

一种具有精确边界的重复体识别算法

霍红卫 白 帆

(西安电子科技大学计算机学院 西安 710071)

摘 要 当前大部分重复体识别算法不是依靠于已经标识的重复体数据库就是定义重复体为两个最大长度的相似序列,而没有一个严格的定义来平衡重复体的长度和频率.针对这些问题文中提出了一种基于局部序列比对算法 BLAST 变型且支持空位的快速识别重复体的 RepeatSearcher 算法.算法通过定义重复体的精确边界运用逐步扩展调和序列来识别重复体.算法使用 C.briggsae 基因组序列作为测试对象,并与当前通用的重复体识别算法 RECON 以及新近的识别算法 RepeatScout 做了比较分析.结果表明 RepeatSearcher 使每一条重复体序列具有了精确的边界,而且相对其它算法在没有损失精度的情况下,缩短了算法的运行时间.

关键词 调和序列;重复体识别;精确边界;BLAST;RepeatSearcher

中图法分类号 TP18

An Algorithm for Identification of Repeats with Accurate Boundaries

HUO Hong-Wei BAI Fan

(School of Computer Science and Technology, Xidian University, Xi'an 710071)

Abstract Most existing methods of repeat identification either rely on annotated repeat databases or limit repeats to pairs of similar sequences that are maximal in length. And there is no an exact definition to correctly balances the importance of the length and the frequency. For these shortages, a fast method for repeats identification of repeat families via extension of consensus seed is proposed in this paper, which enables a rigorous definition of repeat boundaries and is based on the variant of BLAST algorithm. The known C.briggsae is used for testing the RepeatSearcher. RepeatSearcher is compared with RECON, the most popular repeats identification algorithm, and the newly developed RepeatScout. The experimental results indicate that RepeatSearcher has more accurate boundaries for each repeats, and the time of RepeatSearcher is reduced as compared with other methods with guaranteed accuracy.

Keywords consensus sequence; repeat identification; accurate boundaries; BLAST; RepeatSearcher

1 引 言

在生物信息学中重复体识别是分析基因组序列的一个重要组成部分.重复体在很大程度上决定了基因组的进化方向,而且在实际应用中重复体被用

来确定一个基因组的家族关系.在真核生物基因组中重复体 DNA 占据了非常重要的地位,据国际人类基因组协会统计,大约 50% 的人类基因组已经被识别为重复体 DNA.

当前大部分重复体家族识别算法均从相似对集合开始构建重复体集合,比如重复体家族识别算法

REPuter^[1]. 早期的单联聚类法首先把相似对集合中重叠的子串合并起来, 然后使用这些相似对把合并的子串聚合成重复体家族. RepeatFinder 算法^[2]和 RECON 算法^[3]就是根据这种方法改进而得. 使用高分相似对序列作为构建重复体家族集合的基础有两个缺点: (1) 重复体边界定义的问题给分类相关元素为重复体带来了很大的困难. (2) 对于大型的重复体丰富的基因组来说, 构建相似对是一个计算密集的艰巨任务. 比如人类基因组中的 Alu 重复体重复频率为 10^6 , 可以构建约 10^{12} 个相似对, 使用现有方法构造相似对集合是不切实际的^[3]. 使用迭代算法可以在一定程度上解决这个问题. 但迭代法以小规模基因组为样本, 然后与已经识别出的重复体进行掩模对准, 逐渐扩大基因组样本的规模, 从而使得迭代算法从基因组的小样本识别出来的重复体在整体上是很不精确的, 已经识别的子重复体将有可能被从新的重复体集合中排除, 最终导致重复体家族链断开. 算法 RepeatGluer^[4]是另一种把相似对合成重复体家族的方法, 类似于 RECON 算法, 将目标序列或者基因组与自身相比较来识别不同位置上面的局部比对结果, 不同之处只是在归类重复体家族的方法. RepeatScout 算法^[5]改进了这些方法, 利用对种子序列的扩展, 动态地推断出其在基因组中的比对来识别重复体. 另一个值得关注的算法是 PILER^[6], 算法以其敏感性为代价提高识别不同类型重复体的特异性.

