参赛队员姓名：金川杨，王禹听，李滕昊

中学：南京外国语学校

省份：江苏省

国家/地区：中国

指导教师姓名：史钋镭

论文题目：一种 Sanger 测序数据杂合突变识别算法

一种 Sanger 测序数据杂合突变识别算法

金川杨 王禹听 李滕昊
南京外国语学校 南京 江苏 210000 中国

摘要

【目的】

本研究介绍了一种二倍体同源染色体 Sanger 测序中自动识别杂合突变的算法，通过提高自动化检测的准确性和速度，达到提升基因分析工作的效率的目的。

【方法】

采用该算法的系统已经使用 ES6 语言实现，可在浏览器环境中运行，以二倍体 PCR 片段测序的.ab1 格式文件作为源数据，分离出碱基 G、A、T 和 C 在各个位点的包络曲线，通过基于计算几何方法的算法进行典型特征提取，得到 SNP 初步识别结果，最后利用 Phred 分数结合动态规划算法找到噪声区从而修订测序噪声带来的干扰，得到高度精确的 SNP 识别结果。

【结果】

系统在浏览器环境下无需安装、使用方便、系统界面友好、运行稳定、算法早先相关工作中非神经网络算法、无需训练结果稳定

【结论】

本文所构建的 SNP 自动检测系统使用方便，准确性高，不需参考序列，可用于二倍体 PCR 片段测序中 SNP 的高效检测。

关键词：Sanger 测序；杂合突变；包络面积；Phred 分数；动态规划算法

目录

一、研究的生物学背景

本章就本文中所述算法的相关生物学背景进行了陈述。

1. DNA 测序背景

DNA 测序（DNA sequencing，或译 DNA 定序）是指分析特定 DNA 片段的碱基序列，也就是腺嘌呤（A）、胸腺嘧啶（T）、胞嘧啶（C）与鸟嘌呤的（G）排列方式，用于确定重组 DNA 的方向与结构，对突变进行定位、鉴定和比较研究。

第一代测序技术即 Sanger 测序法，该技术直到现在依然被广泛使用，一次可获得一条长度在 700～1000 个碱基的序列。

第二代测序是对传统 Sanger 测序的革命性变革，其解决了一代测序一次只能测定一条序列的限制，一次运行即可同时得到几十万到几百万条核酸分子的序列。第二代测序技术虽然测序的通量大大增加，但是其获得单条序列长度很短，想要得到准确的基因序列信息依赖于较高的测序覆盖度和准确的序列拼接技术，因此最终得到的结果中会存在一定的错误信息。

第三代测序技术也称为单分子测序技术，该技术在保证测序通量的基础上，对单条长序列进行从头测序，能够直接得到长度在数万个碱基的核酸序列信息。

3. 第一代测序（Sanger 测序法）的优缺点及广泛应用

本研究就是基于第一代测序即 Sanger 测序结果的杂合突变识别的算法。

第一代测序的优势：

（1）第一代测序技术的准确性远高于二、三代测序，因此被称为测序行业的“金标准”；

（2）第一代测序每个反应可以得到 700-1000bp 的序列，序列长度高于二代测序；

（3)第一代测序价格低廉，设备运行时间短，适用于低通量的快速研究项目。

第一代测序的劣势：

（1）第一代测序技术一个反应只能得到一条序列，因此测序通量很低。

虽然第一代测序有测序通量低的缺点，但对于当今基于基因测序的很多研究，第一代测序已经足够满足并得到广泛运用：对目的基因的 PCR 产物进行测序，得到目的基因序列；突变、SNPs、插入或缺失克隆产物的验证；微生物和真菌分类学鉴定、HLA 分型、病毒分型等；肿瘤突变基因的检测和肿瘤个体化治疗，致病基因位点明确并且数量有限的单基因遗传病检测；对新一代测序技术的结果进行验证等等。

4. Sanger 测序法的工作流程

Sanger 测序法目前主要用于分析 SNP。在当代的自动化测序仪工作流程下，Sanger 测序被分解为如下的三个步骤：

①**扩增**。通过聚合酶链式反应扩增操作可将染色体（二倍体中可以是两个同源染色体）上指定起始位点至终止位点间的待测序 DNA 片段作为模板，通过引物将其复制为多份用于读取；

②**读取**是利用四种碱基（ATCG）在各自的毛细电泳道上执行电泳和显影后会形成不同波长的荧光的物理特性，使用传感器读取荧光信号，从而获取到待测序 DNA 片段的四条（相对位点、对应碱基的荧光信号强度）向量序列。为了消除读取阶段，反应和传感器的误差，读取的过程常常会重复数千次，其对应碱基荧光信

