“算法设计与分析”期末项目
宏基因组组装

张天翔
16307130026

2019年6月17日

目 录

1 问题描述与分析

1.1 问题描述

将设计一个宏基因组拼接的算法，输入数据为包含多个物种的测序数据（包含pacbio长序列和illumina短序列），可以使用所有数据进行组装，也可以仅使用其中的部分。输出结果为拼接后得到的长片段。衡量拼接结果好坏的依据主要为长度和准确率。

1.2 问题分析

这个问题属于De-novo组装问题：在没有参考序列的情况下，仅使用序列片段所提供的资讯来组装。其中De-novo组装又有多种算法，如Overlap - Layout - Consensus（OLC）法，De-Bruijn（DBG）法和贪婪算法（见维基百科 https://en.wikipedia.org/wiki/Sequence_assembly）。

通过对这些算法进行了解后得知各个算法的特性。OLC法计算序列的重叠是十分精确的，使用了完整的序列信息，结果比较精确，但是运算复杂度较大。而DBG使用了将序列再划分成长度为k的子段并建图的方法，损失了完整序列的信息，但是运算的时间复杂度较低，适用于序列较多的情况。

结合以上两种算法以及本次项目中的数据规模与特性，我决定使用类似OLC的方法，并使用DBG法对结果进行补全。具体算法见下文。

2 解决方法

2.1 步骤一：短序列匹配到长序列

观察data1至data4，可以发现短序列的正确率要高于长序列。如果直接长序列互相匹配，由于长序列很长且错误率过大，可以预料到结果会很差。而短序列较短，长度是长序列的十分之一（100/1000），所以应该存在很多短序列匹配在长序列上，下面进行一些尝试，看看能否发现一些规律。

任意取data3中一条长序列，枚举短序列与长序列进行匹配，找到编辑距离（Levenshtein distance）最小的位置，对距离取一个阈值进行筛选，得到的短序列按照匹配位置排序，再输出，结果如下图：

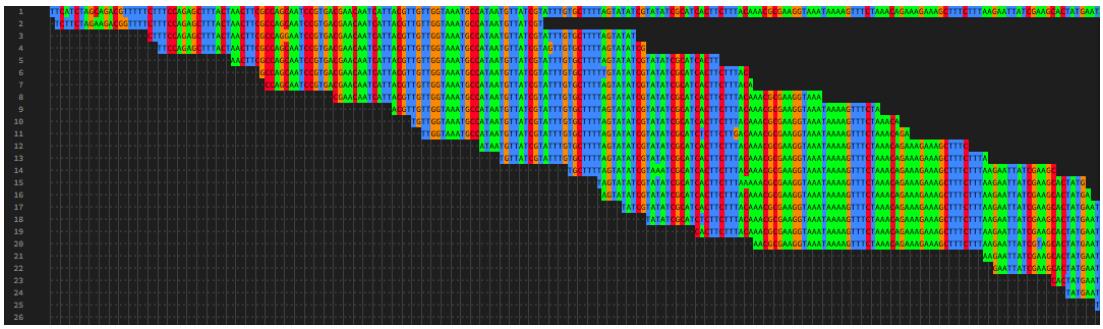


图1

可以发现短序列的覆盖较为均匀，且长序列明显错误较多，短序列中偶尔出现错误，因此长序列这个位置到底是什么可以由匹配了这个位置的所有短序列投票决定，达到对长序列进行修复的目的。同时，通过对匹配结果的观察也可以看出，长序列中基本没有DNA 的增减，只有错误，因此可以将求编辑距离改为求汉明距离（Hamming distance）来表示短序列对长序列的匹配情况。

总之，求短序列对长序列的匹配情况是有意义的。首先进行这一步操作，将所有短序列对长序列的匹配信息求出来。

具体做法为，对于一条长序列和一条短序列，枚举短序列在长序列上的起始位置，求汉明距离，找到距离最小的位置并对距离设置阈值为（长序列错误率+短序列错误率）×短序列长度+常数，常数暂定为5。时间复杂度为，长序列个数×短序列个数×长序列长度×短序列长度。对于最大的规模的数据data4 来说，长序列个数5000×2 （反向互补），短序列个数25000×2×2 （两端及反向互补），复杂度10^{14} ，因为长序列反向互补序列可以由原始方向对匹配结果取反向互补得到，可以减少一般长序列的个数，复杂度$5 \cdot 10^{13}$，按照c++使用CPU 运算一秒运算10^8 的标准来讲，需要139 小时，大概6天，开多进程并行运算，最终两天就得到了所有的匹配信息。data1 至data3 只用了几个小时就得到了匹配信息。这里也可以得知使用汉明距离的优点，汉明距离是$O(n)$ 的，而编辑距离是$O(n^2)$ 的DP，时间复杂度将大大增加。

