哈尔滨工业大学计算学部

读书/论文笔记

课程名称：生物信息学

课程类型：选修

项目名称：基因组序列片段比对 BWA

班级：2103601

学号：2021112845

姓名：张智雄

设计成绩	报告成绩	指导老师
		刘博

一、 论文的主要研究问题描述

这篇论文主要关注于解决新型 DNA 测序技术所带来的挑战，特别是 Illumina/Solexa 测序技术产生的大量短读取在对较大基因组（如人类基因组）进行高效准确映射时所面临的问题。这些短读取的数量通常在数百万至数亿之间，长度在 32 到 100 个碱基对之间，因此对现有的序列对齐程序提出了巨大挑战。

主要的研究问题包括如何开发快速准确的读取对齐程序，以应对大规模短读取的映射需求，尤其是针对较大基因组的映射。现有的序列对齐程序存在一些限制，例如在对长读取进行间隙对齐时的性能问题，以及对于大规模重新测序的速度和内存占用问题。

针对这些问题，本文提出了一种基于 Burrows-Wheeler Transform（BWT）的新型读取对齐软件包 BWA。该软件包利用后向搜索技术和 BWT，能够高效地对齐大型参考序列，如人类基因组，同时支持不匹配和间隙对齐。相较于现有的对齐程序（如 MAQ），BWA 在速度上表现更为优异，同时保持了类似的准确性。

本文的研究问题还包括对 BWT 和后向搜索技术的理论背景进行介绍，并介绍了 BWA 实现的精确匹配和不精确匹配算法。通过对模拟数据和真实数据的评估，论文探讨了 BWA 在不同情境下的性能表现，从而验证了其在高通量测序数据处理中的有效性和实用性。

二、 论文的主要方法

2.1 前缀 trie 和字符串匹配

字符串 X 的前缀 trie 是一棵树，其中每条边都标有一个符号，从叶子到根的路径上的边符号连接给出了 X 的一个唯一前缀。在前缀 trie 上，从一个节点到根的边符号连接给出了 X 的一个唯一子字符串，称为该节点表示的字符串。需要注意的是，X 的前缀 trie 与 X 的反向后缀 trie 相同，因此后缀 trie 理论也可以应用于前缀 trie。

利用前缀 trie，测试查询 W 是否是 X 的精确子字符串等同于找到表示 W 的节点，可以通过从根开始，将 W 中的每个符号与一条边匹配，在 O(|W|)时间内完成。要允许不匹配，我们可以穷举地遍历 trie，并将 W 与每条可能的路径进行匹配。稍后我们将展示如何通过使用 W 的前缀信息来加速这个搜索过程。figure 1 给出了“GOOGOL”前缀 trie 的示例。

2.2 Burrows-Wheeler 变换

设Σ为一个字母表。符号\$不存在于Σ中且按字典顺序小于Σ中的所有符号。字符串$X = a_0, a_1 \dots a_{n-1}$总是以符号\$结尾（即$a_{n-1} = \$$），并且此符号只出现在末尾。设$X[i] = a_i$，$i = 0,1, \dots, n-1$，为 X 的第i个符号，$X[i,j] = a_i \dots a_j$为子字符串，$X_i = X[i, n-1]$为 X 的后缀。X 的后缀数组 S 是整数 0...n−1 的一个排列，使得$S(i)$是第i个最小后缀的起始位置。X 的 BWT 定义为当$S(i) = 0$时$B[i] = \$$，否则$B[i] = X[S(i) - 1]$。我们还定义字符串 X 的长度为$|X|$，因此$|X| = |B| = n$。图 2 给出了构建 BWT 和后缀数组的示例。figure 2 中的算法在时间和空间上是二次的。在实践中，我们通常首先构建后缀数组，然后生成 BWT。

构建后缀数组的大多数算法至少需要 n⌈log2n⌉位的工作空间，这相当于人类基因组的12GB。最近，Hon 等人提出了一种新算法，使用 n 位的工作空间，仅在峰值时需要小于 1GB 的内存来构建人类基因组的 BWT。这个算法在 BWT-SW 中实现。我们改编了其源代码，使其能够与 BWA 一起使用。

