doi:10.3969/j.issn.1672-1292.2024.04.004

基于优化 BWT 索引技术的序列比对算法研究

胡春玲[1],赵俊杰[2],姚梦媛[1],高欢欢[1],朱艺杭[1],汪少鸿[1]

(1.合肥大学人工智能与大数据学院,安徽 合肥 230031)
(2.郑州大学计算机与人工智能学院 河南 郑州 450001)

[摘要] 生物信息学中,大规模的生物基因序列比对是最重要的基础问题. 针对主流的 BWT(burrows-wheeler transform)索引技术的研究,提出一种新的多阶混合 BWT 索引方法 MD-BWT(multi difference cover mod3 burrows-wheeler transform),根据待比对序列的长度,动态选取适合的多位索引查找. 实验结果表明,改进后的方法可以有效减少序列比对算法中的比对次数和计算次数,降低序列比对算法中索引算法的时间复杂度,明显提高序列比对的效率. 在构造 BWT(S)字符串过程中,通过 DC3(difference cover mod 3)算法来构造后缀数组,实验表明 DC3 算法构造后缀数组比倍增算法的时间复杂度更低,时间性能更优.

[关键词] 长序列比对,BWT 索引,DC3,后缀数组
[中图分类号]TP181 [文献标志码]A [文章编号]1672-1292(2024)04-0037-09

Research on Sequence Alignment Algorithm Based on Optimized BWT Indexing Technique

Hu Chunling[1],Zhao Junjie[2],Yao Mengyuan[1],Gao Huanhuan[1],Zhu Yihang[1],Wang Shaohong[1]

(1.School of Artificial Intelligence and Big Data,Hefei University,Hefei 230031,China)
(2.School of Computer Science and Artificial Intelligence,Zhengzhou University,Zhengzhou 450001,China)

Abstract:Large-scale gene sequence alignment is the most important fundamental problem in bioinformatics. Based on the mainstream research of BWT index technology,the paper proposes a new multi-order mixed BWT index method,which dynamically selects the appropiate multi-bit indexing according to the length of the sequence to be compared. The experimental results show that the improved method can effectively reduce the number of comparison and calculation times,reduce the time complexity of the index algorithm,and significantly improve the efficiency of sequence comparison. In the process of constructing BWT(S) string,this paper uses DC3(difference cover mod 3) algorithm to construct the suffix array. Experiments show that DC3 algorithm has better time performance compared to Binary Lifting.

Key words:sequence alignment,BWT index,difference cover mod 3,suffix array

生物序列比对是一种用于比较基因组、蛋白质序列以及 RNA 序列的分析工具,通过将两个或多个生物序列进行比较,找出相同或相似的部分,以研究生物进化、功能、结构等方面的问题[1]. 生物序列比对是生物信息学的基础技术之一,也是现代生物学研究的重要手段之一. 生物序列比对在基因注释和功能、诊断和治疗、生物工程和生物技术中都有比较重要的作用. 通过比对不同物种的基因组或编码蛋白质序列,可以研究物种进化的关系和演化过程;通过比对已知的蛋白质或 RNA 序列,可以预测新序列的功能和意义;基于生物序列比对,可以诊断和治疗某些疾病,例如肿瘤基因测序和抗生素耐药性检测[2].

根据序列比对算法所基于的索引技术不同,序列比对算法主要分为两大类,即基于哈希(hash)索引的序列比对算法和基于 BWT 索引的序列比对算法. 针对第一代规模较小的测序数据,基于 hash 索引的比对软件主要有 SOAP[3]、PASS[4]、RMAP[5-6]、MAQ[7]、SHRiMP[8]等. SOAP 是一种基于哈希索引的快速比对算法,通过将基因组序列分割成较小的片段,构建哈希表来加速比对过程. PASS 算法着重于发现序列间的模式相似性,通过预处理步骤和哈希索引结构实现了高效的模式匹配和序列比对. RMAP 算法主要用

收稿日期:2024-05-12.
基金项目:国家自然科学基金青年项目(62306100)、安徽省教学研究重大项目(2023jyxm0558).
通讯作者:胡春玲,博士,教授,研究方向:机器学习、生物信息学. E-mail:huchunling@ hfuu.edu.cn

于 RNA 序列的比对,利用哈希索引和精确匹配策略,能够准确地识别 RNA 序列间的相似性和结构特征. MAQ 算法是一种综合性的基因组序列比对工具,结合了哈希索引、质量评估和序列拼接等功能,能够处理各种类型的测序数据并提供高质量的比对结果. SHRiMP 算法是专门针对短读取序列的比对工具,通过哈希索引和动态规划算法,实现了精确的序列比对. 基于哈希索引的序列比对技术各有特色,适用于规模较小的生物序列比对任务.

