doi:10.12068/j.issn.1005-3026.2019.05.001

基于频率的 Read Mapping 种子选择算法

马海涛,祁 实,于长永,赵宇海
(东北大学 计算机科学与工程学院,辽宁 沈阳 110169)

摘 要:选择具有最低频率的最优种子是一个复杂的计算问题,往往需要很长时间.提出了一种 read 的基于频率的合并种子选择算法(FMSS),该算法能够高效地选择接近最优的种子集合,可用于改善现有映射工具的性能.实验对比了平均种子选择方法和当前最优的种子选择策略(OSS,optimal seed solver),结果显示 FMSS 算法能够用很少的时间代价给出接近 OSS 的最优种子集合,这表明 FMSS 算法可集成到现有映射工具中用于处理更大规模的 read mapping 问题.

关 键 词:种子频率;读取映射;平均种子;频率合并;最优种子
中图分类号:TP 311.31 **文献标志码**:A **文章编号**:1005-3026(2019)05-0609-05

Frequency-Based Seed Selection Algorithm for Read Mapping

MA Hai-tao, *QI Shi*, *YU Chang-yong*, *ZHAO Yu-hai*
(School of Computer Science & Engineering, Northeastern University, Shenyang 110169, China. Corresponding author: MA Hai-tao, E-mail: mahaitao@126.com)

Abstract: The selection of the optimal seed (that is, the seed with the lowest frequency) is a complex calculation problem, which often takes a long time. A frequency-based merge seed selection (FMSS) algorithm is proposed, which can efficiently select the suboptimal set of seeds and improve the performance of existing mapping tools. In the experiment, FMSS was compared with the average seed selection method and the optimal seed solver (OSS). Experimental results show that FMSS can select the optimal set of seeds close to OSS, and the time cost of FMSS is far lower than that of the OSS algorithm. The FMSS algorithm is more suitable for seed selection in terms of time cost and seed selection quality.

Key words: seed frequency; read-mapping; average seed; frequency merge; optimal seed

Seed-and-extend 启发式方法是近些年被广泛研究的简化大规模基因数据库搜索工作的主要方法,高效的种子选择算法可以大大加速 read mapping 的处理过程.现有基于 seed-and-extend 的映射器集成了大量有效的子选择机制,如 Hobbel Mapper[1],带有自适应种子过滤器的 GEM Mapper[2],支持 K-mer 的 FashHASH[3],最优种子选择方法 OSS[4]等.这些映射器都采用了鸽笼原理的种子过滤技术,通常把 read 划分成不重叠的多个短的片段,称为种子(seeds).种子可以用于索引基因序列来减少搜索空间大小和提高映射处理速度.将种子在基因序列中的匹配位置预先存储到种子数据库中,从而可以通过数据库的查找来快速检索匹配位置.常用的存储方法包括 Hash 表和 BWT FM-Index 方法[5-6].构建高效映射器的关键问题就是如何从基因序列中选择具有最小频率的种子集合来保证快速地检索操作.然而,从 read 中选择具有最小频率的种子集合是一项困难的工作,原因是一个任意长度的种子可以从 read 的任意位置选择,导致相应的搜索空间庞大并且随着种子数量的增长呈指数增长.

本文基于 seed-and-extend 的 read mapping 映射器提出一种基于频率的合并种子选择 FMSS (frequency based merge seed selecting)算法,该算法通过区分位置频率来确定最小频率种子,只需要扫描 read 的每一个位置一遍,然后通过合并具

收稿日期:2018-03-26
基金项目:国家自然科学基金资助项目(61772124,61332014,61401080,61402087);河北省自然科学基金资助项目(F2015501049);河北省教育厅项目(QN2014339);中央高校基本科研业务费专项资金资助项目(N150402002).
作者简介:马海涛(1977-),男,河北昌黎人,东北大学讲师,博士.

有高频率种子为低频率种子并调整种子分隔位置来减少种子集合的总体频率. 最后与目前最好的最优种子选择算法(OSS)[4,7]进行比较,验证了FMSS 的准确性和有效性.