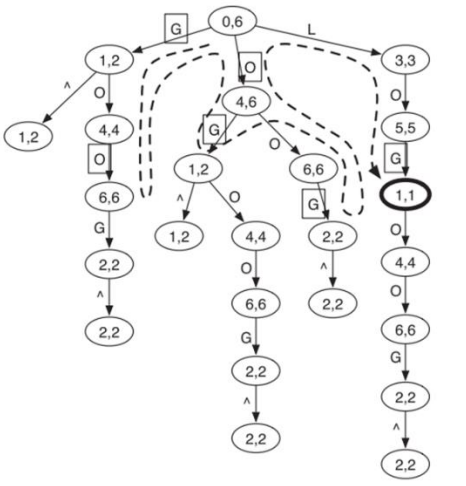

Fig. 1. Prefix trie of string 'GOOGOL'. Symbol ∧ marks the start of the string. The two numbers in a node give the SA interval of the string represented by the node (see Section 2.3). The dashed line shows the route of the brute-force search for a query string 'LOL', allowing at most one mismatch. Edge labels in squares mark the mismatches to the query in searching. The only hit is the bold node [1, 1] which represents string 'GOL'.

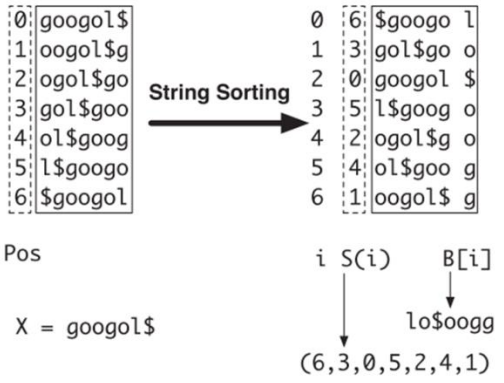

Fig. 2. Constructing suffix array and BWT string for $X = \texttt{googol\$}$. String X is circulated to generate seven strings, which are then lexicographically sorted. After sorting, the positions of the first symbols form the suffix array $(6, 3, 0, 5, 2, 4, 1)$ and the concatenation of the last symbols of the circulated strings gives the BWT string `lo$oogg`.

2.3 后缀数组区间和序列对齐

如果字符串W是X的子字符串，则W在X中的每个出现位置将出现在后缀数组中的一个区间内。这是因为所有具有W作为前缀的后缀都被排序在一起。基于这一观察，定义：

$$\underline{R}(W) = \min\{k: W\text{是}X_{S(k)\text{的前缀}}\} \quad (1)$$

$$\overline{R}(W) = \max\{k: W\text{是}X_{S(k)\text{的前缀}}\} \quad (2)$$

特别地，如果W是一个空字符串，则$\underline{R}(W) = 1$，$\overline{R}(W) = n - 1$。区间$\left[\underline{R}(W), \overline{R}(W)\right]$被称为$W$的$SA$区间，$X$中所有$W$的出现位置的集合是$\{S(k): \underline{R}(W) \le k \le \overline{R}(W)\}$。例如，在图 2 中，字符串'go'的 SA 区间是[1,2]。此区间中的后缀数组值为 3 和 0，给出了所有'go'的出现位置。知道了后缀数组中的区间，我们可以得到位置。因此，序列对齐等同于在X的子字符串中搜索与查询匹配的后缀数组区间。对于精确匹配问题，我们可以找到唯一的这样一个区间；对于不精确匹配问题，可能会有多个。

2.4 精确匹配：后向搜索

让$C(a)$表示在$X[0, n-2]$中按字典顺序小于$\alpha \in \Sigma$的符号的数量，$O(a, i)$表示$B[0, i]$中a的出现次数。Ferragina 和 Manzini 证明了如果W是X的一个子字符串：

$$\underline{R}(aW) = C(a) + O\left(a, \underline{R}(W) - 1\right) + 1 \quad (3)$$