随着测序技术的飞速发展,需要处理的生物数据量越来越大,一系列基于 BWT 索引技术的序列比对算法应运而生[9-11]. 基于 BWT 的比对算法软件主要有 Bowtie[12-13]、BWA[14-15]、二阶 BWT[16]、SBWT[17]、LCP-BWT[18]等. Bowtie 是一种快速、内存占用低的短读取序列比对工具,采用 BWT 和 FM Index 结构[19],能够高效地处理大规模基因组数据. Bowtie2 是 Bowtie 的升级版本,具有更高的比对速度和更好的灵活性,支持更长的读长和更大的基因组数据. BWA 是一种广泛应用的基因组序列比对工具,包括多种模式,适用于不同类型的测序数据和比对需求,其中基于 BWT 技术和 Smith-Waterman 算法的比对工具主要用于长读取序列的比对任务,能够处理序列间的局部相似性和结构特征. 二阶 BWT 通过双位索引查找提高了比对效率,但不支持模糊比对. SBWT 采用间隔后缀,结合鸽巢原理和二分查找建立二级索引加速比对扩展,支持碱基替换的模糊比对,但不支持插入或缺失的近似比对问题. LCP-BWT 通过构建最长公共前缀数组,减少最大重复的计算,需要较大的外部内存来存储额外建立的数组. BWT 索引技术被广泛应用于后缀数组的构建过程,成为序列比对算法的核心技术,但其缺点是需要访存次数多、计算量大[20]. 因此,如何平衡 BWT 索引技术带来的空间、时间和精度问题,已成为优化基于 BWT 索引技术的序列比对算法的核心问题[21-22].

本文提出一种新的多阶混合 BWT 索引方法 MD-BWT,主要从两方面对基于 BWT 索引的比对算法进行性能优化. 一方面,传统的一阶 BWT 方法通过逐位索引来实现查找,二阶 BWT 方法通过双位索引实现查找,本文所提出的基于 BWT 的多阶混合索引方法 MD-BWT 根据待比对序列的长度动态选取适合的多位索引查找,可以有效降低序列比对时的访存次数,明显降低序列比对的时间复杂度. 另一方面,传统的 BWT(S)字符串通过对参考序列进行循环移位,再通过字典序排序选出其中的 BWT(S)字符串. 倍增算法可用于构造后缀数组,省略对字符串排序的过程,只需求出后缀数组,就可求出 BWT(S)字符串. 本文采用 DC3 算法[23]构造后缀数组,该算法在构造后缀数组的时间性能上优于倍增算法.

1 一阶 BWT 算法

一阶 BWT 算法的基本思想是将原始字符串转换为其所有旋转字符串的排序后的矩阵,然后从矩阵的最后一列构建 BWT 转换. 该算法的主要优点是可以快速处理大型数据集,且在大多数情况下可实现快速解压缩.

一阶 BWT 过程可以简单概括为 BWT 的构造过程和 BWT 的解码过程. BWT 构造过程主要是对原始字符串 S 进行预处理,得出 BWT(S)字符串和构造需要的索引辅助数组. BWT 的构造过程可以分为循环移位排序和提取 BWT(S). 循环移位排序是将原始字符串 S 末尾加上一个特殊字符 $(这个字符一定要小于 A,C,G,T),设这个加了 $ 的字符串为 S′,对 S′进行循环移位(把最后的字符放到最前面的位置),最后重复这个过程,直到完成 S 的字符串长度加 1 次循环移位,从而得到一个矩阵 $\boldsymbol{M}'$.

提取 BWT(S)主要是对 $\boldsymbol{M}'$矩阵按照字典序进行排序,从而得到一个新的矩阵 $\boldsymbol{M}$,$\boldsymbol{M}$ 的最后一列构成的字符串就是 BWT(S)字符串,图 1 和图 2 展示了针对字符串 S,构成对应的 BWT(S)的过程.

BWT 的解码过程主要是根据已有的 BWT(S)字符串和索引辅助数组还原出原始字符串 S. 在进行 BWT 构造的过程中,可以构建 3 个辅助索引数组:SA(suffix array)后缀数组、OCC 数组和 BWT(S)字符串数组,其中,SA 保存了 BWT 矩阵中 $ 字符之前各后缀在原始字符串 S 中出现时的起始坐标位置.

设 $\boldsymbol{M}$ 矩阵的第一列字符串和最后一列字符串分别为 F 字符串和 L 字符串,而这两个字符串具有一个性质:F 列和 L 列的映射满足稳定性,即字符串中可能会出现很多相同字符,但这些字符在 F 列和 L 列对应的位置是不变的,也即 L 列中相同字符间的相对位置在 F 列中是不变的,例如,在 F 列中第 2 次出现的字符 C 与 L 列中排在第二个位置的字符 C 是同一个字符. OCC 数组中 OCC[i]代表第 i 行的 BWT(S)字符在整个 BWT(S)字符中出现的次数. F 字符串和 L 字符串还有一个性质:对于某个相同下标,F 字符串中的字符 F[i]是 L 字符串中字符 L[i]的下一个. 假设原始字符串 S = ACCGATG,S′ = ACCGATG$,F =$AACCGGT,

L=G$GACTCA,对于同一个下标 i=5 时,F[5]=G,L[5]=T,在原始字符串 S 中,F[5]对应的字符是 L[5]对应的字符的前一个字符. 通过这一性质和辅助数组 SA、OCC,就可以根据 F 列和 L 列来还原出原始字符串 S,并能根据这些性质和辅助数组进行序列比对. 图 3 描述了这一序列比对过程.

