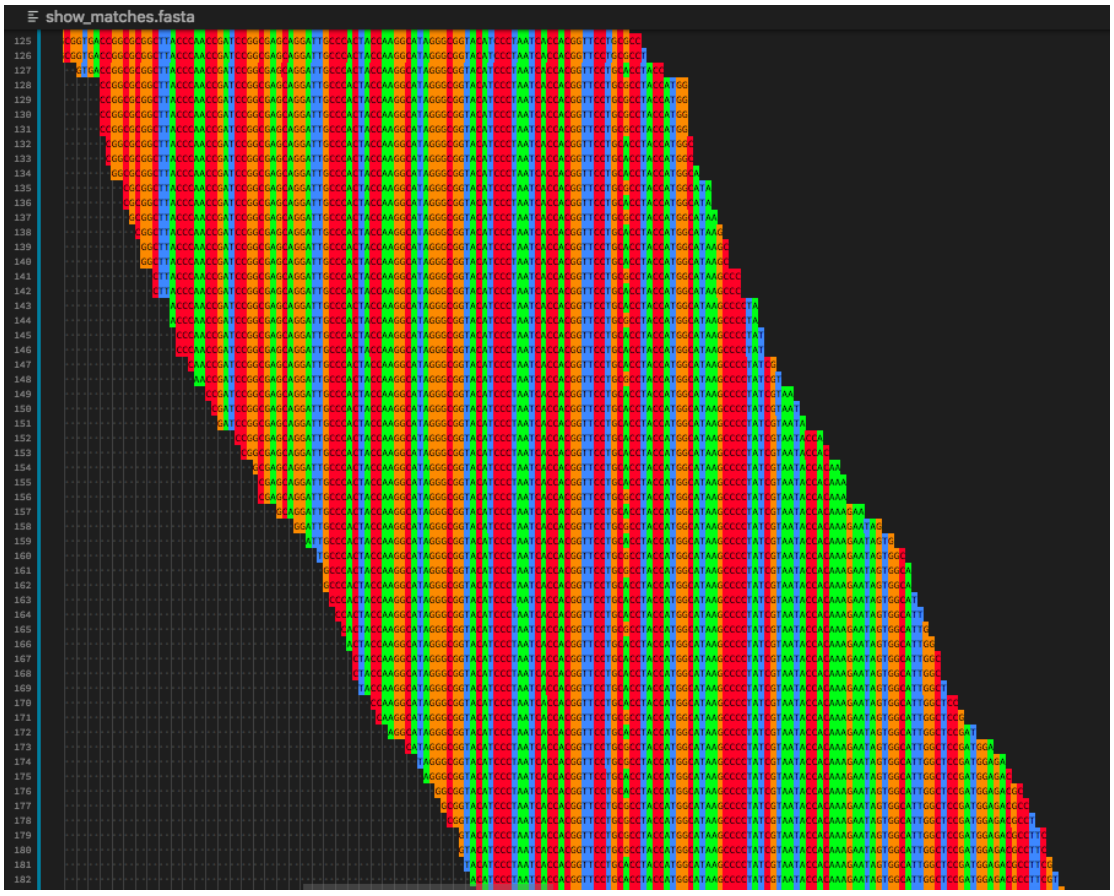
de Bruijn 图是 1994 年提出的，能够求出全局最优解而非局部最优解。其思想是将基因打碎成长度为 k 的片段(k-mers)，每个片段为一个顶点，有 k-1 位相等的两个顶点之间连边得到 de Bruijn图,图上的欧拉路径即为组装结果。



数据预处理

因为长序列的错误率高达 15%，直接使用长序列参与计算效果极差（事实上是反效果）。所以，我们必须设法利用已有的信息，修复长序列中的错误。

从张天翔学长的 github 我了解到这样一种预处理方法：利用错误率较低的短序列信息去匹配长序列中的位置，对每一个位点，如果匹配的**大多数**短序列在该点的值与长序列不同，我们以短序列为准。[https://github.com/zhangtianxiang/genome_assembler]



值得一提的是，因为可能出现插入、删除错，其中的“匹配”应当以编辑距离最小为标准，时间复杂度 $O(n^2)$。考虑到短序列有数千条，长序列有数百条，预处理会需要很长很长的时间。但是经过实验，我们发现实际测资中的错误绝大部分是替换错，这样我们就可以采用汉明距离为标准，时间复杂度 $O(n)$，回到可接受的范围内。*因为根据报告自己复现一直失败，不得不去读张天翔学长的代码了。

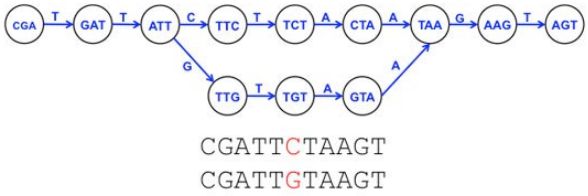
算法描述

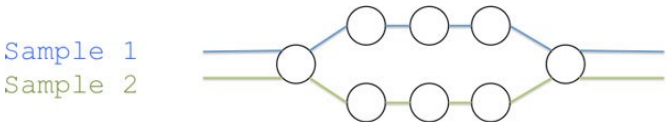
因数据规模较大，宜选用 de Bruijn 图方法求解。

- 第一步：从文件读入所有的序列，并计算出它们的反向、互补、反向互补序列。

- 第二步：建立 de Bruijn 图，同时统计所有节点的入度出度和每条边的重数。其中，输入的每个基因片段被打碎成其所有长度为 k 的子串（k-mers），这里的长度 k 是人为设定的，k 的更改会显著影响输出结果。