算法细节见代码**PerfectMatchesAllMulti.py**，长序列被修补的结果存储至fixed_long_PREDONE_TOTALDONE.fasta ，短序列匹配到长序列的匹配信息存储在matches_PREDONE_TOTALDONE.json。如果分为多个部分求匹配信息，可以通过**Stitch.py**拼接结果存储至fixed_long.fasta 和matches.json。需要说明的是matches.json 本身格式不是正确的json 格式，将其转为json 格式需要删除最末尾的逗号，再在整个文件首尾加上左中括号和右中括号。这样就可以通过json 将其转为一个list。

前文提到，长序列的反向互补可以由原始方向的匹配结果求出，这个操作由**ExtendFixedLong.py**实现，它读入fixed_long.fasta和matches.json文件，将其扩展后存储到fixed_long_extend.fasta 和matches_extend.json中。

2.2 步骤二：根据匹配信息进行长序列组装

已经有了短序列与长序列的匹配信息，接下来多次进行尝试如何更好地利用这些信息。

2.2.1 尝试一：长序列根据连边数量进行组装

两个长序列如何拼接到一起，拼接的位置是什么？如果直接修补后的长序列两两之间暴力匹配，复杂度$(5000 \cdot 2)^2 \cdot 1000^2 = 10^{14}$，这比短序列匹配长序列还要耗费时间。为了利用匹配信息，我采用的方法是，对长序列与短序列编号，所有的长序列，将匹配它的短序列按照短序列编号排序，这样通过两个指针就能快速找到相同标号的短序列。短序列B匹配到长序列A1和A2各有一个位置pos1和pos2，则长序列A2对A1的位移为pos1-pos2，对这个位移求众数信息，便知道了A2对A1最可能的拼接位置。如此得到了所有的可能的边（最多$(5000 \cdot 2)^2 = 10^8$）实际上很可能不存在匹配相同的短序列而边数不多。

经过上面步骤得到了长序列之间的边，及其权重（众数的大小），下面就是直接对长序列进行拼接，我使用了类似Kruskal的算法，从边权较大的边开始选择，将长序列拼在一起。如果两个长序列在同一个集合中，则忽略这条边，否则进行拼接。

其中有个需要解决的问题：拼接后长序列与它属于的集合之中的结果序列的位移是多少？这需要改进并查集算法。我对并查集的修改是，维护长序列与其父亲之间的位移，则它到代表它所在集合的点的位移可通过累积父亲链上的位移。并查集寻找父亲时仍然进行路径压缩，同时维护点到父亲位移。具体看代码AssembleV1.py中的get_fa_and_update_off函数。

除了处理位移问题，我还增加了对两个序列是否可以拼接的判断，如果重叠部分错误率达到一定阈值，则不进行拼接，见代码AssembleV1.py中的try_merge函数。

全部代码见**AssembleV1.py**。

本次尝试的结果为：

dataset	Genome Fraction (%)	Duplication Ratio	NGA50	Mis-Assemblies	Mismatches Per 100kbp
data1	97.056	1.9572	9028.8	2.0	0
data2	87.198	1.6496	8067.8	0.0	0
data3	97.33	1.8888	8939.8	0.0	0
data4	/	/	0	/	/

其中data4效果太差NGA50为0，因为data4跑出来最长串只有三万长度左右。data1到data3的效果也并不是很好，应当尝试改进算法。

2.2.2 尝试二：长序列通过短序列扩展

由尝试一的结果可以看出尝试一有很大问题。对尝试一分析得到一种可能的错误：两个长序列进行合并（try_merge）时，由于只有两个序列进行合并，它们之间的错误没有办法进行投票，因为只有两个序列，所以我选择的方法是直接选取一条序列做结果。而不断拼接下去这个错误率会累积起来从而使得拼接时错误率高到阈值阻止拼接。