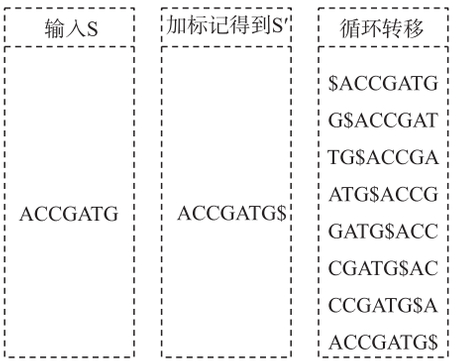

图 1　构造 BWT(S)步骤 1

Fig. 1　Step 1 of constructing BWT(S)

图 2　构造 BWT(S)步骤 2

Fig. 2　Step 2 of constructing BWT(S)

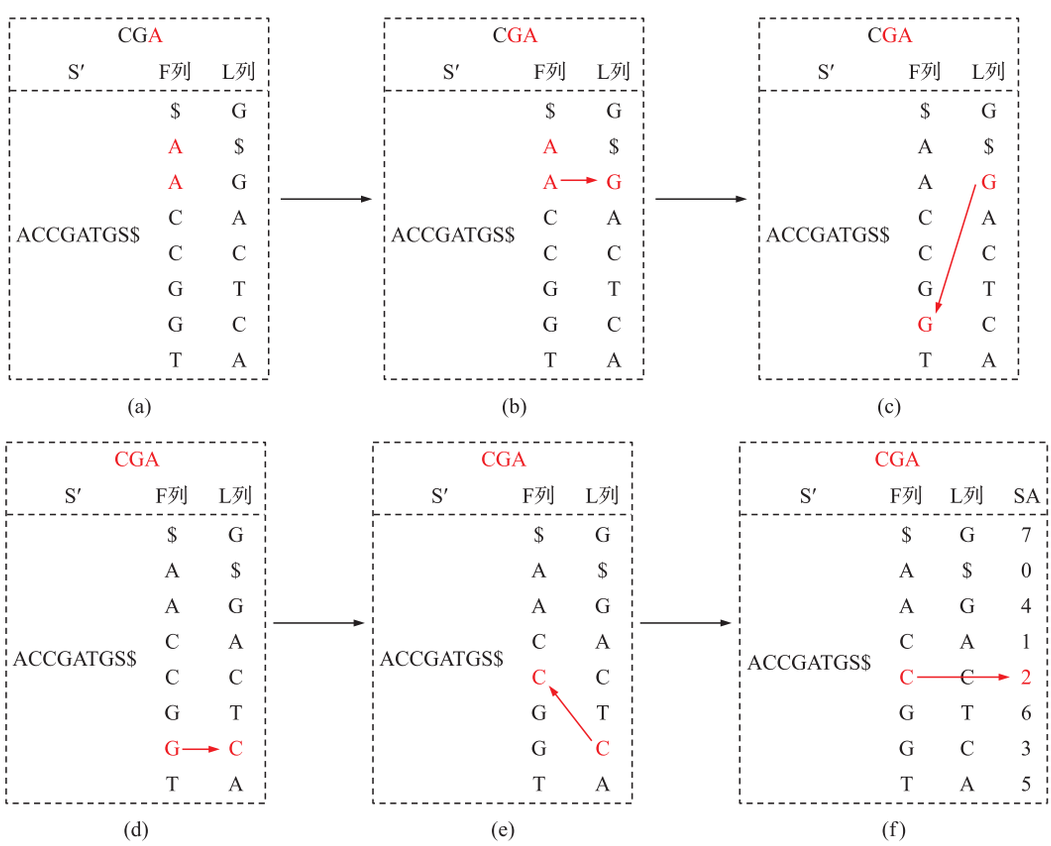

图 3　基于索引数组的序列比对过程

Fig. 3　Sequence alignment process based on the index array

2　改进的多阶 BWT 混合索引序列比对方法 MD-BWT

2.1　后缀数组的优化

计算 BWT(S)字符串的关键步骤是构建原始字符串的后缀数组 SA. 假设字符串 S 长度为 N,对于直接构造 BWT 矩阵然后再求 BWT(S)字符串的时间复杂度是 $O(N^2\log^N)$. DC3 算法是一种高效的构造字符串后缀数组的算法,该算法将原始字符串 S 转化为一个新的三元组序列 T,并通过多次排序得到后缀数组 SA,可以把构造后缀数组的时间复杂度和空间复杂度都降低为 $O(N)$. 表 1 所示为在 DC3 算法实现过程中使用的临时数组与变量.

