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：生物序列表示 deBWT

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

随着高通量测序技术的快速发展和广泛应用，许多基因组已经在尖端基因组学研究中被测序。例如，千人基因组计划（The 1000 Genomes Project Consortium, 2015）和 UK10K 项目（The UK10K Consortium, 2015）已经测序了数千个人类基因组。此外，随着测序成本持续降低，未来基因组数量可能会急剧增加。在这种情况下，对大量基因组进行良好的组织和索引是为了促进未来的基因组学研究至关重要。

Burrows-Wheeler Transform（BWT）是一种自索引数据结构，具有许多基本应用，例如基因组索引、序列比对、基因组压缩、基因组组装和测序错误校正。然而，基因组序列的 BWT 构建是一项非平凡的任务。主要问题在于 BWT 构建的核心是确定输入序列的所有后缀的词典顺序。由于基因组中可能存在许多重复的序列，直接比较所有后缀以确定它们的词典顺序的成本将是非常高的。对于构建许多高度相似的基因组的 BWT，例如大量个体的人类基因组，这个问题甚至更为严重，因为会有许多共同的序列使得整个输入更加重复。

为了**解决传统的 BWT 构建方法在处理大规模基因组数据时面临着计算和内存资源的挑战**，本文提出了一种名为 de Bruijn branch-based BWT constructor (deBWT) 的新型并行 BWT 构建方法。该方法充分利用了 de Bruijn 图 (dBG) 在表示基因组序列中 k-mer 的分布上的优势，并将基因组序列的后缀表示和组织成一种新的数据结构，即 de Bruijn branch encoding。这种新的数据结构不仅提高了对具有长公共前缀的后缀的处理效率，还有效减少了内存占用和计算成本。

另外，deBWT 还通过将 BWT 分块，并利用 dBG 的拓扑结构，避免了对所有块进行不必要的排序，从而提高了算法的效率和可扩展性。这种分块和拓扑结构的利用使得 deBWT 在处理大规模基因组数据时具有更好的性能表现，并且能够充分利用现代服务器和集群的多核心和大内存资源。

对 deBWT 进行了各种数据集的基准测试，结果表明它具有快速的速度和良好的多线程可扩展性。特别是，deBWT 非常适合于构建高度相似的基因组序列集合的 BWT，例如多个人类基因组

二、 论文的主要方法

2.1 符号设置

假设 DNA 序列G是一个字符序列，其中包含G个字符，字符集为$R = \{A, C, G, T\}$。为了索引，假设序列$S = G\$$，这里\$是一个辅助字符，且S的字符字典序为$A < C < G < T < \$$。S_i表示S的第i个字符，$S_{i\ldots j}$表示从S_i开始到S_j结束的子串。S的一个后缀是$S_{i\ldots|G|}$，其中$|\ G\ |$表示G的长度。SA表示S的后缀数组，即SA_i表示第i个字典序最小的后缀的起始位置。BWT（Burrows-Wheeler Transform）是S的一个排列，其中B_{S_i}是S_{SA_i-1}的前一个字符，如果SA_i不等于 0，则B_{S_i}是辅助字符\$。

2.2 基于 Unipath 的 BWT 构建

当我们讨论基于 Unipath 的 BWT 构建时，我们实际上是利用了 dBG（分布式 Bloom 图）的概念。这种方法可以将 DNA 序列表示为一系列相邻的 k-mer，通过这些 k-mer 之间的连接关系来重建原始序列。下面详细介绍这个过程：

2.2.1 dBG（分布式 Bloom 图）的构建：

首先，我们将输入的 DNA 序列G切分成大小为k的k-mer，其中k是预先确定的参数。对于每个k-mer，我们在 dBG 中创建一个节点。这些节点构成了 dBG 的顶点集合。

然后，我们根据 k-mer 之间的重叠关系在 dBG 中添加边。如果两个k-mer 有$k-1$ 个相同的字符，并且它们在原始序列中是相邻的，那么在 dBG 中它们之间就有一条边。这个过程类似于将 DNA 序列视为一个网络，划分成连续的片段作为其中的节点，边表示这些片段之间的相似性和联系，为后续的 Unipath 构建提供了基础。这个过程不仅帮助我们