通过以上分析也可以得知，短序列匹配到长序列上的信息非常准确，可以信任这些信息，并且认为通过短序列联系起来的所有长序列都在最终结果序列上。

基于以上分析我设计了新的算法，枚举每一条未被访问过的长序列，维护一个长序列的集合，同时维护短序列的一个小根堆，短序列通过长序列被加入堆中且只被加入一次，堆按照短序列对长序列匹配的汉明距离排序。当堆不为空时，取出汉明距离最小的短序列，枚举短序列匹配的所有长序列，将所有未访问的长序列加入到长序列的集合中，同时把每个被加入的长序列匹配的所有未访问的短序列放入到小根堆中。如此一直进行下去就能将依靠匹配信息互相联系起来的长短序列求出。其中，长序列之间的位移计算方法与尝试一类似，这里不再赘述。最终，结果序列由长序列集合投票得到结果。

全部代码见**AssembleV2.py**。

dataset	Genome Fraction (%)	Duplication Ratio	NGA50	Mis-Assemblies	Mismatches Per 100kbp
data1	97.056	2.0002	9120.0	2.0	0
data2	97.056	2.0002	9120.0	2.0	0
data3	97.33	2.0	9733.0	0.0	0

data4	76.9518	1.978	59674.0	1.0	0

可以看到data3效果较好，data4有了效果，data1与data2稍微变好了一些，但还是效果太差。

2.2.3 尝试三：修复匹配信息

观察尝试二的输出长度发现，只有8个序列，其中两两相等，所以一共只输出了4个等价序列。但是为什么NGA50还能高于9000呢？这说明参考序列中存在两个序列非常的相似，相似程度可以达到90% 左右。例如，data1中短序列错误率为0，可是匹配信息却出现了不同：