表 1　DC3 算法中的临时数组

Table 1　Temporary array of the DC 3 algorithm

符号	定义
T_k	T 中所有下标 mod3 等于 k 的位置,$T_k[i]$代表的是以之为起点的后缀 $T[T_k[i]\cdots]$
ST_{12}	T_1+T_2
SA_0	T_0 按照其所代表的后缀排序(Step2)后的结果
SA_{12}	T_{12}按照其所代表的后缀排序(Step1)后的结果
R_{12}	辅助数组,记录 T_{12}中元素的排名,即:在 SA_{12}中的位置(排名从 1 开始,0 作为哨兵)
SA	T 的后缀数组
R	辅助数组,记录 T 的每个后缀 $T[i\cdots]$($0\leqslant i<m$ 的排名)

DC3 算法的描述如下,其流程分为以下 3 个步骤:

Step1　构造 T_{12},并排序生成 SA_{12}和 R_{12},若 R_{12}中有重复元素(排名暂不能确定)则递归调用:$SA_{12}=DC3(R_{12})$.

构造 T_{12}以后,以 T_{12}中每一个下标为开头,把三元组字符串 T 分块为长度为 3 的字符串,根据字符串后缀的重复特性,对这些分块进行基数排序. 若后缀的前 3 个字符互不相同,得到的 R_{12}中的值也互不相同,只根据前 3 个字符便可正确排序得到 SA_{12}. 而后将基数排序的结果赋值给 R_{12},若 R_{12}中的值有相同部分,则需要把 R_{12}的值再进行分块,重复进行基数排序,直至最终 R_{12}的值互不相同. 当 R_{12}的值都不相同时,就开始 Step2.

Step2　根据 SA_{12}构造 SA_0.

经过 Step1 以后,当基数排序到 R_{12}中的值都不相同时,即得到 SA_{12}和 R_{12}. 此时,$T[T_{12}[i]]$的次序已经存在于 R_{12}中,因而在对 T_0 所代表的后缀进行排序时,只需比较字符对($T[T_0[i]]$,$R_{12}[i]$),就可得到 SA_0.

Step3　将 SA_{12}和 SA_0 合并生成后缀数组 SA,并返回 SA 数组.

得到了 SA_0 和 SA_{12}以后,两者所代表的后缀各自有序,通过一次归并排序构造 SA,即可计算出 T 的后缀数组 SA.

算法 1　DC3 算法

输入:长度为 N 的字符串 S
输出:S 的后缀数组 SA
(1)初始化:$S[N]=S[N-1]=S[N-2]=0$
(2)if($N\%3==1$)then $S[N]=S[N-1]=0$
(3)if($N\%3==2$)then $S[N]=0$
(4)初始化长度为 3 的元组数组 T
(5)将 S 中所有长度为 3 的子串的起始下标依次存入 T 中
(6)把 T 按字典序排序
(7)初始化长度为 3 的排名数组 P,令 $P[T[1]]=1$,其余元素的排名依次递增,相同的排名相等
(8)将长度为 3 的排名数组 P 扩展到长度为 N 的排名数组 P1,其中未出现的下标的排名暂定为 0
(9)初始化:$k=3$
(10)while $k<=N$
　①用 P1 生成长度为 $2k$ 的元组数组 T1
　②把 T1 按字典序排序
　③初始化长度为 $2k$ 的排名数组 P2,令 $P2[T1[1]]=1$,其余元素的排名依次递增,相同的排名相等
　④将长度为 $2k$ 的排名数组 P2 扩展到长度为 N 的排名数组 P3
　⑤令 $k=2k$
(11)end while
(12)$SA[1]=N$
(13)for $i=1$ 到 N
　if $SA[i+1]=T[SA[i]]$then $SA[i+1]\%3!=0$
(14)end for
(15)for $i=0$ 到 N

①if SA[i]<N-2
if(SA[i]%3)= =1 then Tmp[j++]=SA[i]+1
②end if
(16)end for
(17)radix_sort(Tmp,j,S,0,N);//基数排序
(18)for i 到 j SA[k+i]=Tmp[i]
①j=k+j
②k=0
(19)for i 到 N if(SA[i]<N-2)if(SA[i]%3)= =0 then Tmp[k++]=SA[i]
(20)for(i=0,j=0;i<k&&j<j;i++)
①if(S[Tmp[i]]<S[SA[j]]||(S[Tmp[i]]= =S[SA[j]]&&P1[Tmp[i]+1]<P1[SA[j]+1]))SA[k++]=Tmp[i++]
②else SA[k++]=SA[j++]
(21)end for
(22)while(i<k)SA[k++]=Tmp[i++]
(23)while(j<N)SA[k++]=SA[j++]
(24)return SA