这样，dBG 就表示了 DNA 序列G中的所有k-mer 以及它们之间的连接关系。

2.2.2 Unipath 的构建：

在构建了 dBG 之后，我们可以从 dBG 的顶点序列中获取 Unipath。Unipath 是由 dBG 的顶点序列$KMG_0, KMG_1, \dots, KMG_{G-k}$组成的有序列表，其中$KMG_i$是 DNA 序列$G$中位置$i$的 k-mer。Unipath 的构建过程实际上是对 DNA 序列的一种精细化表示，能够捕捉到序列中的重复结构和潜在的功能区域，为后续的后缀比较和 BWT 构建奠定了基础。

此外，Unipath 的构建还具有一定的灵活性，可以根据需要调整k-mer 的大小，从而在保留序列结构信息的同时，适应不同的分析需求和数据特点。

通过顺序遍历这些顶点，我们可以获得 DNA 序列的 Unipath 表示。

2.2.3 后缀比较：

一旦我们有了 DNA 序列的 Unipath 表示，就可以利用它来进行后缀比较。后缀比较是确定 BWT（Burrows-Wheeler Transform）中字符顺序的关键步骤之一。当我们比较两个后缀$Suf_S = S_{i\dots G}$和$Suf_{S'} = S_{j\dots G}$时，

- 如果它们的起始k-mer KMG_i和KMG_j不同，那么它们的字典序就可以直接确定。
- 如果$KMG_i = KMG_j$，则需要进一步比较它们对应的 Unipath 以确定它们的顺序。

这个过程实际上是在利用序列的局部相似性和全局结构信息来确定字符顺序，为序列比对和功能分析提供了重要的基础。

2.3 deBWT 方法

deBWT 方法将 BWT 的构建分为三个主要阶段，每个阶段的过程如下：

2.3.1 dBG 构建和分析：

1. **k-mer 提取和图构建：**首先，从输入的 DNA 序列中提取长度为k的k-mer，并使用这些k-mer 构建分布式 Bloom 图（dBG）。这个过程包括将每个k-mer 作为图的节点，并在图中添加表示k-mer 之间连接关系的边。

2. **Unipath 识别：**在构建完整的 dBG 之后，需要识别其中的 unipath。Unipath 是一条在 dBG 上的路径，它表示了 DNA 序列的一种有序的组织方式。识别 unipath 可以帮助我们

理解 DNA 序列的结构和特征。

3. **多出和多入顶点识别：**除了识别 unipath 之外，还需要识别 dBG 中的多出和多入顶点。多出顶点表示在 dBG 中有多个后续节点的顶点，而多入顶点表示在 dBG 中有多个前驱节点的顶点。识别这些顶点对于后续的 BWT 构建至关重要。

2.3.2 de Bruijn 分支编码生成：

这个编码的生成是基于 dBG 的结构，可以高效地反映出 DNA 序列的拓扑特征，主要分为两个步骤。

1. **哈希表构建：**为了有效地管理多出和多入顶点，需要构建一个哈希表数据结构来索引这些顶点。

2. **de Bruijn 分支编码生成：**通过扫描 dBG 图，可以生成 de Bruijn 分支编码。这个编码用于表示 DNA 序列中的 k-mer 以及它们之间的连接关系。