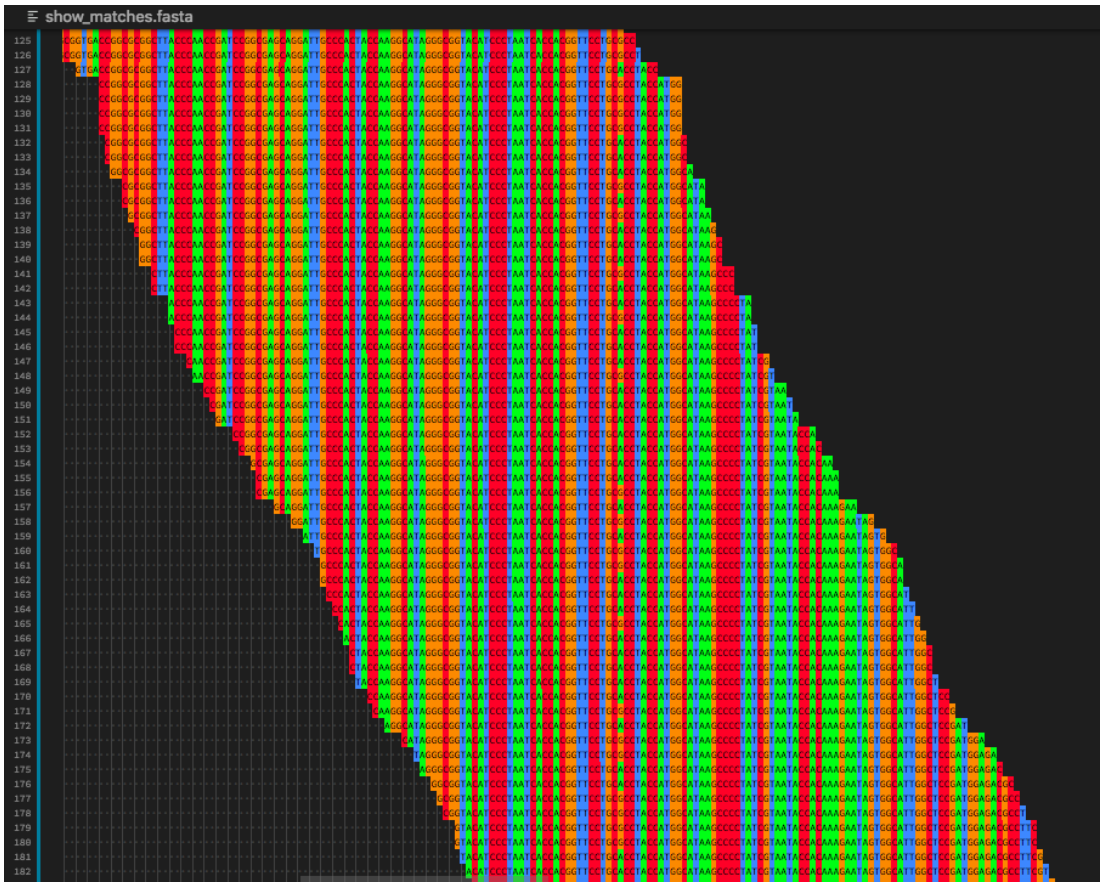


图2

因此，存在非常相似的短序列，短序列错误率为0告诉我们其中一部分短序列不应该匹配这条长序列。此时不应该通过投票来决定长串此位置是什么,而是让较少的部分不来匹配这条长序列。

同时，我在**AssembleV3.py**中进行了一些尝试，发现改进效果仍然不好。因此着手于对匹配信息进行修正。首先对长序列每一个位置求短序列给出的所有可能情况，如果众数的个数不能达到一定的阈值，说明存在相似的长序列，有很大一部分短序列不应该匹配当前长序列。我的做法是将第二多的结果的所有短序列删去。但是直接从前往后删会导致前面的结果影响后面的判断。我的做法将众

数所占比例最低的位置，且所占比例低于一定阈值的位置进行删除操作，不断循环进行，直到所有位置众数所占比例都高于阈值。阈值暂定为短序列错误率的两倍。之后再进行尝试二的匹配。

全部代码见**MatchesFix.py**。这个尝试只针对data1和data2。

dataset	Genome Fraction (%)	Duplication Ratio	NGA50	Mis-Assemblies	Mismatches Per 100kbp
data1	97.062	2.0	9120.6	2.0	0
data2	99.414	2.0	9361.8	2.0	0

data1只有略微的改进效果，data2改进的效果较好。

2.3 步骤三：以DBG法修补结果两侧

在步骤一求匹配信息的时候，因为长短序列都有一定的错误率，没有进行短序列对长序列两端的匹配计算，然而有可能长序列没有覆盖到基因的两端，短序列却覆盖到了。所以以短序列连同以短序列修补后的长序列建DB图，并尝试对答案序列的两端进行DB图上的搜索。

为答案序列两端枚举一定的长度进行扩展，选择一个扩展的最长的结果拼接到原序列上。

然而对于data4来说，DB图中创建四百万左右的点，python会报段错误，因此使用python将图的信息输出，要搜索路径的起点列表输出，让c++进行计算，再读取结果。

全部代码见**DBGCompleteV2.py**，**DBGcpp.cpp**。

最终效果如下：

dataset	Genome Fraction (%)	Duplication Ratio	NGA50	Mis-Assemblies	Mismatches Per 100kbp
data1	99.886	2.0	9403.0	2.0	0
data2	99.942	2.0	9410.2	2.0	0
data3	98.176	2.0	9817.6	0.0	0
data4	88.8782	3.265	70481.6	17.0	0

可以看到data1，data4改进较多，data2，data3有所改进。

其中data4进行DB图扩展时，可能达到最大深度，当最大深度逐渐增加时，Genome Fraction，Duplication Ratio，NGA50，Mis-Assemblies全部都在增加，这说明了DB图法虽然可以拼接出更长的结果，但不是很稳定，可能出现非常多的拼接错误。而不采用DB图则Genome Fraction就会变低。经过多次尝试，我的

算法经过DB图法扩展后，在data4能达到的NGA50最高为70481.6，Genome Fraction为88.8782。

3 最终效果

最终效果即为步骤三的结果：

dataset	Genome Fraction (%)	Duplication Ratio	NGA50	Mis-Assemblies	Mismatches Per 100kbp
data1	99.886	2.0	9403.0	2.0	0
data2	99.942	2.0	9410.2	2.0	0
data3	98.176	2.0	9817.6	0.0	0
data4	88.8782	3.265	70481.6	17.0	0