2.2 序列比对算法 MD-BWT

一阶 BWT 进行序列比对时,是每次从 reads(DNA 或 RNA 序列中读取的短序列)中从后往前循环查找,每次比对一个字符. 当有大量 reads 需要进行比对时,序列比对会消耗大量时间和内存. 本文所提 MD-BWT 方法在进行序列比对时,动态选择合适的字符比对长度,不再是每次比对一个字符,因而序列比对的时间效率会大大提升. MD-BWT 方法采用二进制划分法选择合适的序列长度[24],该划分法根据待比对序列 reads 的长度,把长度转换为二进制,根据二进制中 1 的数量和位置来确定具体的序列比对长度. 例如,待比对序列 reads 的长度为 13,转换为二进制是$(001101)_2$,故进行序列比对时可以分为{8,4,1}的序列长度来比对,该划分方法能有效减少序列比对的次数. 当 MD-BWT 方法构造出 $\boldsymbol{M}$ 矩阵后,只需维护原始字符串 S 的后缀数组,就可以构造出 $\boldsymbol{M}$ 矩阵的多阶信息. 一阶 BWT 索引技术选取矩阵的第一列和最后一列来构造辅助数组. 根据 BWT 索引技术的稳定性,若要构造二阶 BWT 索引,$\boldsymbol{M}$ 矩阵保持不变,只需要选取矩阵的前两列和最后两列,通过这个性质可以把索引信息推广到多阶. 所以只需要维护原始字符串 S 的后缀数组,就可以构造出 $\boldsymbol{M}$ 矩阵的多阶索引信息. 把多阶索引信息都存储下来,BWT[i][S]表示长度为 i 的 BWT(S)字符串,OCC[i][]也同理. 这样维护会产生大量的字符串信息,可以使用类哈希映射方法将这些字符串映射为独一无二的数字,以便于存储和记录字符串频次以及进行 FM 索引查找,可以减少很多内存的消耗. MD-BWT 方法的算法描述如下.

算法 2 MD-BWT 方法

输入:待查询的子序列 sub
输出:字符串 sub 在参考序列中出现的全部位置
(1)调用算法 1 计算后缀数组 SA
(2)通过 sub 字符串的长度,选择合适的序列字符比对长度,假设选择的序列比对长度为 x
(3)sub_x=截取 sub 的后 x 个字符
(4)计算 sub_x 在 $\boldsymbol{M}$ 矩阵中前 x 列第一次出现的位置(由于统计了所有字符串出现的频次,因此维护一个前缀和数组可以很方便地求取其出现的位置),赋值为 sp
(5)计算 sub_x 在 $\boldsymbol{M}$ 矩阵中前 x 列最后一次出现的位置(sp+sub_x 出现的频次就是最后一次出现的位置),赋值为 ep
(6)i=sub_len-x
(7)while sp<ep and i>=1
①根据现有的 sp,ep 和索引数组信息,根据后缀数组 SA,计算在 BWT[x][]中对应的出现位置
②bwt_sp=从 sp 位置向下,找到 sub_x 在 BWT[x][]中第一次出现的位置
③bwt_ep=从 ep 位置向上,找到 sub_x 在 BWT[x][]中最后一次出现的位置
④根据 F 字符串和 L 字符串的性质,找到下一个 sp 和 ep 位置
⑤sp=bwt_sp+count_num[x][bwt_sp][map]//map 为 sub_x 字符串的映射
⑥ep=bwt_ep+count_num[x][bwt_ep][map]

⑦根据剩余 sub 未匹配的字符串长度,进行新一轮的选择算法,进行合适长度的序列比对,假设选择长度为 y,进行循环

(8)end while

(9)if sp<ep

①for i=sp 到 ep

输出 SA[k]

②end for

(10)end if

多阶混合索引序比对过程如图 4 所示. 从图 4 可知,一阶 BWT 算法的 BWT(S)和二阶 BWT 算法的 BWT(S)对应的字符串都是最后一列或最后两列,这也是能进行动态混合序列比对的最重要属性之一.

与一阶 BWT 索引结构一样,通过构建多阶 BWT 索引的数组就可以进行序列比对. 对于给定待比对字符串 SUB,假设长度为 SUB_LEN,采用二进制划分法选出最合适的 BWT(S)字符串,也即选择最合适的阶数进行序列比对,例如,选择的 V 阶索引结构,就只需进行 SUB_LEN/V 次序列比对,当最后剩下的字符串长度不足 V 时,就需要动态切换索引结构来达到减少序列比对次数的目的. 这样的操作会大大降低序列比对的时间复杂度.