号强度按照相对位点进行累加，从而最终输出四条（相对位点、对应碱基的荧光信号强度累加和）向量序列。

③**分析**。通过对比每个位点上四种碱基的荧光信号强度，通常情况下每个位点仅有一种波长荧光信号强度值较高，说明该位点上是该荧光波长对应的碱基。如果在同一位点强度值最大的有两种波长的荧光信号的强度值相当，则说明二倍体中（可以是同源染色体）的等位基因在该位点上的碱基不同，发生了杂合突变。

二、本研究的创新性与价值

目前运用 Sanger 测序数据去识别突变分为两种：杂合突变和纯合突变。识别纯合突变的技术已相当成熟，运用的是 BLAST 算法进行参考序列搜索，搜索到了进行比对即可找到纯合突变。但因为其原理是先直接把图形转换成碱基，所以只能识别出纯合突变，而对于杂合突变这种双峰就无法正确识别出来。因此对于杂合突变的识别还未有成熟的技术支持，只能单纯依靠人工肉眼看 Sanger 测序的结果图去挑拣出杂合突变所在位置，存在效率低下的问题。所以我们实现的一种 Sanger 测序数据杂合突变自动化识别算法具有绝对创新性，通过基于计算几何方法的求出包络面积能够高效的进行识别，准确率高且稳定可靠。该算法可以规避人工识别虽然比较准确，但由于大量重复性工作容易造成疲劳从而出错的问题，也有效地节约了人力成本，为研究人员争取到了宝贵的时间和精力。

不仅如此，我们创新的杂合突变的识别算法在药品研发，疾病研究，医疗服务等领域有广阔的应用前景，对推动生物基因工程的发展具有重要意义。生物与信息的结合必将是未来科技发展的热门更是趋势，我们的这次尝试，不仅为基因工程的发展提供了一种可能性，更为未来的技术研究埋下种子。

三、ABIF 格式读取与包络曲线绘制

ABIF 格式是 Thermo Fisher 公司制造的 Sanger 测序设备输出的标准格式，文件后缀名为 ab1。其中包含了样品信息（试管 ID，样品盘 ID），测序配置（仪器信号，染色剂、电泳电压、激光功率），原始测序数据（ATCG 测序原始数据 Y 坐标、X 坐标），测序结果（DNA 碱基字符串、Phred 质量分）等数百项信息[1]。

本文参考 ABIF 说明书提供的格式定义实现了 ABIF 数据加载[2]。使用二进制方式打开 abif 格式文件，验证开头 4 字节固定 ABIF 串，2 字节版本号一致后，通过一种目录索引方式描述文件中的全部数据。每个目录索引包含了数据名称、编号、数据存储格式、数据条数、元素个数、数据大小、偏移量，和句柄（废弃不用）。第一个目录索引包含的元素就是文件中所包含的所有数据的目录条目（格式也是目录索引），再遍历这些条目，就可以提取出 ABIF 格式全部数据了。ABIF 支持的数据存储格式有 byte, char, word, short, long, float, double, date, time, pString, cStrin, user。