1 问题定义

定义 1 最优种子选择问题:给定长度为 L 的 read R,参考基因序列 ref,编辑距离门限 τ. 设 $\mathrm{freq}(r)$ 表示任意字符串 r 在参考基因组序列 ref 中出现的频率. 设 $\pi \in \Pi, \pi = \{\mathrm{seed}_i \mid i = 1,2,\cdots,\tau+1\}$ 表示 read R 拆分的一组种子集合,其中 Π 表示种子选择的空间. 最优种子选择问题就是找到一组种子满足:

$$Z = \min_{\pi \in \Pi} \sum_{i=1}^{\tau+1} \mathrm{freq}(\mathrm{seed}_i). \tag{1}$$

式中,一组种子 π 是 read R 的一个划分,即顺序连接 seed_i 就会构成 read R,也就是将 read R 拆成 $\tau+1$ 段. 将 read R 拆成 $\tau+1$ 段共有 C_{L-1}^{τ} 种方法,因此,read R 的种子选择共有 C_{L-1}^{τ} 种方法. 最优选择方法就是在这些方法中选择频率和最小的作为最优的种子.

目前,常见的方法有 Average 种子选择方法[8]. 例如已知长度为 15 的 read R,编辑距离 τ 为 3,也就是将 read R 划分成 4 段,那么每段 seed 的长度就是 4,4,4,3. 然而,当数据集和种子数量较大时,这种方法的代价是十分高昂的,原因是需要考虑所有子串中的全部候选种子的组合.

文献[4]提出了 OSS 种子选择方法,它是一种动态规划算法,是目前所有基于鸽笼原理的 seed - and - extend 种子选择策略中最优种子选择方法. 最优种子选择意味着超矩阵 $\boldsymbol{F} = \sum(\tau+1)\mathrm{freq}(\mathrm{seg}^R(i))$ 上的所有 K - mer 的总匹配位置的数量是最小的. 现在要将 read R 划分为 $k = \tau+1$ 个段,使用 $H(i,j)$ 表示通过把 $R[0,j]$ 划分成 i 段的最小频数,那么 $H[k,L]$ 表示将 R 分成 k 段的最小频数. 为了计算 $H[k,L]$,通过以下动态规划公式计算矩阵 $\boldsymbol{H}^{k\times L}$ 确认方法.

$$H(1,j) = \mathrm{freq}(R[1,\cdots,j]), \tag{2}$$

$$H(i,j) = \min_{l=i-1}^{j-1}(H(i-1,l) + \mathrm{freq}(R[l+1,\cdots,j])). \tag{3}$$

但是该方法的计算复杂度高达 $O(kL^2)$,运行的时间代价很高.

为了提高种子选择方法的性能,提出一种新的算法 FMSS,该方法能够高效地选择接近最小频率的种子集合. FMSS 算法分为两个步骤:首先,算法通过计算 R 中每一个位置频率识别出频繁出现种子的位置,通过合并相邻频繁位置的高频率种子来产生具有较低频率的新种子,直到产生 x 个种子为止;然后,通过调整选择的种子之间的分隔位置来进一步减少选择种子的总频率.

1.1 种子位置频率

通常从一个给定 read 中选择具有最小频率的种子集合的困难在于随着种子数量的增加需要在一个指数空间内搜索候选种子集合[8],由于一个种子可以从 read 中的任意位置选取,观察到从 read 中的不同位置选取的种子的频率会有很大差异. 为了描述这种差异性,定义 read 中任意位置的位置频率见定义 2.

定义 2 位置频率:给定一个长度为 L 的 read R,让 pos 表示 R 上的位置,且满足 $0 \leqslant \mathrm{pos} \leqslant L-1$. 设 (pos,m) 表示通过从位置 pos 向左或向右扩展 m 个位置选取的有效子串(即 $\mathrm{pos}-m \geqslant 0$ 和 $\mathrm{pos}+m+1 \leqslant |R|$),$k$ 表示扩展子串的个数,r 代表一个整数. 求位置 pos 的频率就是基于位置 pos 进行扩展子串,求出所有子串频率之和的平均数:

$$F(\mathrm{pos},\mathrm{AVE},r) = (\sum_{m=0}^{k} \mathrm{freq}(\mathrm{pos},m))/k.$$

定义 3 种子分段频率:给定一个长度为 L 的 read R,让 pos 表示 R 上的位置,且满足 $0 \leqslant \mathrm{pos} \leqslant L-1$. $\mathrm{SEG}(i)$ 代表 read R 的分段,$R = \{\mathrm{SEG}(i) \mid i = 1,2,\cdots,\tau+1\}$,求位置 pos 的 segment 频率意味着求位置 pos 所占的分段 i 在参考序列出现的频数. 所以定义位置 pos 的分段频率为

$$F(\mathrm{pos},\mathrm{SEG},r) = \mathrm{freq}(\mathrm{SEG}(i)).$$

例 1 给定一个 read R = CCAGTGCATATACGACTT,选定参考基因序列为:CCAGTGCTAGCTGCCCTG CCAGTGCAATCCAGGCT CCAGTGCGCATGCTAGCT CCAGTGCGCTAGCTACA CCAGTGCAAATACGACTT CCAGTGCAAGCCGGGGT CCAGTGCTACGTACGAGT CCAGTGCTATTCGATAT CCAGTGCAAGCTAGCATG.