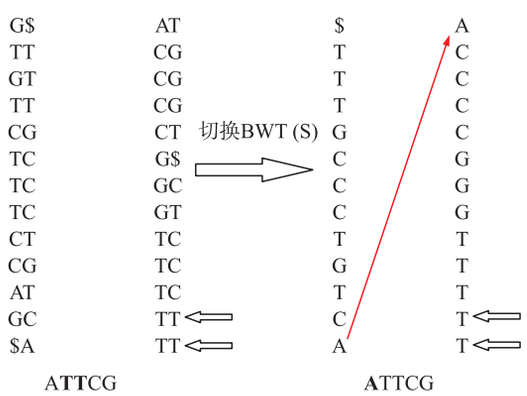

图 4　BWT 多阶混合索引序列比对图

Fig. 4　BWT multi-order mixed index sequence alignment

根据 BWT 矩阵的性质可知,切换 BWT(S)并不会增加时间复杂度,且对应的 sp 和 ep 也是原来索引的 sp 和 ep,因此可根据 SUB 长度的大小来选择合适的 BWT(S)字符串,实现最小次数的序列比对过程.

在 BWT 动态多阶混合序列比对中,只要内存容量足够大,就可记录更多阶数的 BWT(S)字符串,可以用更多的空间来换取时间上的优化. 由于生物序列基因通常不会突变,而待匹配序列 reads 有可能是几十万甚至几百万,所以维护更多索引信息用大量的空间换取时间是有意义的. 假设对生物序列基因的索引信息维护到了 k 阶,待匹配序列 read 的长度是 m,则匹配一个字符串的时间复杂度是 $O(k/m)$.

3　实验结果与分析

为了检验本文所提多阶混合序列比对算法 MD-BWT 的性能,本文进行以下测试:后缀数组的性能比对;多阶技术与传统一阶 BWT 索引的性能比对;MD-BWT 与其他同类软件的性能比对.

实验中参考基因组采用全人类基因组 GRCH37;模拟 reads 数据截取自参考基因组,长度为 36bp;真实 reads 数据为 SRX000600,长度为 36bp. 人类基因组和真实 reads 数据源自 NCBI[25],格式为 fasta.

实验环境:内存:8GB;硬盘:256GB;开发环境:Dev-C++;运行软件环境:Windows10 操作系统.

当下常用的比对软件主要有 Bowtie2 和 BWA,以下通过实验来对比验证 BWT 多阶混合索引比对方法 MD-BWT 的有效性. 由于 Bowtie2 和 BWA 只能在 Linux 和 Mac 系统下进行运行,故本文通过 WSL 实现 Windows 下 Linux 子系统来进行实验测试.

3.1　MD-BWT 算法正确性验证

本实验参考基因组采用全人类基因组 GRCH37,模拟 reads 数据截取自参考基因序列,进行序列比对时匹配率为 100%. 通过程序脚本,分别截取长度为 100~500bp 的 reads,每组大约截取 20 万组数据,并与同类比对软件进行比对,实验结果如表 2 所示.

表 2　MD-BWT 精准比对测试

Table 2　Precision comparison test of MD-BWT

reads 数量	reads 长度/bp	精确比对率/%
200 000	100	100
200 000	200	100
200 000	300	100
200 000	400	100
200 000	500	100

继续通过程序随机模拟一些 reads 数据来验证比对精度是否符合预期. 随机生成长度为 100~500 bp 的 reads,每组数据量大约为 200 万~500 万组,通过与 BWA 和 Bowtie 进行比较,来验证是否与同类比对

算法精度一致. 实验结果如表 3 和表 4 所示.

表 3　MD-BWT 与 BWA 比对测试

Table 3　Comparison between MD-BWT and BWA

MD-BWT 方法			BWA 软件方法		
reads 数量	reads 长度/bp	精确比对率/%	reads 数量	reads 长度/bp	精确比对率/%
2 000 000	100	73.6	2 000 000	100	73.6
2 000 000	200	42.3	2 000 000	200	42.3
2 000 000	300	11.2	2 000 000	300	11.2
2 000 000	400	12.6	2 000 000	400	12.6
2 000 000	500	13.7	2 000 000	500	13.7
3 472 910	100	42.7	3 472 910	100	42.7
5 481 240	200	15.4	5 481 240	200	15.4
1 212 200	300	12.1	1 212 200	300	12.1
3 201 100	400	5.6	3 201 100	400	5.6
5 000 000	500	4.3	5 000 000	500	4.3

表 4　MD-BWT 与 Bowtie 软件比对测试

Table 4　Comparison between MD-BWT and Bowtie

MD-BWT 方法			Bowtie 软件		
reads 数量	reads 长度/bp	精确比对率/%	reads 数量	reads 长度/bp	精确比对率/%
2 000 000	100	53.6	2 000 000	100	53.6
2 000 000	200	47.2	2 000 000	200	47.2
2 000 000	300	41.2	2 000 000	300	41.2
2 000 000	400	33.6	2 000 000	400	33.6
2 000 000	500	15.7	2 000 000	500	15.7
3 782 110	100	44.7	3 472 910	100	44.7
5 592 143	200	18.7	5 481 240	200	18.7
2 413 220	300	12.1	1 212 200	300	12.1
3 000 000	400	7.8	3 201 100	400	7.8
5 000 000	500	9.85	5 000 000	500	9.85

通过与 BWA 和 Bowtie 测试结果的比对可知,MD-BWT 技术可以达到预期效果,可以实现序列比对的效果,测试数据与现有软件完全一致. 经过在模拟数据和真实数据上的比对实验可以发现,本文提出的 MD-BWT 方法可以完成序列比对任务.