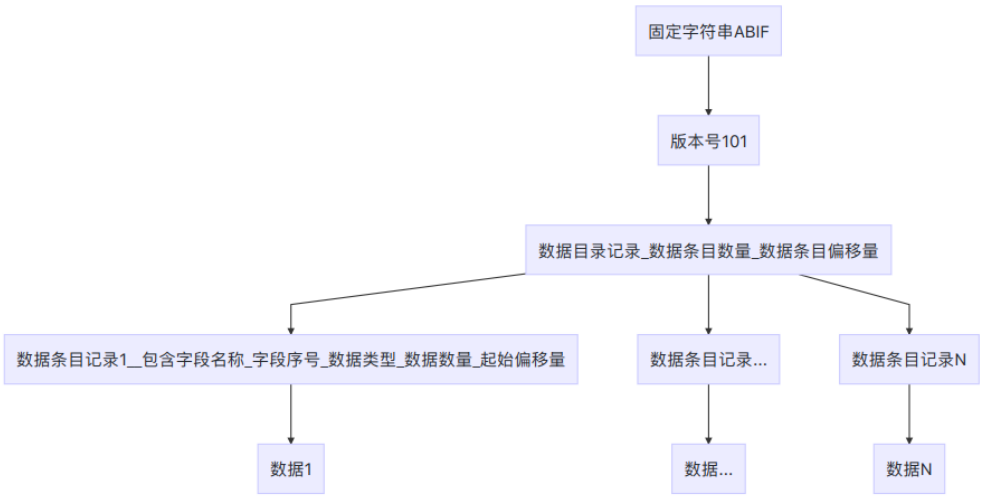

图 2 ABIF 格式读取

在本项目中主要读取的内容是数据名称为 *DATA*，编号 *9*、*10*、*11*、*12* 的 *GATC* 原始 Y 轴数据，格式为 *double*，以及数据名称为 *PLOC*，编号为 2 的 *GATC* 原始 X 轴（位点）数据，格式为 *double*，存储每个整数位点在原始 Y 轴的哪个下标，例如（2，63，72...），代表 Y 轴的第 2 条数据为原始位点 1，第 63 条数据对应原始位点 2，第 72 条数据对应原始位点 3 位置。通过扫描计算出 X 轴的填充密度，计算出每个原始 Y 轴数据的具体位点位置，形成包络曲线数据，用于后续流程。

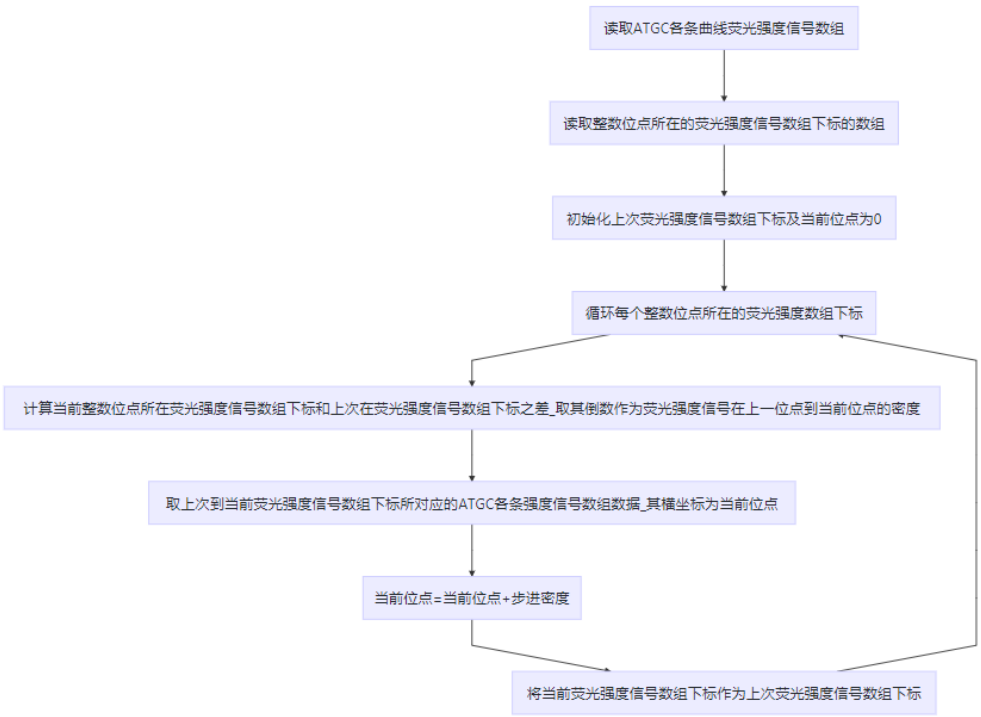

图 3 荧光强度信号数据处理

有了如上的包络数据，结合 HTML5 Canvas 技术[3]，根据观察范围进行数据裁剪，定义样式，进行坐标变换，绘制路径，从而逐一完成坐标系、坐标轴标签、四条碱基对应曲线、图例、辅助线的绘制。系统使用 React 组件化开发，由基于拖拽交互的缩放，移动的控制器，支持多包络曲线对比的渲染器和可控图例构成，由此完成包络曲线绘制工作。