本文提出一种具有精确边界的重复体快速识别算法 RepeatSearcher, 算法把高频率的 *l-mer* 种子作为一组重复体家族的核心序列, 然后通过改进 BLAST 算法^[7], 贪婪地向两边扩展每一个种子, 并计算出调和序列. 本文使用知名的 C.briggsae^[8]基因组序列测试了 RepeatSearcher 算法, 并与 RECON 算法和 RepeatScout 算法做比较, 最后通过 RepeatMasker 分析几个算法的输出. 结果表明 RepeatSearcher 相对这两种算法在没有损失精度的情况下缩短了算法的运行时间, 识别出来的每个重复体序列都具有精确的边界.

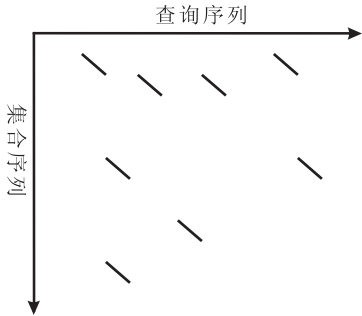
2 BLAST 算法

BLAST 算法是一种查找长为 W 的字串的启发式搜索方法(在 blastp 中 W 值默认为 3), 当与查询序列比对时基于打分矩阵对位打分. 数据库中打分为 T 以上的字串在两个方向被扩展, 目的是找出无空位的局部最优比对, 或者找出分值至少为 S

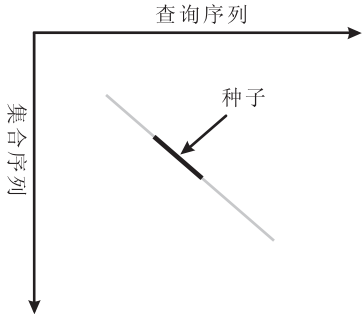
或小于某一给定阈值 E 的高分值片段配对 (High Score Segment Pair, HSP). BLAST 输出满足这些标准的 HSP.

2.1 BLAST 算法过程

算法从一个输入 DNA 序列 S_q 和比对象数据库开始, 先从数据库中取出一条序列 $S_i (1 \leq i \leq n)$, 然后把 S_i 分解为长度为 W 的片断 $S_{i,j} (1 \leq j \leq |S_i|)$. 在输入序列 S_q 中找出和片断 $S_{i,j}$ 完全匹配的片断 $S_{q,k}$, 记录下匹配的位置 k . 然后将匹配的片断 $S_{q,k}$ 根据打分矩阵逐个和 $S_{i,j}$ 进行比对, 并根据相似度逐渐把 $S_{q,k}$ 和 $S_{i,j}$ 向左右两边延伸, 每延伸一个字节则计算一次局部比对的 HSP 值, 并记录最高分值的位置, 得到最大 HSP 值, 直到在一定范围内这个分值不再增加为止, 则这次比对结束. 对于每一个 W 片断 $S_{i,j} (1 \leq j \leq |S_i|)$ 依次进行上述过程, 得到最大 HSP 值, 这样就找到了所有共享相同片断的重复体序列, 最后将这些重复体序列组成最高分值片断组. 图 1 展示了 BLAST 算法的 3 个步骤.



(a) 寻找短的高分命中序列(步骤 1 和 2)



(b) 扩展短的高分命中序列(步骤 3)

图 1

BLAST 算法的比对过程基于改进的 Smith-Waterman 的局部比对方法, 在查询序列和集合序列之间寻找最长的相似对. 算法通过设置比对分值来确定相似对的边界. 基本 BLAST 算法不支持空位的插入, 这样保证了算法有较高的运算速度, 但却损失了结果的精度.

3 RepeatSearcher 算法

RepeatSearcher 算法使用 BLAST 算法中的 3 个主要步骤:分割一个基因组序列 S , 为 l -mer 的短种子序列, 记录下来这些种子在序列 S 中的位置, 然后比对具有相同种子的序列, 并逐步向两边扩展直到得到最大比对分值. 这样就可以计算出夹杂着一些伪重复体元素的序列. 怎样精确地判断出每一条重复体序列的精确边界和找出家族的调和序列将是算法需要解决的问题.

针对重复体识别问题, RepeatSearcher 对 BLAST 基本算法进行了扩展, 改双序列局部比对为多序列局部比对, 在 BLAST 算法中加入空位罚分机制. 鉴于局部比对问题的特点, 算法限制了空位的插入范围, 这样加快了比对的速度同时也避免了 BLAST 算法不支持空位带来的不精确性. 空位限制的范围大小会影响到比对的速度, 越小的范围比对速度越快.