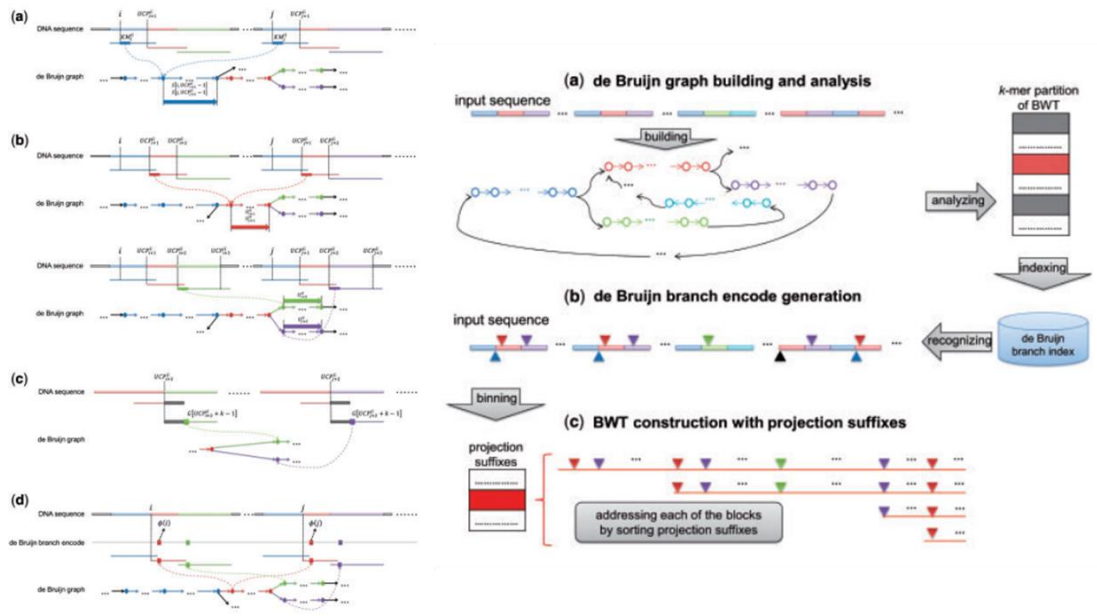

图 1 方法流程图

（左为具有相同初始 k-mer 的两个后缀之间的单路径比较，右为 deBWT 方法的示意图）

2.3.3 使用投影后缀构建 BWT：

1. **投影后缀生成：**对于每个多入顶点，需要生成其对应 BWT 部分的投影后缀。投影后缀是从 DNA 序列中某个位置开始的后缀，并且根据 de Bruijn 编码进行了投影。

2. **SA 构建：**对于每个 BWT 部分，需要构建其对应的后缀数组（SA）。后缀数组表示了投影后缀在 BWT 中的顺序。

3. **快速排序：**使用快速排序算法对投影后缀进行排序，以便构建 SA。由于 SA 的构建是 BWT 构建的关键步骤，因此这个过程需要高效且可靠的算法来实现。

4. **并行处理：**由于 BWT 的构建是一个计算密集型任务，可以通过并行处理来提高效率。在这个阶段，可以将不同 BWT 部分的处理分配给多个处理单元并行执行。可以

- ***k*-mer 计数：**可以并行执行，使用 Jellyfish 工具。
- ***k*-mer 排序：**使用基数排序(radix sort)并行处理 *k*-mer 的分区和排序。
- **多入多出顶点识别：**通过对排序后的 *k*-mer 列表进行归并操作来识别，虽然这一步骤没有并行化，但是其时间复杂度是线性的。

2.3.4 额外处理

而对于那些起始位置在#或$之前$k$个位置以内的后缀，它们被单独设置并排序以填充 BWT 字符串。这个额外处理确保了 BWT 的完整性和正确性，同时也简化了后续 BWT 的使用和查询。

三、 论文的主要实验结果

3.1 数据集

我们使用三个数据集对 deBWT 进行了基准测试，模拟了不同的真实应用场景。

(i) 一个数据集包含了 10 个体外人类基因组（总共 30.9 亿碱基对）。每个基因组是通过将来自 1000 基因组计划（1000 Genomes Project Consortium, 2015）中特定样本的变异整合到人类参考基因组 GRCh37/Hg19 中而生成的。该数据集模拟了对多个个体人类基因组进行索引，这在基因组研究中有许多应用。

(ii) 一个数据集包含一组模拟的 contig。随着长读取测序技术的进步，如单分子实时测序，已将人类基因组组装的 contig N50 提高到> 1000 万 bp，我们从 10 个体外人类基因组中随机提取了 3000 个序列（每个序列约 10M bp 长，总共 30.2 亿碱基对），这是通过一个从 Wgsim 模拟器修改而来的自制脚本进行的。

(iii) 一个数据集包含八个灵长类基因组，包括长臂猿、大猩猩、猩猩、恒河猴、狒狒、黑猩猩、倭黑猩猩和人类。这个数据集评估了 deBWT 对于索引更多样化基因组的能力。

3.2 实验结果

在一台装有四个 Intel Xeon E4820 CPU（总共 32 个核心）的服务器上进行了基准测试，每个 CPU 的主频为 2.00 GHz，内存容量为 1 TB，运行 Linux Ubuntu 14.04 操作系统。DeBWT 使用了 Jellyfish（版本 2.1.4）来实现输入序列的 k-mer 计数（参数 k 配置为 31）。