3.2　DC3 算法有效性验证

后缀数组的时间性能提升直接关系到构造索引数组的速度. 分别采用倍增算法和 DC3 算法构建后缀数组 SA,通过时间判断进行性能分析. 设每组有 20 个测试样例,每个测试样例的数据量在 200 万~800 万,测试数据均来自于全人类基因组中的参考基因. 实验结果如图 5 所示.

根据实验数据可知,对于长度为 N 的相同的人类基因组,DC3 算法构造后缀数组的时间明显优于倍增算法. 且随着实验数据的增大,倍增算法的时间增长速率更快,DC3 算法的时间增长速率则较慢,这一性能差异主要归因于两种算法的时间复杂度不同:DC3 算法构造后缀数组的时间复杂度大约是 $O(N)$,而倍增算法的时间复杂度则是 $O(N\log N)$.

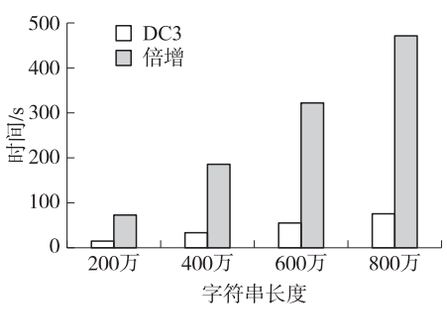

图 5　DC3 算法和倍增算法构造后缀数组时间比对

Fig. 5　Temporal comparison of the suffix array between DC3 and multiplication algorithm

3.3　MD-BWT 算法有效性验证

针对相同的数据集的序列比对问题,本文比较一阶索引序列比对 BWT 算法和多阶混合索引方法 MD-BWT 的时间性能. 测试数据源自 NCBI,待比对序列从中直接截取子串,截取的长度分别为 100~400bp,每组数据会进行 10 万条 reads 的测试. 实验结果如图 6 所示,表明与一阶 BWT 算法相比,随着子串长度的增加,算法 MD-BWT 的时间性能优势明显.

基于 BWT 技术的索引算法可以对索引数组进行跳跃的压缩存储[26]. 在实际应用操作中,将索引数组进行隔行 64 行存储一次,减少辅助数组的空间浪费,用相同实验数据量和规模进行再次实验,实验结果如图 7 所示,表明在压缩存储的情况下,与一阶 BWT 算法相比,算法 MD-BWT 同样具有时间性能优势.

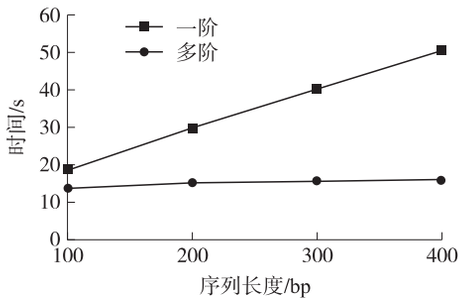

图 6　一阶 BWT 与 MD-BWT 的时间对比

Fig. 6　Temporal comparison between the first-order BWT and MD-BWT

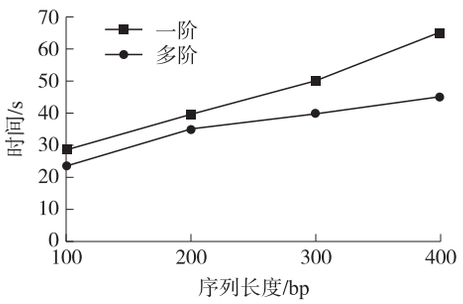

图 7　一阶 BWT 与 MD-BWT 的时间对比(压缩存储)

Fig. 7　Temporal comparison between the first-order BWT and MD-BWT(compression storage)

4　结论

本文在生物序列比对的经典一阶索引 BWT 方法的基础上,提出了多阶混合 MD-BWT 算法. 该算法可以根据待比对序列中 reads 长度的不同来选择合适的字符比对长度,可以显著减少序列比对中循环和计算的次数,显著提高算法的时间性能. 从真实的实验数据中可以发现,MD-BWT 方法在保证序列比对精度的前提下,有效提升了序列比对的效率. 在 MD-BWT 方法中,选择合适的比对步长对算法的时间性能影响很大,故下一步的工作重心是设计出更优的序列长度选择算法,进一步提升序列比对算法的效率,减少存储多阶索引信息,降低序列比对算法的空间复杂度.

[参考文献](References)

[1] SARKAR A,GHOSH S,RAY S S. A hardware-based memory-efficient solution for pair-wise compact sequence alignment[J]. IETE Journal of Research,2023,69(6):3638-3649.