截至19-6-16晚23:00，data1至data3结果截图如下：

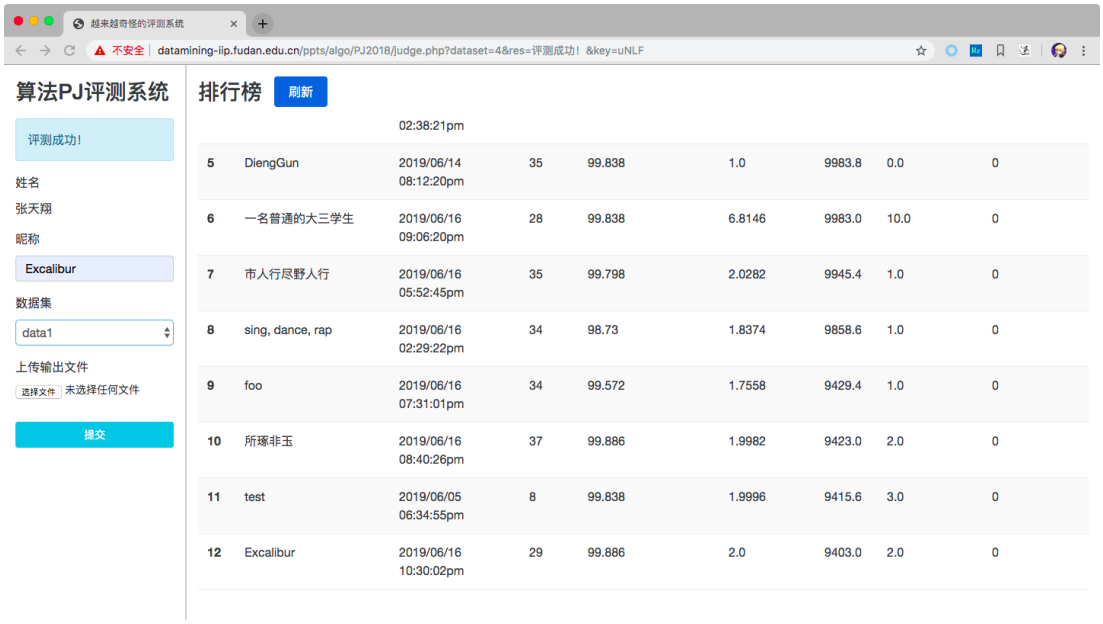


图3（data1）