为了进行比较，我们还对同样的数据集使用了两种最近发布的方法，RopeBWT2 和 ParaBWT（版本 1.0.8-binary-x86_64），结果如下：

3.2.1 时间比较

经过的时间（见表 1）表明，deBWT 和 ParaBWT 的速度相当，而 RopeBWT2 较慢，可能是因为它不支持并行计算，而且算法可能不适用于长序列。

3.2.2 各步骤时间成本

调查了 deBWT 各步骤的时间成本（见表 2）。主要观察到了两个问题。

首先，deBWT 的大多数核心步骤，即 k-mer 计数、k-mer 排序、de Bruijn 分支编码和 d 值生成以及投影后缀排序，都是高效的。原因如下：

(i) de Bruijn 分支编码大大减少了具有长公共前缀的后缀排序的成本。

(ii) 这些步骤的设计适用于并行计算，可以充分利用多个 CPU 核心。

(iii) 除了并行性外，多个步骤，如 k-mer 计数、k-mer 的基数排序、de Bruijn 分支编码和$|d|$值的生成，具有准线性时间复杂度。

其次，I/O 操作成为主要瓶颈。在文件转换和处理大型序列时，需要进行大量的 I/O 操作。虽然这些操作在理论上具有较低的时间复杂度，但它们的性能受到计算机文件系统和程序实现的影响。

Table 1. Running Time with 32 CPU cores (in minutes)

Methods	Human genomes	Human contigs	Primate genomes
deBWT	134	129	330
deBWT (no conversion)	48	56	100
ParaBWT	241	262	180
RopeBWT2	1694	2247	1546

Table 2. The time of the various steps of deBWT (in minutes)

Steps	Human genomes	Human contigs	Primate genomes
Phase1: dBG building and analysis			
k-mer counting	16	16	26
File conversion	87	74	229
k-mer sorting	3	3	8
dBG analysis	8	7	19
Phase2: The generation of de Bruijn branch encoding and projection suffixes			
de Bruijn branch encoding and $\phi(\cdot)$ values generation	9	12	16
Phase3: BWT construction with projection suffixes			
Projection suffixes sorting	4	12	10
Additional processing			
Additional processing	7	6	22
Supplement			
k-mer counting with KMC2	7	9	12

图 2 实验结果（表 1 和表 2）

除了上述提到的两个问题之外，投影后缀排序步骤的时间成本尤为关键，因为它是处理输入序列中长重复的核心步骤。这一步的总时间成本为$\sum_{j=1}^{|U^G|} t(U_j^G)$，其中$t(U_j^G)$是解决 BWT 的第$j$个未解部分所需的时间。由于每个未解部分都是用快速排序处理的，时间成本可以表示为$t(U_j^G) = O\left(N_j logN_j + \sum_{i=1}^{N_j} DP(PS_i)\right)$，其中 N 是未解部分$j$涉及的投影后缀个数的总和，$DP(PS_i)$是该部分中第$i$个投影后缀（记为$PS_i$）的区分前缀的长度。

在这里，PS的区别前缀是PS_i的最短前缀，用于确定相应部分的 BWT 字符。对于高度相似的输入序列G_1，G_2，…，G_{ND}，在理论上，$DP(PS_i)$的上限是$O(|\mathrm{dE}^{G_M}|)$，这是由于长重复的存在，其中G_M是具有最长 de Bruijn 分支编码的输入序列，$|\mathrm{dE}^{G_M}|$是相应 de Bruijn 分支编码的长度。

Table 3. Quantiles of $\overline{DP_j}$ and DP_j^M values of the 10 human genomes dataset

Quantiles	0.50	0.90	0.95	0.99	0.999	0.9999
$\overline{DP_j}$	107	588	2382	95 019	298 598	515 006
DP_j^M	1760	11 872	238 368	1 925 600	3 232 832	3 387 040

Table 4. Running time with various numbers of threads (in minutes)

Methods	8 threads	16 threads	24 threads	32 threads
Human genomes				
deBWT	194	153	142	134
deBWT (no conversion)	109	68	56	48
ParaBWT	265	240	240	241
Human contigs				
deBWT	183	154	123	129
vdeBWT (no conversion)	116	86	56	56
ParaBWT	294	277	276	262
Primate genomes				
deBWT	423	355	332	330
deBWT (no conversion)	193	125	105	100
ParaBWT	196	182	181	180