3.1 目标函数定义

首先定义多序列局部比对的目标函数. 这里称长度为 l 而且高度重复的序列片断为 l -mer 种子. 设 S_1, S_2, \dots, S_n 表示 DNA 序列中包含相同 l -mer 种子的子序列, 这些子序列都通过 l -mer 种子向左右做了相似性扩展. S_1, S_2, \dots, S_n 的子串 R_1, R_2, \dots, R_n 代表每一个具有精确边界的重复体序列. Q 表示调和序列, 它使得式(1)取最大值. 这里 $A(Q; S_1, S_2, \dots, S_n)$ 是多序列匹配优先的比对目标函数, $|Q|$ 表示调和序列 Q 的长度, c 表示重复体要求的最小出现频率, μ 表示匹配得分. 由此可知 R_1, R_2, \dots, R_n 是 S_1, S_2, \dots, S_n 比对时目标函数 $A(Q; S_1, S_2, \dots, S_n)$ 取得最优值时的子串. 如式(1)所示, 目标 $A(Q; S_1, S_2, \dots, S_n)$ 等于 Q 分别和 S_1, S_2, \dots, S_n 比对得分函数最大值之和减去调节因子 $c\mu|Q|$:

$$A(Q; S_1, S_2, \dots, S_n) = \left[\sum_k \max\{s(Q, S_k), 0\} \right] - c\mu|Q| \quad (1)$$

在一个重复体家族中, 重复体家族为重复体子串所共享, 这些重复体子串各自有各自的边界. 我们通过目标函数 $A(Q; S_1, S_2, \dots, S_n)$ 来定义每一个重复体子串的边界, 使得目标函数取最大值时的所有重复体序列的边界位置就是各重复体序列的边界. 下面给出比对得分函数 $s(Q, S_k)$ 的计算方法.

定义比对得分函数 $s(Q, S_k)$ 如下^[5]:

$$f(i, 0) = \max(-\gamma_i, 0), f(0, j) = 0;$$

$$f(i, j) = \max \begin{cases} f(i-1, j-1) + \mu_{i,j} \\ f(i-1, j) - \gamma \\ f(i, j-1) - \gamma \\ -\gamma |i-j| \end{cases};$$

$$s(Q, S) = \max \begin{cases} f(i, j), & i = |Q| \\ f(i, j) - \gamma|Q|, & i < |Q| \end{cases} \quad (2)$$

μ_{ij} 表示 Q_i 和 S_j 的匹配得分, γ 表示一个固定空位罚分值.

3.2 比对优化过程

3.1 节定义了多序列局部比对的目标函数, 并且给出了比对得分函数的基本计算方法, 也就是限制了调和序列的扩展方法和整个家族的扩展范围. 以下优化多序列局部比对过程, 并确定重复体家族中每一条重复体序列的边界.

我们可以扩展 BLAST 算法来优化多序列局部比对过程. 从基因组中提取出高频率出现的 l -mer 种子, 设 S_1, S_2, \dots, S_n 是共享这个种子的序列片断, 这些序列两边已经超出了每个重复体的边界. 令 Q_0 等于 l -mer 种子, 然后贪婪地向左右扩展 Q_0 , 每次扩展一个碱基, 每次扩展的碱基必须使目标函数 $A(Q; S_1, S_2, \dots, S_n)$ 取最大值. Q_0 在第 k 次扩展时, 针对每一个碱基 N 计算 $A(Q_k N; S_1, S_2, \dots, S_n)$ ($N \in \{A, T, C, G\}$), $Q_k N$ 表示 Q_k 扩展一个碱基 N (Q_k 的左边或右边) 的序列. Q_{k+1} 表示 $Q_k N$ 使得目标函数 $A(Q_k N; S_1, S_2, \dots, S_n)$ 取得最大值时 $k+1$ 次循环的结果. 结合式(2)可以看出第 $k+1$ 次循环的输入就是第 k 次循环的输出. 算法每次将 Q_0 扩展一个字节, 为了限制运行时间, 算法定义了一个阈值 I , 当 Q 扩展了 I 次后还没有提高目标函数 $A(Q; S_1, S_2, \dots, S_n)$ 的值就停止扩展过程.