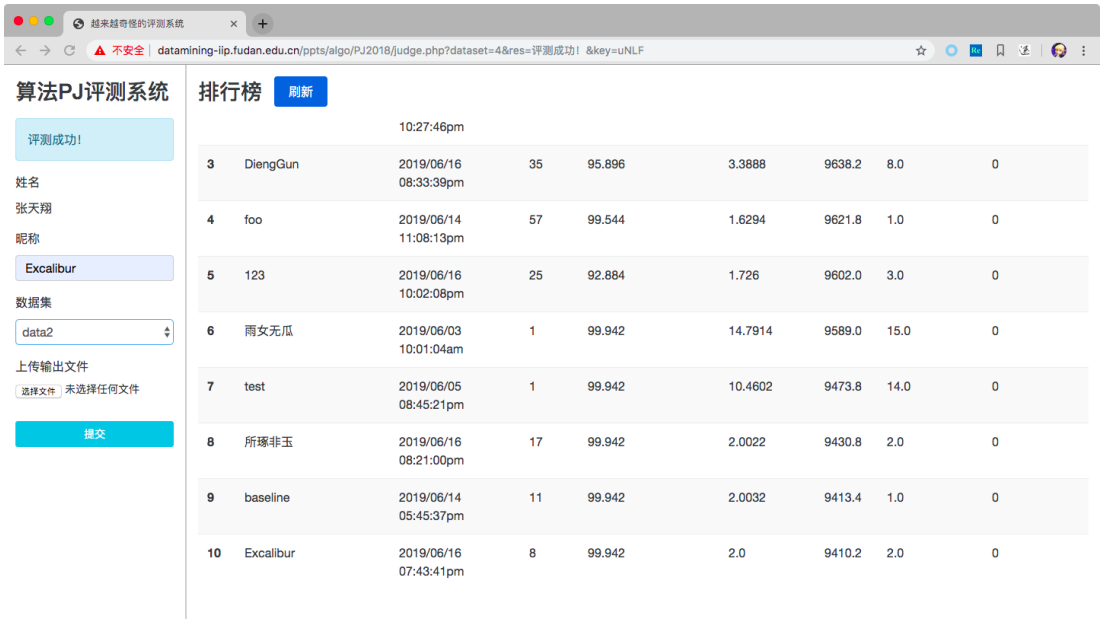
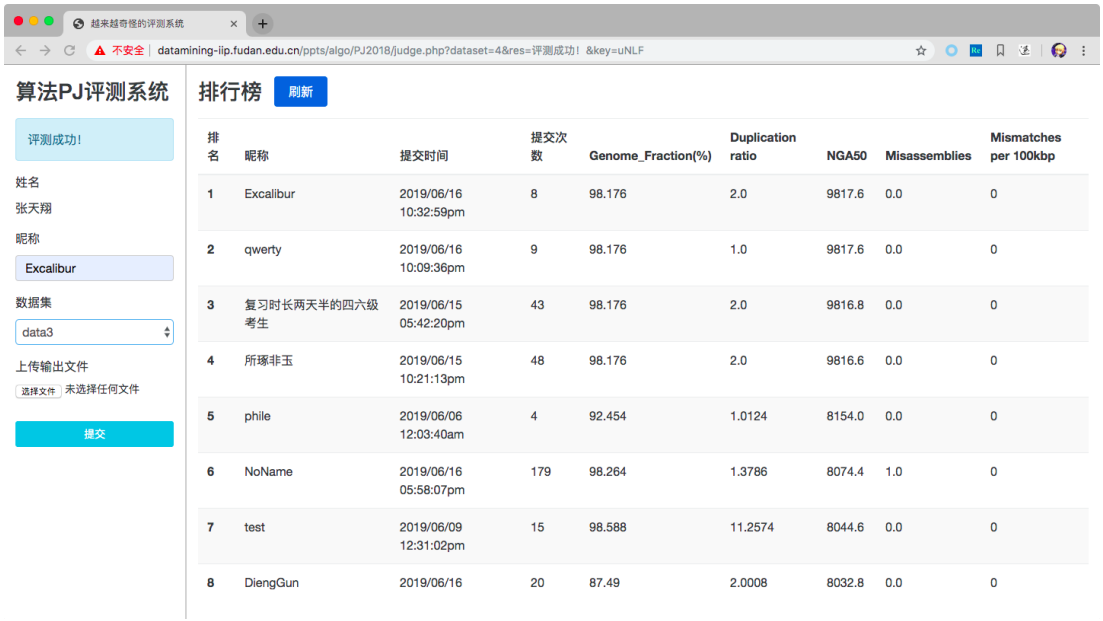