图 3 实验结果（表 3 和表 4）

尽管理论上的上限很大，但$DP(PS_i)$的实际值也在很大程度上取决于基因组变异的分布以及输入序列的重复性，这可能导致实际中的值较低。作者评估了基准测试中最重复数据集

（即 10 个人类基因组数据集）的$\overline{DP_j}$值和DP_j^M值，其中$\overline{DP_j}$和DP_j^M分别是第 j 个未解决部分内投影后缀的区分前缀的平均长度和最大长度。表 3 显示了 10 个人类基因组数据集的 DPj 和 DPM j 值的一系列分位数。这些分位数表明，对于大多数块，区分前缀很短。

我们还使用 8、16、24 个线程运行了 deBWT 以调查其可扩展性。结果（表 4）表明，deBWT 随着线程数量的增加逐渐加速，即具有良好的可扩展性。然而因为 ParaBWT 方法的增量特性，ParaBWT 的速度在各种线程设置下几乎相同。

使用不同数量线程的 deBWT 各个步骤的时间如图 3 所示。结果表明两个核心步骤，de Bruijn 分支编码和$|d|$值的生成以及投影后缀排序（步骤 4 和 5），是最具可扩展性的步骤。

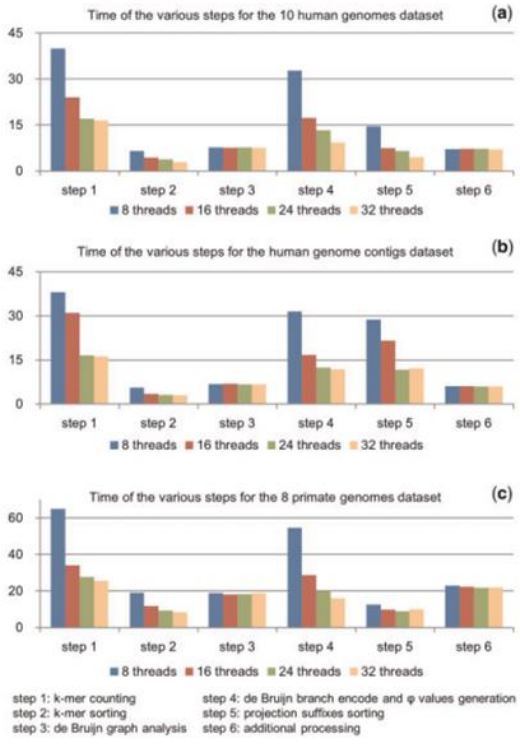

Fig. 3. Time consumption of the various steps of deBWT. The bars respectively indicate the elapsed time (in minutes) of the various steps of deBWT for the 10 human genomes dataset (**a**), the human genome contig dataset (**b**) and the 8 primate genomes dataset (**c**). Bars in the same color correspond to a specific number of threads, i.e. blue, red, green and purple bars are respectively for 8, 16, 24 and 32 threads

Table 5. Running time of the *in silico* human genome dataset with various configurations on the k parameter (in minutes)

Methods	k = 19	k = 23	k = 27	k = 31
deBWT	142	124	131	134
deBWT (no conversion)	75	51	47	48

'DeBWT' indicates the elapsed time of deBWT, and 'deBWT (no conversion)' deducts the time of the format conversion of Jellyfish output file.

Table 6. Memory footprints with 32 CPU cores (in Gigabytes)

Methods	Human genomes	Human contigs	Primate genomes
deBWT	120/78/38	120/63/34	235/203/58
ParaBWT	30	30	29
RopeBWT2	30	24	40
Supplement			
KMC2	119	119	119

For the '*x*/*y*/*z*' of deBWT in the memory columns, the x, y and z values respectively indicate the memory footprints of Jellyfish, phase1 of deBWT, and phases2 and phases3 of deBWT.