为了在局部序列比对时支持空位的插入, 算法在扩展每个碱基的同时参考其周围 $[-b, b]$ 一定范围内的碱基. 首先假设当前序列 S_i 已经不可以扩展, 根据式(2)使用比对分值减去不完全匹配罚分, 记录下这个分值. 在 $-b \sim b$ 范围内, 首先比较调和序列 Q 的候选碱基 N 和空位, 空位个数从 1 到与当前碱基 N 匹配的位置和 $-b$ 的差值, 记录下来比对最大分值. 然后比较调和序列 Q 的候选碱基与序列 S_i 的当前碱基, 根据匹配得分计算出比对值. 最后比较这几个值, 取出一个最大比对值. 于是比对得分函数可以修改为式(3):

$$s(Q_k N, S_i) = \max_{f \in [-b, b]} s_f(Q_k N, S_i) \quad (3)$$

使用上述方法我们计算出每一个 S_i ($1 \leq i \leq n$) 和 Q 比对分值 $s(Q, S_i)$. 这里使用目标函数取最大值的比对作为选择调和序列当前位置碱基的标准. 这样

的比对方法避免了 BLAST 算法不支持空位的问题,增加了比对结果的精确性。

3.3 重复体序列边界的确定

确定调和序列的扩展范围后,在这个范围内使每一条长度小于调和序列的重复体家族序列和调和序列比对,如果比对分值还可以增加则增加比对分值到最大,这个比对范围只需要在重复体序列当前边界开始,到调和序列的最边缘为止,最后确定下来这个最大比对位置.计算出调和序列以及重复体家族序列和调和序列的比对分值之后,算法就可以通过 S_i 与调和序列 Q 的比对分值来确定 S_i 的边界.边界函数可表示为式(4):

$$s_{\text{BOUND}}(Q, S_i) = \max_{j \in [L_{\text{max}}, L_{\text{seed}}]} s_j(Q, S_i) + \max_{k \in [R_{\text{seed}}, R_{\text{max}}]} s_k(Q, S_i) \quad (4)$$

边界比分值应该取左右扩展比分值之和的最大值。

根据上述过程,给定一个基因组和一组高频率的 l -mer 种子,算法得到代表一个重复体家族的调和序列.我们使用上述过程去计算每一个高频率的 l -mer 序列来识别给定基因组中所有的重复体家族序列.但是计算每一个高频率的 l -mer 序列非常花费时间,而且是多余的.因为一个重复体家族会由于变异产生大量的高频率 l -mer 序列,其数目很可能大于重复体的长度.为了解决这个问题,算法在每次计算出一个重复体家族的调和序列后记录下来调和序列的所有 l -mer 种子,然后判断原序列中的 l -mer 种子是否在调和序列中出现,假设出现的次数为 x ,那么就将原序列中相应的种子频率减去 x ,这样就刷新了原序列的种子频率。

3.4 算法描述

RepeatSearcher 算法的思想是把种子向两边扩展,扩展过程每次向左或向右扩展一个字节,计算每个碱基在这个位置上的最大比分值,然后取值最大的那个碱基将作为这个位置上的候选碱基,将其插入到调和序列中.假如在向右扩展时,如果 S_k 和 Q 比对时比分值一直没有提高则将比分值取最高值的点作为 S_k 的右边的边界,用同样方法也可以确定 S_k 左边的边界.由此可以在比对过程中确定每个 l -mer 种子周围的边界.计算出一个重复体家族中的调和序列后,为了避免重复查找,我们根据这个调和序列刷新 l -mer 种子表中种子的频率。

算法伪代码的描述见算法 1。

算法 1.