$$\overline{R}(aW) = C(a) + O\left(a, \overline{R}(W)\right) \quad (4)$$

并且当且仅当aW是X的子字符串时，$\underline{R}(aW) \le \overline{R}(aW)$。这个结果使得我们可以在$O(|W|)$的时间内通过从$W$的末尾开始迭代计算$\underline{R}$和$\overline{R}$来测试$W$是否是$X$的子字符串，并计算$W$的出现次数。这个过程称为$backward$搜索。需要注意的是，方程（3）和（4）实际上实现了对X的前缀 trie 的自顶向下遍历，假设我们可以在已知其父节点的区间的情况下，以常数时间计算子节点的 SA 区间。在这个意义上，后向搜索等同于在前缀 trie 上进行精确字符串匹配，但不需要显式地将 trie 放入内存中。

```
Precalculation:
  Calculate BWT string B for reference string X
  Calculate array C(·) and O(·,·) from B
  Calculate BWT string B' for the reverse reference
  Calculate array O'(·,·) from B'
Procedures:
  InexactSearch(W, z)
    CalculateD(W)
    return InexRecur(W, |W|−1, z, 1, |X|−1)

  CalculateD(W)
    k ← 1
    l ← |X|−1
    z ← 0
    for i = 0 to |W|−1 do
      k ← C(W[i]) + O'(W[i], k−1) + 1
      l ← C(W[i]) + O'(W[i], l)
      if k > l then
        k ← 1
        l ← |X|−1
        z ← z+1
      D(i) ← z
```

Fig. 3 不精确搜索算法

```
InexRecur(W, i, z, k, l)
  if z < D(i) then
    return ∅
  if i < 0 then
    return {[k, l]}
  I ← ∅
  I ← I ∪ InexRecur(W, i−1, z−1, k, l)
  for each b ∈ {A, C, G, T} do
    k ← C(b) + O(b, k−1) + 1
    l ← C(b) + O(b, l)
    if k ≤ l then
      I ← I ∪ InexRecur(W, i, z−1, k, l)
      if b = W[i] then
        I ← I ∪ InexRecur(W, i−1, z, k, l)
      else
        I ← I ∪ InexRecur(W, i−1, z−1, k, l)
  return I
```

优化

```
CalculateD(W)
  z ← 0
  j ← 0
  for i = 0 to |W|−1 do
    if W[j, i] is not a substring of X then
      z ← z+1
      j ← i+1
    D(i) ← z
```

Fig. 4

2.5 不精确匹配：有界遍历/回溯

Figure 3 给出了一种递归算法，用于搜索与查询字符串 W 具有不超过 z 个差异（不匹配或间隙）的 X 的子字符串的 SA 区间。基本上，这个算法使用后向搜索从基因组中抽样不同的子字符串。这个过程受到 D(・)数组的约束，其中 D(i)是 W[0,i]中差异数量的下界。D 的估计越好，搜索空间就越小，算法就越高效。一个朴素的界限是将对所有 i 设置 D(i)=0，但由此得到的算法在差异数量上显然是指数级的，效率会较低。

图 3 中的 CalculateD 过程提供了一个更好的界限，虽然不是最优的。它在概念上等同于图 4 中描述的方法，更容易理解。作者使用反向（非互补）参考序列的 BWT 来测试 W 的子字符串是否也是 X 的子字符串。需要注意的是，如果仅使用 BWT 字符串 B 进行此测试，CalculateD 将成为一个$O(|W|^2)$的过程，而不是图 3 中描述的$O(|W|)$。

图 3 中的算法保证找到所有允许最大 z 个差异的区间。理论上它是完整的，但在实践中，作者也做了各种修改。

- **对不匹配、间隙开放和间隙延伸支付不同的惩罚**，更符合生物数据的实际情况。
- 使用**类似堆的数据结构来保持部分匹配**，以实现广度优先搜索（BFS），并处理反向互补的读取序列。递归模拟了前缀 trie 上的深度优先搜索（DFS）。
- 采用**迭代策略，优先搜索唯一的顶部区间**，并仅搜索具有最多 z+1 个差异的命中，

加速了 BWA，但速度对读取与参考之间的不匹配率敏感。

- 允许在读取的前几十个碱基对中**设置最大允许的差异限制**，称之为种子序列。在模拟中，种子对对齐效果显著，但对于较短的读取，种子效果较差。

2.6 减少内存

上述描述的算法虽然强大，但需要将发生数组 O 和后缀数组 S 加载到内存中，这可能会占用大量内存。为了减少内存使用量，我们不是存储完整的 O 和 S 数组，而是只存储其中的一小部分，其余的在运行时计算。发生数组$O(\cdot,\cdot)$通常需要$4n\lceil log2n\rceil$位，因为每个整数需要$\lceil log2n\rceil$位，数组中有 4n 个整数。实际上，只有$O(\cdot,k)$（其中k是 128 的因数）被存储在内存中，其余的使用 BWT 字符串 B 进行计算。当使用两位表示一个核苷酸时，B 占用 2n 位。因此，后向搜索的内存约为$2n+n\lceil log_2 n\rceil/32$位。考虑到需要存储反向基因组的 BWT，用于计算间隔的内存需求翻倍，对于一个 3 Gb 的基因组，约为 2.3 GB。