Table 7. Statistics on the in silico human genomes and contigs

Statistics	Human genomes	Human contigs
length of input sequences	30955436371	30200003020
distinct k-mers	5073730669	4025285321
multiple-out k-mers	18820763	17238123
multiple-in k-mers	18821805	17237511
copies of multiple-out k-mers	2364004617	2301293218
copies of multiple-in k-mers	2364445711	2300904807

图 4 实验结果（图 3 和表 5,6,7）

进一步在虚拟人类基因组数据集上运行 deBWT，使用不同的 k 参数配置来调查其影响（表 5）。从结果可以观察到，在大范围的k参数（即$k = 23{\sim}31$）上，总运行时间接近，但对于较小的k参数，时间消耗更高。这可能是因为适度长的k-mer（例如 23 到 31-mer）可能具有类似的能力来跨越短重复序列。

deBWT 的内存占用（表 6）取决于方法本身和所使用的k-mer 计数工具。deBWT 三个阶段的主要 RAM 成本不同。

- 第一阶段，主要成本来自 k-mer 排序的数据结构。
- 第二阶段的成本更为复杂，需要同时在内存中保留输入序列、de Bruijn 分支索引和生成的 de Bruijn 分支编码。

四、 论文方法的优缺点分析

deBWT 的主要优点在于**其基于 dBG 的后缀表示和组织，这有助于处理具有长公共前缀的后缀的比较，并避免不必要的比较。**此外，由于其非增量设计，deBWT 具有**良好的可扩展性**，适用于各种计算资源。因而 deBWT 非常适合于使用现代服务器或集群构建大量高度相似或重复的基因组的 BWT。在实验中，deBWT 在索引多个个体人类基因组和 contigs 的速度上实现了显著的改进。对于更多样化的基因组，例如多个灵长类基因组，deBWT 也显示出更快的速度和更好的并行化；然而，改进幅度较小，可能是因为 dBG 的密度较低。也就是说，要处理的 k-mer 和单路径更多，但输入的整体重复性低于高度相似的基因组。

但是，与最先进的方法相比，deBWT 具有**明显更大的内存占用**。有潜在的解决方案可以减少 deBWT 各阶段的内存占用。

- 对于第一阶段，将 k-mer 分成几个子集并分别在有限的内存中对每个子集进行排序是可行的。多个子集的结果可以直接合并到具有较小内存空间的所有 k-mer 的排序列表中。
- 对于第二阶段，通过仅保留一部分/d/值和 BWT 字符，也可以减少内存占用。因为在第二阶段之前就已知所有多入 k-mer 及其副本的数量，所以它可以将所有多入 k-mer 的整个集合划分为几个子集。每个子集具有有限数量的 k-mer 副本。因此，第二阶段可以多次扫描输入序列，而不是一次性进行。在每次扫描中，只有在相应子集内的多入 k-mer 的副本才会被识别、记录并输出到具有有限 RAM 空间的特定文件中。
- 由于第三阶段的所有子集彼此独立，因此可以分别处理子集的文件以生成 BWT 的各种部分。此外，BWT 部分可以直接合并以完成构建。这种策略在有限的工作空间中是可行的，但是会牺牲时间，因为它需要多次执行第二阶段。
- 对于第三阶段，也可以只在内存中保留未解 BWT 分区的一部分，因为所有这些分区都是独立的。

基于上述缺点，作者提出了三种改进方案：

- 首先，deBWT 直接使用快速排序对投影后缀进行排序。因为 de Bruijn 分支编码也可以看作是一种特殊的 DNA 序列，所以也可能使用其他方法进一步加速投影后缀排序步骤。例如，Karkkainen 提出的方法使用 DCS 加速原始输入序列的分箱后缀排序。这种方法也可以用于对投影后缀的分箱排序，而不会损失并行计算的能力，因为它是非增量的。
- 其次，对于当前版本的 deBWT，I/O 密集型步骤仍未优化，这降低了速度。我们计划进一步优化 I/O 密集型步骤，以提高 deBWT 的效率。
- 同时，由于 k-mer 计数仍然是一个开放性问题，而先进的 k-mer 计数工具正在发展，我们还计划使用其他更先进的 k-mer 计数工具替换 Jellyfish，或通过直接访问默认的 Jellyfish 输出文件来删除文件转换步骤，以打破该方法的实际瓶颈。

未来基因组学研究中，随着已组装基因组数量的快速增加，对许多基因组的良好组织和索引将得到广泛需求。作为重要的基因组索引数据结构，BWT 仍有许多应用和发展空间。