要想计算位置 5 的频率就是在位置 5 周围进行扩展子串,如图 1 所示,可以得到 $n1,\cdots,n6$ 子串,那么位置 5 的频数计算如下:

$$F(5) = [\mathrm{freq}(n1) + \mathrm{freq}(n2) + \mathrm{freq}(n3) + \mathrm{freq}(n4) + \mathrm{freq}(n5) + \mathrm{freq}(n6)]/6.$$

若将 read R 先进行平均分段,也就是分成 "CCAGT","GCATA","TACG","ACTT" 4 段. 这个分段策略的匹配 ref 的频数为:(9,1,3,1),

总频数为 14. 其中" CCAGT" 明显是一个频数高的分段. 如果利用最优分段技术，就会分成" CCAGTGCAT" ," ATAC" ," GAC" ," TT" . 这个分段策略的匹配频数为：(1，1，1，2)，总频数为 5，大大降低了种子 segment 的频数. 图 2a 表示 read R 上 18 个点的位置频数，平均分段策略和最优分段策略的频数.

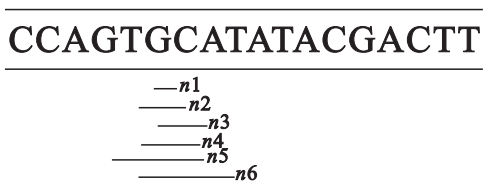

图 1　扩展位置得到的子串

Fig. 1　Substrings obtained by position extension

1.2　合并频繁种子算法

根据定义 3，$F(\mathrm{pos},\mathrm{AVE},r)$ 给出了查询字符串上每一个点处的一个频率值，这个值越高说明在该点周围的 segment 在 ref 上出现频数会越大，反之该点周围的 segment 出现频数会越小[9]. 如图 2 所示，在一个 read R 上，各个位置的 PAF 值差异很大. 利用图 2，可以将 R 分为 N 段，总频数用式(4)计算.

$$\left.\begin{aligned}&\int_0^N F(\mathrm{pos},\mathrm{AVE},r)\,\mathrm{dpos},\\&\int_0^N F(\mathrm{pos},\mathrm{SEG},r)\,\mathrm{dpos}.\end{aligned}\right\} \tag{4}$$

也就是 $F(\mathrm{pos},\mathrm{AVE},r)$ 曲线与 x 轴围成的区域的面积. 而分段后的分段总频数可以通过 $F(\mathrm{pos},\mathrm{SEG},r)$ 曲线的积分面积来计算[10].

一个很好的性质是，两段相邻的 segments 合并后的 segment 的频数会小于等于合并前的任意 segment 的频数. 那么有 3 种合并策略：①低频 segment 和低频 segement 合并，找到当前频数和最小的两个相邻 segment(如位置 17 和 18)并将它们合并，接着在合并后的分段中继续找当前频数和最小的两个 segment，直到产生 x 个种子为止. ②低频 segment 和高频 segment 合并，找到当前频数差距最大的两个相邻 segment 合并(如位置 7 和 8)，接着在合并后的分段中继续执行上面的操作，直到产生 x 个种子为止. ③高频 segment 和高频 segment 合并，通过每次找到当前频数和最大的相邻的两段 segments 进行合并(如位置 6 和 7)，逐步将所有的高频的 segments 都合并为尽量低频数的 segments，直到产生 x 个种子为止. 种子初始化分段会将种子划分为每段只有一个字符的 segment，每个 segment 的位置频率如图 2a 所示，就初始化分段对 segment 进行 3 种合并策略进行讨论.