- 第三步：清洗数据，处理“气泡”(bubble)。短序列有 1% 的测序错误率，我们采用这样一种思路处理：测序错误将导致路径出现分支，因为错误是随机的且概率很低，





从开始出现分支的那个节点看去，错误分支的边的重数会明显低于正常分支：

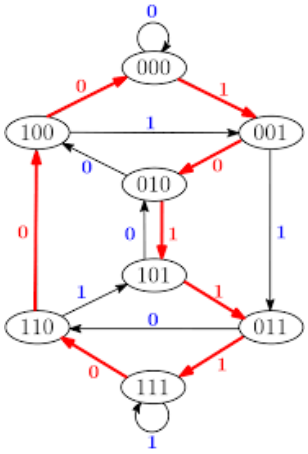
```
if ((double)p1->mul / p2->mul < 0.677) {
```

此时，我们遍历两个分支，到第一次走到入度大于 1 的节点时，删除较短的那个分支。这里取的值 0.677 是随意指定的，经测试，在合理范围内修改该值对答案影响很小。

- 第四步：组装基因片段。

  用 DFS 遍历所有路径。这里有两种方法找路径，一种是优先找环，这种方式复杂度会很高，以至于 data2、3、4 均会爆内存，但在 data1 中的效果非常好。另一种是每次找到长度最大的路径（路径长度记忆化），我在 data2、3、4 中都采用了这种方法。

  找到路径后记录该路径为答案，然后删除沿途的节点，循环进行该过程。



- 第五步：输出答案。因为最后可能剩下很多无法组装的零碎基因片段，对我们的答案没有帮助，所以我们设定一个最小输出长度限制，长度低于该限制的路径将不被输出。*这里隐含一个取舍问题(trade-off)，如果将最小输出长度限制设得很低，有助于增加 Genome-Fraction，但设得越低效果就越不明显，也会大大增加输出数据量和耗时，且可能对其他两项指标产生影响。测试表明，设为 400 是一个相对合理的值。

开发步骤与讨论

- 前半学期主要仿照 de Bruijn 图的模板复现算法。当时只使用了短序列，效果尚可，但也有改进空间。
- 后半学期主要研究洗数据。我复现了张天翔学长的洗数据程序（短序列修复长序列），data3 和 data4 取得了很好的效果，但 data1 和 data2 的结果反而是倒退了。这里值得一提的是 data1 即使只用短序列（我前半学期的程序）就取得了非常好的效果，由此可以猜测原因可能是长序列和短序列在内容上有较多重叠，短序列基本可以覆盖长序列内容，导致最终拼接出的基因组长度异常偏大。
- 此外，后半段有一个意外发现：如果更改 k-mers 的 k 值，效果可能变好。（灵感来源于陈中钰学长的程序[https://github.com/zychn/genome-assembly]）

  k 值的更改对结果的影响大致是：k 降低，则更倾向于拼接出长序列，但如果将 k 设得太小（如 k=11），最后会输出三倍于参考基因组长度的寥寥 2-3 段基因；而 k 增大，则更倾向于拼接出较多的短序列。原因也不难猜想，k 变小则意味着满足更短的重叠字段的基因片段会被拼接到一起，k 变大则相反，变得更“严格”。

  前半学期我听说一个经验常数是 k=25，但后半学期发现，如果更改 k 的值，效果可以上升。最后发现，data1 的最佳选择是 k=29、data2 是 k=31、data3 和 data4 是 k=23。从而使得计算结果在前半学期的基础上有所提升。
- 后半学期添加的巨大测资 data5 完全无法运行，我的电脑的内存大小甚至无法支持构造完 de Bruijn 图。考虑到还需要时间来优化 data1~data4，遂放弃 data5。

运行方法

首先将数据文件夹直接放在根目录（./DATA1），longfix.py 和 join.py 放到数据文件夹下，运行 longfix.py 和 join.py 洗数据（先设置一下这两个源码里的相关文件名常量）。这里的 longfix.py 需要输入[L, R]表示当前运行需要求出的下标范围，用于支持多线程。(以下以双线程为例)

```
python3 ./longfix.py 0 250
python3 ./longfix.py 250 500
python3 ./join.py
```

预处理完成后，在根目录下运行：

```
g++ main.cpp -o main -DDATA1
./main
```

评测结果

排名	昵称	提交时间	提交次数	Genome_Fraction(%)	Duplication ratio	NGA50	Misassemblies	Mismatches per 100kbp
3	人民群众	2020/07/01 11:02:00am	31	99.838	6.8152	9983.8	10.0	0
8	人民群众	2020/07/01 11:02:54am	32	99.942	8.0352	9409.2	10.0	0
2	人民群众	2020/07/03 12:46:43pm	29	98.176	8.0	9817.6	0.0	0
3	人民群众	2020/07/02 12:00:33pm	22	99.19	7.9592	63161.4	70.0	0

参考资料

[1] https://en.wikipedia.org/wiki/De_novo_sequence_assemblers

[2] https://github.com/zhangtianxiang/genome_assembler

[3] https://github.com/zychn/genome-assembly

图片来源：

https://en.wikipedia.org/wiki/De_Bruijn_graph

https://www.researchgate.net/publication/264113418_Reference-free_SNP_detection_dealing_with_the_data_deluge

http://debruijnsequence.org/db/graph

https://github.com/zhangtianxiang/genome_assembler