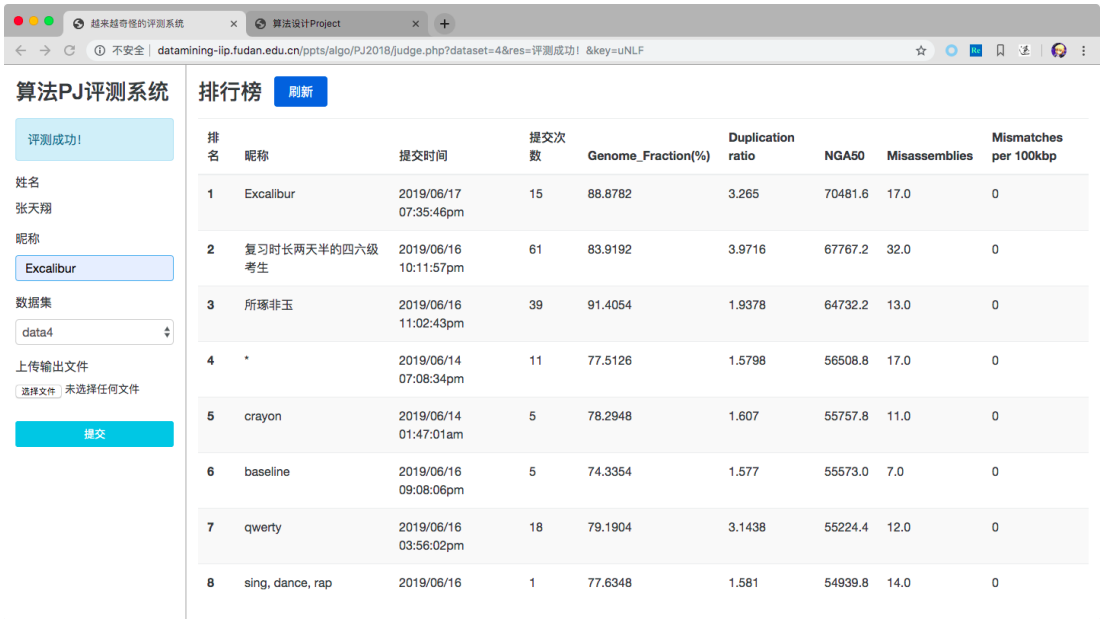
图4（data2）



排名	昵称	提交时间	提交次数	Genome_Fraction(%)	Duplication ratio	NGA50	Misassemblies	Mismatches per 100kbp
1	Excalibur	2019/06/16 10:32:59pm	8	98.176	2.0	9817.6	0.0	0
2	qwerty	2019/06/16 10:09:36pm	9	98.176	1.0	9817.6	0.0	0
3	复习时长两天半的四六级考生	2019/06/15 05:42:20pm	43	98.176	2.0	9816.8	0.0	0
4	所琢非玉	2019/06/15 10:21:13pm	48	98.176	2.0	9816.6	0.0	0
5	phile	2019/06/06 12:03:40am	4	92.454	1.0124	8154.0	0.0	0
6	NoName	2019/06/16 05:58:07pm	179	98.264	1.3786	8074.4	1.0	0
7	test	2019/06/09 12:31:02pm	15	98.588	11.2574	8044.6	0.0	0
8	DiengGun	2019/06/16	20	87.49	2.0008	8032.8	0.0	0

图5（data3）

截至19-6-17晚20:00，data4结果截图如下：



排名	昵称	提交时间	提交次数	Genome_Fraction(%)	Duplication ratio	NGA50	Misassemblies	Mismatches per 100kbp
1	Excalibur	2019/06/17 07:35:46pm	15	88.8782	3.265	70481.6	17.0	0
2	复习时长两天半的四六级考生	2019/06/16 10:11:57pm	61	83.9192	3.9716	67767.2	32.0	0
3	所琢非玉	2019/06/16 11:02:43pm	39	91.4054	1.9378	64732.2	13.0	0
4	*	2019/06/14 07:08:34pm	11	77.5126	1.5798	56508.8	17.0	0
5	crayon	2019/06/14 01:47:01am	5	78.2948	1.607	55757.8	11.0	0
6	baseline	2019/06/16 09:08:06pm	5	74.3354	1.577	55573.0	7.0	0
7	qwerty	2019/06/16 03:56:02pm	18	79.1904	3.1438	55224.4	12.0	0
8	sing, dance, rap	2019/06/16	1	77.6348	1.581	54939.8	14.0	0

图6（data4）

4 结果分析

data1与data2的效果并不是最好的，data3与data4的效果较好。

data1至data3中Duplication Ratio指标均为2.0，这是因为原始方向串与反向互补串同时存在，可以通过肉眼配对，手动操作将其变为1.0，或者因为答案串个数很少可以$O(n^2)$尝试配对来去除重复，但是必要性不大就没有进行这个操作。data4 在尝试二跑出59674.0 结果时Duplication Ratio只有1.978，然而加上DB图的扩展后剧增至3.265。观察DB扩展的输出信息，发现一些串扩展时，最长的结果传只扩展了一万三千左右，而其它较短的串扩展到了在DBGcpp 中给定的最大深度，根据我自己的算法猜测其原因可能是长序列较少，短序列较多，长序列进行匹配得到了一个个独立的集合，可能存在多个集合在同一个基因上却没有长序列相连，我的算法会把他们分别输出来。但是这些独立的集合可能通过短序列连起来，所以DB图法进行扩展后会导致很高的重复率。理论上也可以去重使的这项指标降低到一半，但是就较为复杂了。

Mis-Assemblies指标data1，data2较低，而data4在各种尝试中随着NGA50的增加到六万以上时就开始剧增，七万时到达17.0，而这是DB图扩展后的结果，这说明DB图也并不是十分的可靠。

Mismatches per 100kbp全部为0，没有参考价值。

Genome fraction data1 99.886%，data2 99.942% 与很多人相同，但是NGA50较低，虽然可以通过手动增加一些长度来增高NGA50，但是本着不手动操作结果的原则没有这么做。data3 98.176% 与NGA50的9817.6 完全一致是效果最好的。data4为88.8782% 不算太好。

5 结论

本文提出的使用长短序列匹配的信息进行长序列组装后再由DB图法进行结果的修补的算法，适用于本课程项目中规模较小的情况，且取得了较好的效果。