枚举每个出现位置需要后缀数组 S，如果完全存储在内存中，则需要$n\lceil log2n\rceil$位。然而，只要知道部分 S，就可以重建整个 S。后缀数组 S 和逆压缩后缀数组Ψ^{-1}满足一个关系，其中Ψ^{-1}可以通过发生数组 O 计算得到。在 BWA 中，只有可以被 32 整除的 k 值的 S(k)被存储在内存中。对于不能被 32 整除的 k，Ψ^{-1}被重复应用，直到找到某个 j 使得$(\Psi^{-1})^{(j)}(k)$是 32 的因数，然后可以查找$S\left((\Psi^{-1})^{(j)}(k)\right)$并使用方程(5)计算 S(k)。

对于小于 4 Gb 的基因组，对齐过程使用$4n+n\lceil \log_2 n\rceil/8$位，即 n 字节的内存。这包括了 BWT 字符串、原始和反向基因组的部分发生数组和部分后缀数组的内存。此外，还需要几百兆字节的内存用于堆、缓存和其他数据结构。通过这些内存优化技术，该算法在计算效率和内存使用之间取得了平衡，即使处理大规模的基因组数据集，也能实现强大的对齐能力。

2.7 其他 Illumina 读取的实际考虑因素

2.7.1 含疑难碱基的处理

针对读取中的非 A/C/G/T 碱基，算法将其视为不匹配。参考基因组中的非 A/C/G/T 碱基被转换为随机核苷酸，以避免在含有疑难碱基的区域产生假阳性结果。

2.7.2 配对端对齐

BWA 支持配对端对齐，首先找到所有好的匹配位置，然后通过线性扫描进行配对。配对过程耗时，因为需要频繁查找后缀数组。为了加速配对，采用缓存大的间隔区域的策略，减少了配对所需的时间。

2.7.3 确定允许的最大差异数

根据读取长度，BWA 设置最大允许的差异数，以确保匹配的准确性和效率。差异数的设置考虑了读取长度和统一碱基错误率，使得不同长度的读取可以容忍不同数量的差异。

2.7.4 生成映射质量分数

对于每个对齐，BWA 计算映射质量分数，表示对齐不正确的概率。与 MAQ 类似，但 BWA 假设始终可以找到真实的命中，因此修正了 MAQ 公式的问题，以避免低估映射质量。

尽管可能因此而高估映射质量，但在实际模拟中，错误率相对较低，说明这种修改影响较小。

2.8 SOLiD 读取的映射

对于 SOLiD 读取，BWA 将参考基因组转换为二核苷酸“颜色”序列，并为颜色基因组构建 BWT 索引。读取在颜色空间中进行映射，其中序列的反向互补与其本身相同，因为颜色的互补是它本身。对于 SOLiD 配对端映射，如果以下两种情况之一为真，则认为读取对处于正确的方向：(i) 两端都映射到基因组的正链上，R3 读取的坐标较小；(ii) 两端都映射到基因组的反链上，F3 读取的坐标较小。另外，也会在颜色空间进行 Smith-Waterman 对准。

在对准后，BWA 使用动态规划将颜色读取序列解码为核苷酸序列。给定一个核苷酸参考子序列$b_1 b_2 \ldots b_{l+1}$和一个映射到该子序列的颜色读取序列$c_1 c_2 \ldots c_l$，BWA 推断出一个核苷酸序列$\overrightarrow{b_1}\,\overrightarrow{b_2} \ldots \overrightarrow{b_{l+1}}$，使得它最小化以下目标函数：

$$\sum_{i=1}^{l+1} q' \cdot (1 - \delta_{\hat{b}_i, b_i}) + \sum_{i=1}^{l} q_i \cdot \left[1 - \delta_{\hat{c}_i, g(\hat{b}_i, \hat{b}_{i+1})}\right]$$