[2] 刘济晗. 三代 RNA 测序序列的比对和分析工具[D]. 哈尔滨:哈尔滨工业大学,2018.

[3] LI R Q,LI Y R,KRISTIANSEN K,et al. SOAP:short oligonucleotide alignment program[J]. Bioinformatics,2008,24(5):713-714.

[4] CAMPAGNA D,ALBIERO A,BILARDI A,et al. PASS:a program to align short sequences[J]. Bioinformatics,2009,25(7):967-968.

[5] SMITH A D,XUAN Z Y,ZHANG M Q. Using quality scores and longer reads improves accuracy of Solexa read mapping[J]. BMC Bioinformatics,2008,9(1):128.

[6] SSERWADDA I,MBOOWA G. rMAP:The rapid microbial analysis pipeline for ESKAPE bacterial group whole-genome sequence data[J]. Microbial Genomics,2021,7(6):000583.

[7] LI H,RUAN J,DURBIN R. Mapping short DNA sequencing reads and calling variants using mapping quality scores[J]. Genome Research,2008,18(11):1851-1858.

[8] RUMBLE S M,LACROUTE P,DALCA A V,et al. SHRiMP:accurate mapping of short color-space reads[J]. PLoS Computational Biology,2009,5(5):e1000386.

[9] BURROWS M J,WHEELER D. A block-sorting lossless data compression algorithm:SRC Research Report 124[R]. Palo Alto,CA,USA:Digital Systems Research Center,1994.

[10] FUENTES-SEPMLVEDA J,NAVARRO G,NEKRICH Y. Parallel computation of the Burrows Wheeler Transform in compact space[J]. Theoretical Computer Science,2020,812:123-136.

[11] GAGIE T,MANZINI G,SIRÉN J. Wheeler graphs:A framework for BWT-based data structures[J]. Theoretical Computer Science,2017,698:67-78.

[12] LANGMEAD B,TRAPNELL C,POP M,et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human

genome[J]. Genome Biology,2009,10(3):R25.
[13] LANGMEAD B,SALZBERG S L. Fast gapped-read alignment with Bowtie 2[J]. Nature Methods,2012,9(4):357-359.
[14] LI H,DURBIN R. Fast and accurate long-read alignment with Burrows-Wheeler transform[J]. Bioinformatics,2010,26(5):589-595.
[15] KEEL B N,SNELLING W M. Comparison of burrows-wheeler transform-based mapping algorithms used in high-throughput whole-genome sequencing:application to illumina data for livestock genomes[J]. Frontiers in Genetics,2018(9):35.
[16] 赵雅男,徐云,程昊宇. 序列比对算法中的 BW 变换索引技术研究及其改进[J]. 计算机工程,2016,42(1):282-286.
[17] CHANG C H,CHOU M T,WU Y C,et al. sBWT:Memory efficient implementation of the hardware-acceleration-friendly Schindler transform for the fast biological sequence mapping[J]. Bioinformatics,2016,32(22):3498-3500.
[18] EGIDI L,LOUZA F A,MANZINI G,et al. External memory BWT and LCP computation for sequence collections with applications[J]. Algorithms for Molecular Biology,2019,14:6.
[19] LIU Y C,SCHMIDT B,MASKELL D. CUSHAW:a CUDA compatible short read aligner to large genomes based on the Burrows-Wheeler transform[J]. Bioinformatics,2012,28(14):1830-1837.
[20] KUMAR S,AGARWAL S,RANVIJAY. Burrows wheeler transform and wavelet tree based retrieval of genome sequence in an indexed genome database[J]. Recent Advances in Computer Science and Communications,2020,13(6):1213-1220.
[21] SILVA M,PRATAS D,PINHO A J. AC2:an efficient protein sequence compression tool using artificial neural networks and cache-hash models[J]. Entropy(Basel),2021,23(5):530.
[22] CHEN Y,YOU D L,ZHANG T J,et al. SLDMS:a tool for calculating the overlapping regions of sequences[J]. Frontiers in Plant Science,2022,12:813036.
[23] BINGMANN T. Scalable string and suffix sorting:algorithms,techniques,and tools[EB/OL]. (2018-08-02)[2024-05-12]. https://doi.org/10.48550/arXiv.1808.00963.
[24] ZHENG W B,CHEN J,DOAK T G,et al. ADFinder:accurate detection of programmed DNA elimination using NGS high-throughput sequencing data[J]. Bioinformatics,2020,36(12):3632-3636.
[25] KIM Y,CHOI S,JEONG J,et al. Data dependency reduction for high-performance FPGA implementation of DEFLATE compression algorithm[J]. Journal of Systems Architecture,2019,98:41-52.
[26] GRIEBLER D,HOFFMANN R B,DANELUTTO M,et al. High-level and productive stream parallelism for Dedup,Ferret,and Bzip2[J]. International Journal of Parallel Programming,2019,47(2):253-271.

[责任编辑:严海琳]