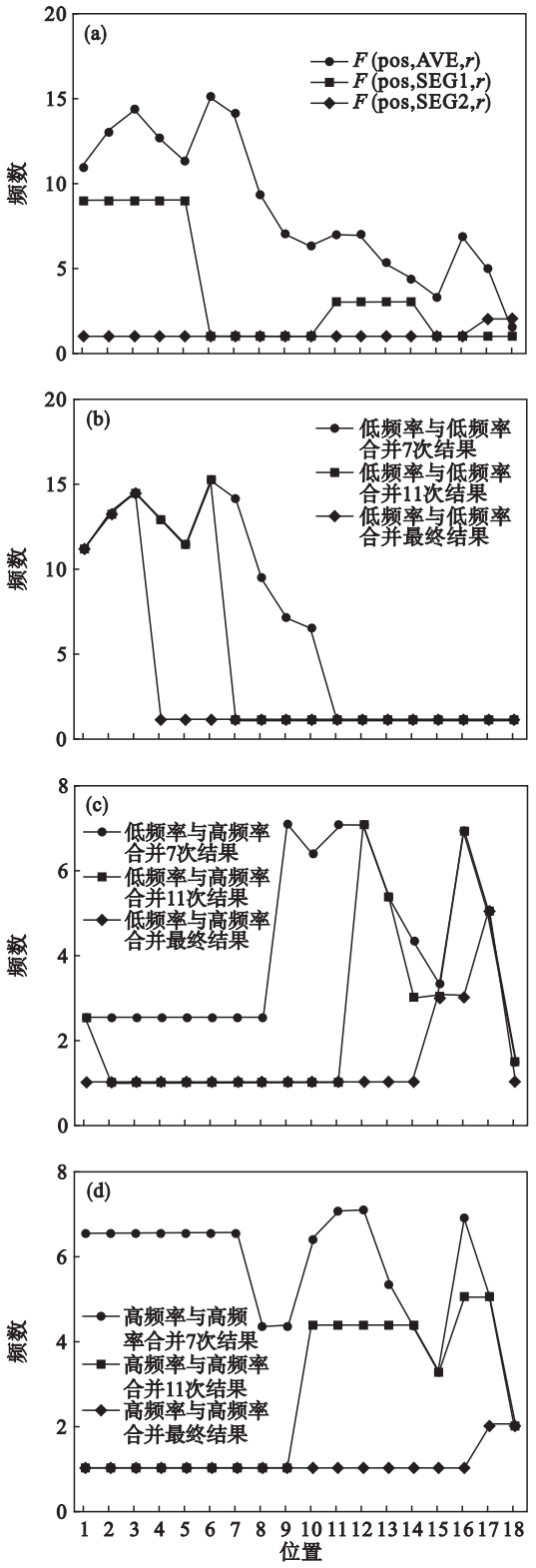

图 2　种子位置频数和三种合并策略

Fig. 2　Seed location frequency and three merge strategies

(a)—平均和最优分段策略的频率；(b)—低频率与低频率；(c)—低频率与高频率；(d)—高频率与高频率.

图 2b ~ 图 2d 分别表示对 3 种策略执行 7

次,11 次和最终合并的结果. 如图 2b 所示,可以观察到策略①是不断地将低频逐步合并,一直利用的是低频的分段,合并最终原来高频的 segment 频数依旧很高,并且高频率和低频率位置的 PAF 值差异很大,浪费了低频资源,显然这种方法是不可行的. 如图 2c 所示,策略②这样的合并方法反而会使高频和低频的差值变大,每一次都要将合并后的低频与相邻高频进行合并,浪费低频资源,时间代价也很大. 而图 2d 所示的策略③可以很快降低高频率的 segment,也没有浪费低频资源. 相比图 2b 和图 2c,图 2d 各个位置的 PAF 值差异也很小,曲线与 x 轴围成的面积也是最小的. 综上,高频 segment 与高频 segment 合并是最好的策略.

MergeSeed 算法实现了合并种子策略(算法 1). 为从 read R 中选择 x 个种子,算法 1 分为 3 个步骤. 首先,计算 read R 中全部的位置频率;然后,算法根据位置频率找出具有最高频率的相邻种子;最后,通过合并具有最高频率的相邻种子为一个新的低频率种子直到获得 read 的 x 个种子终止. 给定长度为 L 的 read, 算法 1 至多迭代 $L-x$ 次终止计算.

定理 1 算法 MergeSeed 复杂性. 从长度为 R 的 read 中确定 x 个种子,算法 1 最坏情况下需要 $O((L/k-x)\times L/k)$ 步操作. 算法首先计算 R 的全部位置频率,R 中至多有 L/k 个位置,接下来算法迭代 $O(L/k-x)$ 次来确定 x 个种子,在每一次迭代中,算法检查 $O(L-2k)$ 个种子对并进行合并种子操作. 综上,整个算法的复杂性为 $O((L/k-x)\times L/k)$.