其中，q'是突变的 Phred 标度概率，q_i是颜色c_i的 Phred 质量，函数$g(b, b') = g(b', b)$给出了两个相邻核苷酸\$b\$和b'对应的颜色。如果$b_i \neq \hat{b}_i$则支付惩罚q'，如果$c_i \neq g(\hat{b}_i, \hat{b}_{i+1})$则支付惩罚$q_i$。这种优化可以通过动态规划来完成，因为超出位置$i$的最佳解码只取决于$\hat{b}_i$的选择。设$f_i(\hat{b}_i)$是到位置$i$的最佳解码分数。迭代方程为：

$$f_1(\hat{b}_1) = q' \cdot (1 - \delta_{\hat{b}_1, b_1})$$

$$f_{i+1}(\hat{b}_{i+1}) = \min_{\hat{b}_i} \left\{ f_i(\hat{b}_i) + q' \cdot (1 - \delta_{b_{i+1}, \hat{b}_{i+1}}) + q_i \cdot \left[1 - \delta_{\hat{c}_i, g(\hat{b}_i, \hat{b}_{i+1})}\right] \right\}$$

BWA 将基本质量近似为如下。设$\hat{c}_i = g(\hat{b}_i, \widehat{b_{i+1}})$。第$i$个碱基质量$\hat{q}_i, i = 2 \ldots . l$,计算如下：

$$\hat{q}_i = \begin{cases} q_{i-1} + q_i & \text{if } c_{i-1} = \hat{c}_{i-1} \text{ and } c_i = \hat{c}_i \\ q_{i-1} - q_i & \text{if } c_{i-1} = \hat{c}_{i-1} \text{ but } c_i \neq \hat{c}_i \\ q_i - q_{i-1} & \text{if } c_i = \hat{c}_i \text{ but } c_{i-1} \neq \hat{c}_{i-1} \\ 0 & \text{otherwise} \end{cases}$$

最终，BWA 将序列$\widehat{b_2} \ldots \widehat{b_l}$和质量$\widehat{q_2} \ldots \widehat{q_l}$作为 SOLiD 映射的最终结果输出。

三、 论文的主要实验结果

该研究旨在评估四种短读取比对程序在模拟数据和真实数据上的性能表现。作者选取了 BWA、MAQ、SOAPv2 和 Bowtie 这四个代表性的比对程序进行评估。

3.1 模拟数据评估

在模拟数据评估中，使用了 wgsim 程序从人类基因组中模拟了一系列读取，并将其映射回基因组。结果显示，BWA 和 MAQ 在比对准确性方面表现相似，且比 Bowtie 和 SOAPv2 更准确。尤其是，BWA 和 MAQ 在自信映射读取的比例和自信映射的错误率方面都优于 Bowtie 和 SOAPv2。另外，SOAPv2 是最快的程序，但如果禁用带间隙的比对，其速度会更快。而 Bowtie 在默认设置下比较快，但以准确性为代价。在速度方面，BWA 比 MAQ 快 6–18 倍，取决于读取长度。然而，BWA 的速度提升是以准确性为代价的。值得注意的是，所有基于 BWT 的比对器的内存使用量不受比对读取数量影响，而 MAQ 的内存使用量与读取

数量成线性关系。

3.2 真实数据评估

在真实数据评估中，作者下载了从 European Read Archive 获取的 Illumina 读取，并将其映射回人类基因组。实验结果显示，几乎所有 BWA 和 MAQ 的自信比对都存在一致的配对，而 MAQ 给出的自信比对较少。另外， BWA 在慢速模式下能够获得更好的比对结果。对于真实数据的处理，SOAPv2 如果禁用了带间隙的比对会更快，而 BWA 则即使禁用带间隙的比对仍能够识别出许多短的插入/删除（indels）。在鸡基因组上的比对结果显示，BWA 的比对错误率约为 0.06%，在选择更高映射质量阈值的情况下，BWA 的自信比对百分比和映射到错误基因组的读取数量均优于 Bowtie。