Algorithm RepeatSearcher(S, l, m, b)

Input: string S of length n , seed length l , minimum repeat frequency m , banding constraint b

Output: every repeat family's consensus sequence

1. $l\text{-mer_table} \leftarrow \text{Build_}l\text{-mer_table}(S, l, m)$
2. $l\text{-mer_seed} \leftarrow \text{most frequency seed in } l\text{-mer_table}$
3. while (there exists $l\text{-mer_seed}$ satisfying the condition) do //改为判别表是否为空
4. $\text{Extend_right}(S, l\text{-mer_seed}, b, Q)$
5. $\text{Extend_left}(S, l\text{-mer_seed}, b, Q)$
6. if $(Q.\text{right} - Q.\text{left} \geq \text{LENGTH})$
7. then write Q into the output file
8. $\text{Updata_}l\text{-mer_table}(Q, l\text{-mer_table})$
9. else
10. $\text{Updata_}l\text{-mer_table}(Q, l\text{-mer_table})$;
11. return output file

算法开始创建 l -mer 种子列表,创建过程中,扫描输入的基因组序列中所有长度为 l 的序列片断,依次判断每一个长度为 l 的片断的出现频率和出现位置.最后将这些频率信息和位置信息存入哈希表中.种子序列刷新过程 $\text{Updata_}l\text{-mer_table}(Q, l\text{-mer_table})$ 根据 3.3 节最后部分的方法来刷新种子频率。

比分值是扩展过程的主要依据.它可以通过多重循环来实现,比分值 $s(Q_k N; S_i)$ 可以很快地通过上一次的计算结果 $s(Q_k; S_i)$ 计算出来.令比对的偏移量为第 k 次循环插入的数目减去删除的数目,接着计算和存储每一个偏移 $f(f \in [-b, b])$ 的比分数 $s_f(Q_k; S_i)$,这里使用带状态动态规划,使 b 作为比对带宽的约束值,空位的插入只在这个范围内实现. $s_f(Q_k N; S_i)(f \in [-b, b])$ 可以使用上一次循环的结果按照式(3)计算出来.在程序计算过程中相应地减小 b 值可以提高算法的计算速度.算法 2 给出了计算目标函数扩展种子序列过程的伪代码。

算法 2.

Subroutine $\text{Extend_right}(S, l\text{-mer_seed}, b, Q)$

输入: 串 S 的长度 n , 所选择的 l -mer 种子, 带宽约束 b
输出: 右端扩展的调和序列

1. for $y \leftarrow 1$ to L do
2. for $i \leftarrow 1$ to 4 do
3. for $j \leftarrow 1$ to N do
4. for $k \leftarrow -b$ to $+b$ do
5. $\text{bestscore} = \max_{\text{offset}} (s(Q_y i, S_j))$
6. $\text{bestscore}_i \leftarrow \text{bestscore}_i + \text{bestscore}$
7. $\text{best}_i \leftarrow \max \text{ the bestscore}_i$
8. $\text{master}[y] \leftarrow \text{best}_a$
9. if the total score has not increased after 200 intervals then break
10. $Q \leftarrow \text{master}$

过程 $\text{Extend_left}(S, l\text{-mer_seed}, b, Q)$ 和过程 $\text{Extend_right}(S, l\text{-mer_seed}, b, Q)$ 的算法过程相似,区别只是从种子的左边界自右到左扩展。

4 实 验

4.1 实验结果

程序 RepeatSearcher 在 FreeBSD 环境上通过 C 语言实现. Bao 和 Eddy 建议种子序列长度 $l = \lceil \log_4 L + 1 \rceil$, 这个值在多次实验中都已经经过了验证. l -mer 种子频率阈值和重复体频率阈值均为 3. 两个元素匹配则得 1 分, 不匹配得 -1 分, 插入一个空位得 -5 分, 一个序列不完全匹配则得 -20 分, 比对带宽约束值 b 设为 5. 假定在向两边扩展种子时如果扩展了 200 次而没有提高目标函数 $A(Q; S_1, S_2, \dots, S_n)$ 的值就停止扩展过程. 因为几乎所有重复体家族的调和序列长度在 50~10000 之间, 所以结果中不包含调和序列长度小于 50 的重复体. 本实验测试机器为 Intel 至强 3.66GHz, 8GB 内存的服务器.

实验使用 108M 的 C.briggsae^[8] 基因组作为测试 RepeatSearcher 数据. 同时计算 RECON 算法、RepeatScout 算法的输出结果, 然后把这 3 个算法的输出结果作为 RepeatMasker 程序的输入, 比较它们的性能. 因为 RECON 算法的输出只包含重复体出现次数超过 10 次的重复体, 所以 RepeatSearcher 算法和 RepeatScout 把输出结果中重复次数小于 10 的重复体序列去掉. 表 1 列出了这 3 种算法的运行时间. 表 2 和表 3 列出了使用 RepeatMasker 程序对这 3 种算法的分析结果. 其中时间单位为分钟(min).