算法 1 种子合并

输入:read R 和种子数目 x
输出:read R 的 x 个种子集合 S,频率和接近于最小频率
1:初始化种子位置 $S=\{|R|/k$ 种子的长度相等$\}$
2:**For** R 中每一个种子位置 p
3: 计算 R 的位置频率
4:**end for**
5:**For** $i=1$ to $|R|/k$
6: 从 p 位置选取频率最高的两个相邻种子 s_i 和 s_j
7:**If** $\text{freq}(s_i)+\text{freq}(s_j)>\text{freq}(s_k)$ **then**
8: 合并 s_i and s_j 称为种子 s_k 并且把 s_k 加入 S 集合
9:**else**
10:考虑下一组种子对
11:**End If**
12:**End For**
13:返回种子集合 S

1.3 调整种子分隔算法

合并相邻高频率种子之后,可以获得一个相对较低频率的种子集合,还可以通过调整已选择种子之间的分隔位置来进一步降低种子集合的总体频率. 对种子之间的频率差值,通过调整位置 p 左移 1 个位置获得新的种子对为(s_i', s_j'),则

$$\Delta(f)=(\text{freq}(s_i)+\text{freq}(s_j))-(\text{freq}(s_i')+\text{freq}(s_j')).$$

如果 $\Delta(f)$ 大于 0,可以通过调整位置 p 左移 1 个位置来把种子对(s_i, s_j)优化成(s_i', s_j'). 算法通过不断调整具有最大频率差值的相邻种子对,可以进一步降低 read 中已选择的种子集合的总体频率.

ShiftDivider 算法实现了种子移动位置策略(算法 2). 为进一步降低 R 中已选择的种子集合 S 的总体频率,算法 2 首先计算 S 中相邻种子对的频率差值,然后调整种子之间的分隔位置来减少种子对的频率和,通过至多 L 次迭代,将获得具有更低总体频率的种子集合.

定理 2 算法 ShiftDivider 复杂性. 为减少长度为 L 的 read 的 x 个种子集合的总体频率,算法 2 最坏情况下需要 $O((x/2)\times x)$ 步操作. 算法迭代检查 R 中具有最大频率差值的种子对并调整其分隔位置,至多有 $x/2$ 个种子对需要检查和 x 个位置需要调整. 当不存在两个相邻种子频率差值大于 0 时,算法终止并返回具有更低总体频率的种子集合.

算法 2 ShiftDivider

输入:x 个种子的集合 S
输出:x 个种子的新集合 S',总频率小于 S
1:初始化 $S'=S$
2:计算 S 中所有相邻种子的频率差值
3:**for** 最大的 $\Delta(f)$ 两个相邻的种子 s_i 和 s_j **do**
4:**if** $\Delta(f)>0$ **then**
5: 将 s_i 和 s_j 的分隔位置 p_{ij} 向左移动 1-bp 的位置
6: 移除种子 s_i 和 s_j 并且将新种子 s_i' 和 s_j' 加入到 S' 中
7: 考虑其他最大频率差值的种子
8:**else**
9: 返回种子集合 S'
10:**end if**
11:**end for**
12:返回种子的集合 S'

2 实验结果

本文比较了提出的算法 FMSS、采用平均长度种子选择策略 AVE 最优种子选择算法 OSS 的

准确性和有效性. 准确性采用给定 read 的选取种子集合的总体频率的平均值度量,有效性采用选取种子集合的平均时间代价度量. 全部算法采用 C + + 语言实现,运行在一台 CPU 为 Intel(R) Core(TM) i7 - 2600 3.40 GHz, 主存大小为 32 GB 的 PC 机上,Ubuntu 16 的操作系统.

实验数据来源于真实的公开 DNA 数据库和 DBLP 数据库,随机选取 4 000 条 read 作为查询串,read 的长度为 108 bp. 在 DBLP 上选取 200 条片段作为查询,每条长度为 78 bp. 将每条 read 在平均长度种子选择策略 Average,最优种子选择算法 OSS 和本文提出的 FMSS 3 种策略下进行分段. 图 3 表示两个数据库下每条 read 在 3 种策略下分成的 segment 在 4 000 条 read 中出现的频数和. 其中,“ - a”表示 AVE 方法,“ - b”表示 OSS 方法,“ - c”表示本文提出的 FMSS 方法. 可以发现,对于不同种子数量,在 DNA 数据库下的 FMSS 选择的平均种子频率总和接近于算法 OSS,但远远小于算法 AVE. 在 DBLP 数据库的 FMSS 选择的平均种子频率总和远远小于 OSS 和 AVE.