Table 1. Evaluation on simulated data

	Single-end			Paired-end		
Program	Time (s)	Conf (%)	Err (%)	Time (s)	Conf (%)	Err (%)
Bowtie-32	1271	79.0	0.76	1391	85.7	0.57
BWA-32	823	80.6	0.30	1224	89.6	0.32
MAQ-32	19797	81.0	0.14	21589	87.2	0.07
SOAP2-32	256	78.6	1.16	1909	86.8	0.78
Bowtie-70	1726	86.3	0.20	1580	90.7	0.43
BWA-70	1599	90.7	0.12	1619	96.2	0.11
MAQ-70	17928	91.0	0.13	19046	94.6	0.05
SOAP2-70	317	90.3	0.39	708	94.5	0.34
bowtie-125	1966	88.0	0.07	1701	91.0	0.37
BWA-125	3021	93.0	0.05	3059	97.6	0.04
MAQ-125	17506	92.7	0.08	19388	96.3	0.02
SOAP2-125	555	91.5	0.17	1187	90.8	0.14

One million pairs of 32, 70 and 125 bp reads, respectively, were simulated from the human genome with 0.09% SNP mutation rate, 0.01% indel mutation rate and 2% uniform sequencing base error rate. The insert size of 32 bp reads is drawn from a normal distribution $N(170,25)$, and of 70 and 125 bp reads from $N(500,50)$. CPU time in seconds on a single core of a 2.5 GHz Xeon E5420 processor (Time), percent confidently mapped reads (Conf) and percent erroneous alignments out of confident mappings (Err) are shown in the table.

Table 2. Evaluation on real data

Program	Time (h)	Conf (%)	Paired (%)
Bowtie	5.2	84.4	96.3
BWA	4.0	88.9	98.8
MAQ	94.9	86.1	98.7
SOAP2	3.4	88.3	97.5

The 12.2 million read pairs were mapped to the human genome. CPU time in hours on a single core of a 2.5 GHz Xeon E5420 processor (Time), percent confidently mapped reads (Conf) and percent confident mappings with the mates mapped in the correct orientation and within 300 bp (Paired), are shown in the table.

研究结果表明，在模拟数据和真实数据上，BWA 在比对准确性和速度方面表现出色，尤其是在处理真实数据时。此外，BWA 生成的映射质量允许用户根据需要选择更高准确度的比对结果，这对于后续的生物信息学分析非常有用。因此，作者推荐研究人员在短读取比对任务中优先考虑使用 BWA 程序。

四、 论文方法的优缺点分析

对于人类参考基因组的短读取比对任务，**BWA 比 MAQ 快一个数量级，同时达到了相似的比对准确性。**它支持对单端读取进行带间隙的比对，这在读取变长且倾向于包含插入/删除（indels）的情况下变得越来越重要。BWA 输出的比对结果采用 SAM 格式，以利用 SAMtools 中实现的下游分析功能。BWA 与 SAMtools 结合提供了 MAQ 软件包的大部分功能，并具有额外的功能。

与速度、内存和映射读取数量相比，比对准确性在真实数据上要难评估得多，因为我们无法获知真实情况。在本文中，我们使用了三个评估比对准确性的标准。第一个标准，只能通过模拟数据评估，是自信映射数量和自信映射中的比对错误率的组合。需要注意的是，仅

自信映射数量可能不是一个好的标准：我们可以以准确性为代价获得更多的映射。第二个标准，是对实际数据进行评估的，即对齐读取数量和以一致方式映射的读取对数量的组合，基于实验中的配对信息正确且结构变异较少的假设。尽管这个标准与比对器如何定义“一致性”配对相关，但根据我们的经验，如果设置了正确的配对参数，这个标准是非常信息丰富的。第三个标准是，如果我们有意向参考基因组添加来自不同物种的参考序列，则映射到错误参考序列的读取分数。

尽管从理论上讲，BWA 可以处理任意长的读取，但其在长读取上的性能受到影响，特别是当测序错误率较高时。此外，BWA 总是需要对整个读取进行比对，从第一个碱基到最后一个碱基（即相对于读取的全局），但长读取更有可能受到结构变异或参考基因组的错误组装的干扰，这将导致 BWA 失败。对于长读取，一个可能更好的解决方案是将读取分成多个短片段，分别使用上述算法进行比对，然后将部分比对连接起来，以获取读取的完整比对结果。