表 1 RepeatSearcher、RECON 和 RepeatScout 算法的运行结果

方法	重复体 家族个数	重复体 家族大小/MB	运行时间/ min
RepeatSearcher	1391	0.65	340
RECON	723	0.43	3370
RepeatScout	1401	0.69	455

表 2 RepeatMasker 在 C.briggsae 基因组上分析 RepeatSearcher 和 RECON 输出的结果

	RepeatSearcher 覆盖的大小/MB	RepeatSearcher 没有 覆盖的大小/MB	总大小/ MB
RECON 覆盖的大小/MB	22.9	2.1	25.0
RECON 没有 覆盖的大小/MB	4.7	78.7	83.4
总大小/MB	27.6	80.8	108.4

表 3 RepeatMasker 算法在 C.briggsae 基因组上分析 RepeatSearcher 和 RepeatScout 输出的结果

	RepeatSearcher 覆盖的大小/MB	RepeatSearcher 没有 覆盖的大小/MB	总大小/ MB
RepeatScout 覆盖的大小/MB	26.8	1.5	28.3
RepeatScout 没有 覆盖的大小/MB	0.8	79.3	80.1
总大小/MB	27.6	80.8	108.4

4.2 结果分析

基于表 1~表 3 的实验结果, 我们分别从算法识别重复体的结果和运行时间两个方面对实验结果进行分析.

重复体识别结果. RepeatSearcher 算法在 C.briggsae 基因组中识别出来了 1391 个重复体家族, 远多于 RECON 的识别结果 723 个重复体家族, 而且和 RepeatScout 的 1401 个重复体家族结果相当. 从使用 RepeatMasker 重复体分析工具分析 3 个算法的结果可以看出, RepeatSearcher 总共覆盖了 27.6 MB 的重复体序列, 而 RECON 和 RepeatScout 各覆盖了 25 MB 和 28.3 MB. RepeatSearcher 的结果明显优于 RECON 的结果, 而和 RepeatScout 的结果相差仅有 0.7 MB. 根据算法的特点和结果分析, 这些区别在于重复体家族之间的区别和重复体家族序列的边界区别. 从总体上来看, 因为 27.6 与 0.65 的比值稍大于 28.3 与 0.69 的比值, 所以 RepeatSearcher 算法的敏感度略高于 RepeatScout 算法, 而远高于 RECON. 而且 RepeatSearcher 算法找出的每一个重复体家族序列均有精确的边界.

运行时间. RECON 算法直接对整个基因组计算将会导致算法无法运转下去, 所以这里将整个基因组分割为大小为 3MB(测试大小)大小的序列, 然后逐个计算, 并将总时间计算出来. 从表 1 可以看出, RepeatSearcher, RECON 和 RepeatScout 的运行时间为 340 min, 3370 min 和 455 min. 从运行时间上 RepeatSearcher 算法远远优于其它两种算法, 这也证明了算法在时间复杂度上有所改进. 下一步算法将在如何提高算法的精度上做改进, 并将引入概率统计方法来进一步提高算法的效率.

5 结 论

本文根据当前重复体识别算法中存在的问题提出了一种基于 BLAST 变型算法且具有精确边界的重复体快速识别算法 RepeatSearcher. 算法在计算多序列局部比对的同时定义了每一个重复体序列的边界条件, 保证了每一个重复体序列具有精确的边界. 算法结合限制范围的空位插入方法, 既提高了运行速度也增加了结果的精确性. 在识别重复体的同时算法计算出来了重复体家族的调和序列. 实验结果表明 RepeatSearcher 算法的运算速度相对其它算法得到了很大的提高而且也保证了非常高的精确性, 是一种高效的重复体识别算法.