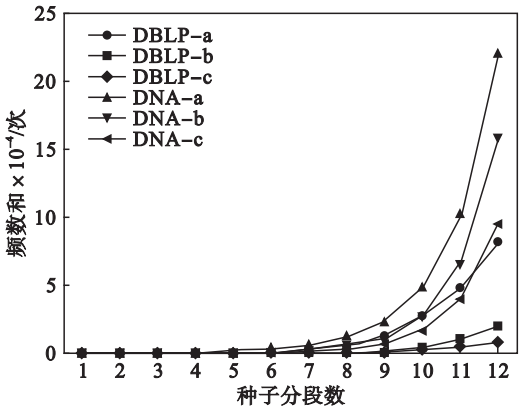

图 3　实验结果

Fig. 3　Experimental results

表 1　运行时间

Table 1　Running time　　s

段数	DBLP			DNA		
	AVE	FMSS	OSS	AVE	FMSS	OSS
1	0.03	0.25	3.85	0.09	2.62	44.57
2	0.02	0.28	8.37	0.10	3.10	86.86
3	0.02	0.34	13.71	0.11	3.23	127.92
4	0.02	0.38	19.80	0.12	3.26	167.20
5	0.03	0.41	26.25	0.13	3.23	207.44
6	0.03	0.44	33.28	0.14	3.05	245.66
7	0.03	0.47	41.38	0.14	2.88	282.33
8	0.04	0.55	50.46	0.14	2.72	320.26
9	0.04	0.54	0.54	0.14	2.40	354.29
10	0.04	0.56	70.33	0.14	2.22	388.86
11	0.50	0.59	80.67	0.14	2.24	422.70
12	0.05	0.61	92.00	0.13	2.02	455.31

同时,3 种策略的运行时间如表 1 所示,可以发现在 DBLP 数据库下 FMSS 消耗的运行时间基本与算法 AVE 接近,远远低于 OSS 算法. 在 DNA 数据库下 FMSS 消耗的运行时间高于算法 AVE,远远低于 OSS 算法. 表明 FMSS 能够用更少的时间,来选择接近最优频率的种子集合,因此,可用于处理较大规模的 read mapping 问题.

3　结　　论

1) 本文提出了一种有效的基于频率的种子选择算法 FMSS,解决了 read mapping 过程中低频数种子的选择问题.

2) 实验结果表明 FMSS 生成种子的频数与最优种子频数相差很小. 同时 FMSS 算法的时间花费远远低于最优种子选择算法 OSS. 表明 FMSS 算法更适用于处理较大规模的 read mapping 问题.

参考文献:

[1] Paolo F, Giovanni M, Veli M, et al. Compressed representations of sequences and full-text indexes [J]. *ACM Transactions on Algorithms*, 2007, 3(2): 1 - 25.

[2] Xin H Y, Nahar S, Zhu R, et al. Optimal seed solver: optimizing seed selection in read mapping [J]. *Bioinformatics*, 2016, 32: 1632 - 1642.

[3] Flicek P, Birney E. Sense from sequence reads: methods for alignment and assembly [J]. *Nature Methods*, 2009, 6(1): 6 - 12.

[4] Bin M, John T, Ming L. Patternhunter: faster and more ensitive homology search [J]. *Bioinformatics*, 2002, 18: 440 - 445.

[5] Xin H Y, Lee D, Hormozdiari F, et al. Accelerating read mapping with FastHASH [J]. *BMC Genomics*, 2013, 14(1): 1 - 13.

[6] Santiago M, Michael S, Roderic G, et al. The GEM mapper: fast, accurate and versatile alignment by filtration [J]. *Nature Methods*, 2012, 9(12): 1185 - 1187.

[7] Suzuki S, Kakuta M, Ishida T, et al. Faster sequence homology searches by clustering subsequences [J]. *Bioinformatics*, 2015, 31: 1183 - 1190.

[8] Hao W, Jeffrey X, Can L. String similarity search: a Hash-based approach [J]. *Transactions on Knowledge & Data Engineering*, 2018, 99: 170 - 184.

[9] Heng L, Richard D. Fast and accurate read alignment with burrows-wheeler transform [J]. *Bioinformatics*, 2009, 25: 1754 - 1760.

[10] Ahmadi A, Behm A, Honnalli N, et al. Hobbes: optimized gram-based methods for efficient read alignment [J]. *Nucleic Acids Research*, 2012, 40(6): 41.