参 考 文 献

- [1] Kurtz S, Schleiermacher C. REPuter: Fast computation of maximal repeats in complete genomes. *Bioinformatics*, 1999, 15(5): 426-427
- [2] Volfovsky N, Haas B J, Salzberg S L. A clustering method for repeat analysis in DNA sequences. *Genome Biology*, 2001, 2(8): research0027.1-0027.11
- [3] Bao Z, Eddy S R. Automated de novo identification of repeat sequence families in sequenced genomes. *Genome Research*, 2002, 12(8): 1269-1276
- [4] Pevzner P A, Tang H, Tesler G. De novo repeat classification and fragment assembly. *Genome Research*, 2004, 14(9): 1786-1796
- [5] Price A L, Jones N C, Pevzner P A. De novo identification of repeat families in large genomes. *Bioinformatics*, 2005, 21(Supplement): 351-358
- [6] Edgar R C, Myers E W. PILER: Identification and classification of genomic repeats. *Bioinformatics*, 2005, 1(1): 1-7

- [7] Altschul S F, Gish W, Miller W, Myers E W, Lipman D J. Basic local alignment search tool. *Journal of Molecular Biology*, 1990, 215(3): 403-410
- [8] Stein L D, Bao Z, Blasiar D, Blumenthal T, Brent M R, Chen N, Chinwalla A, Clarke L, Clee C, Coghlan A et al. The genome sequence of *Caenorhabditis briggsae*: A platform for comparative genomics. *PLoS Biology*, 2003, 1(2): 166-192
- [9] Altschul S F, Madden T L, Schaffer A A, Zhang J, Zhang Z, Miller W, Lipman D J. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 1997, 25(17): 3389-3402
- [10] Myers E. W. A four russians algorithm for regular expression pattern matching. *Journal of ACM*, 1992, 39(4): 430-448
- [11] Deepak Sharma, Biju Issac, Raghava G P S, Ramaswamy R. Spectral Repeat Finder (SRF): Identification of repetitive sequences using Fourier transformation. *Bioinformatics*, 2004, 20(9): 1405-1412
- [12] Guillaume A, Frederic B, Eduardo P C Rocha, Alain Vian, Eric Coissac. Repseek, a tool to retrieve approximate repeats from large DNA sequences. *Bioinformatics*, 2006, 23(1): 119-121



HUO Hong-Wei, born in 1963, Ph. D., professor. Her research interests include algorithm design and analysis, bioinformatics algorithms, and parallel computing.

BAI Fan, born in 1980, master. His research interests include bioinformatics algorithms and algorithm design.

Background

Repetitive DNA comprises a significant fraction of eukaryotic genomes, e. g. about 20% of *Caenorhabditis elegans* and *Caenorhabditis briggsae* genomes and about 50% of the human genome have been identified as repetitive DNA. Repeat identification is a critical part of the analysis of a newly sequenced genome, both because repeats drive genome evolution in diverse ways and because of a pragmatic need for thorough repeat masking prior to performing homology searches.

Multiple genome projects have generated large volumes of biological data. Novel techniques are called for to process these data and extract useful information. Several fundamental questions remain unanswered in Genomics. Some of the most intriguing questions are the roles of non-coding DNA, and in particular the role of repetitive sequences.

Most existing algorithms for constructing repeat families start with a set of pairwise similarities to construct a set of repeats, such as WU-BLAST or REPuter. They only report all pairs of repeated segments. The early single linkage clustering approach first merges overlapping substrings occurring in the set of pairwise similarities, and then uses the pairwise similarities to combine the merged substrings into repeat families. The algorithms, RepeatFinder and RECON, have

significantly extended and improved the single lineage clustering approach. The RepeatGluter algorithm is the distinct method to glue pairwise similarities into repeat families, in which A-Bruijn graph is proposed to explore the block structure of the repeat sequences. The PILER algorithm is also the deserved algorithm, which achieves high specificity in identifying different types of repeats at the cost of some sensitivity. There are two drawbacks using a set of pairwise similarities as the basic to construct the set of repeat families. First it is a tough work to define the true element boundary. Second, exact constructing a set of pairwise similarities involves computationally intensive tasks for the large repeat-rich genomes. For example, there are over 10^6 copies Alu repeats in human genome and these copies can lead to about 10^{12} pairwise alignments, thus constructing such a set of pairwise similarities makes it computationally infeasible.

A fast method for repeats identification of repeat families via extension of consensus seed is proposed in this paper, which enables a rigorous definition of repeat boundaries and is based on the variant of BLAST algorithm. the scoring function is given and the exact boundary function is defined. The experimental results indicate the feasibility of the given algorithm.