在数据裁剪中的主要技巧是在裁剪边界处可以根据边界两侧的点线性推算边界上的数值，避免绘制的时候边界出现明显的折线现象。

ABIF 格式读取与包络曲线绘制的代码见附录。

四、使用包络线面积差法识别杂合突变

下图展示了杂合突变的两个例子，在下图指出的两个重合峰处，一对同源染色体上的一对等位基因中一个基因出现突变，造成测量出的荧光信号强度值相当。

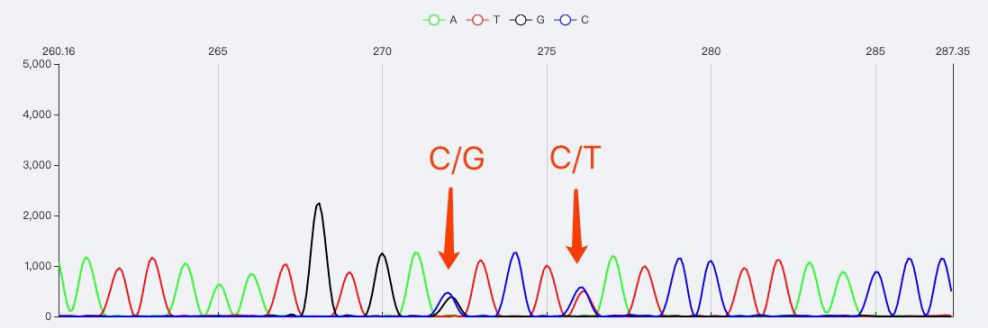

图 4 从 Sanger 测序结果中识别杂合突变

杂合突变的发现，主要的思路就是在碱基荧光信号包络线上，每个位点上是否出现重合峰。从几何上来看，在碱基荧光信号包络线上，杂合突变的主要特征是：两个碱基的荧光信号包络线在同一特定位点处附近形成的峰面积相当。由此，本文提出一种使用包络线面积识别杂合突变的算法，这个算法的基本思路如下：

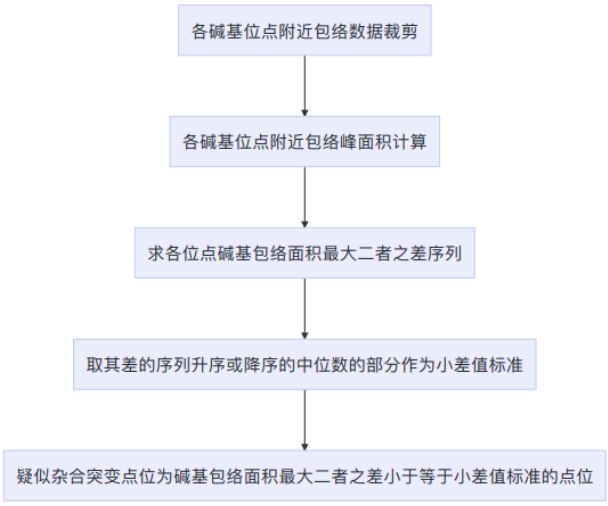

图 5 包络面积识别杂合突变算法思路

各碱基位点附近包络数据裁剪是指，按每种碱基的每个位点左右各 0.5bp 的位置截取包络数据。这其中的主要困难是一定情况下对应截取点无数据，造成后续面积计算变小，此时应使用斜率推算截取点的数据值，避免此问题。

各碱基位点附近包络峰面积计算是指，将包络两组坐标之间的面积视作为梯形，利用梯形面积公式进行计算。

求各位点碱基包络面积最大二者之差是指，每个位点上的碱基包络面积求出后，进行求差操作，大部分非杂合突变的面积差应当显著大于杂合突变的两个峰的面积差。

取其差的集合的中位数的部分作为小差值标准，疑似杂合突变位点为碱基包络面积最大二者之差小于等于小差值标准的位点。这是指从图形上看杂合突变的发生是小概率事件，因此其中位数应当表征了非杂合突变的场景，我们取其数值的 20%作为小差值标准，通过此标准过滤各位点碱基包络面积差序列，可以显著的过滤出杂合突变的数据。

```
// 计算离散点与 x 轴所包围区域的面积
```

```
function calcAreaByPoints(points) {
  let total = 0.0;
  for(let i = 1; i < points.length; i++) {
    const lastPoint = points[i - 1];
    const point = points[i];
    // (上底 + 下底) * 高 / 2.0
    const area=( lastPoint[1]+point[1] )*( point[0] - lastPoint[0] ) /
2.0;
    total += area;
  }
  return total;
}
// 按照碱基位置聚合计算左右各 0.5 区域的区块面积
function calcAreaByBlock(lines) {
  // 循环 ATGC 的每一类碱基
  return lines.map(points => {
    // 从 0bp 开始计算
    let currentBP = 0;
    let lastPoint = [0, 0];
    let currentBPPoints = [lastPoint];
    const areas = [];
    // 对于每个数据点
    for(const point of points) {
      const [x, y] = point;
      // 小数位偏差每到一次 0.5，计算当前 bp 内的图形面积，否则将点加入
待计算点
      const offset = x - currentBP;
      if(offset < 0.5) {
        if(x === lastPoint[0]) {
          lastPoint[1] = point[1];
        }
        else {
          currentBPPoints.push(point);
          lastPoint = point;
        }
        continue;
      }
      else {
        // 如果恰好是 0.5，这个点可以直接用于计算面积
        if(offset === 0.5) {
          currentBPPoints.push(point);
        }
        else {
          // 如果不是恰好 0.5，那么可以用斜率推导 0.5 位置坐标，然后用
```

它计算面积

```
          // 计算上一点到这一点斜率
          const k = (y - lastPoint[1]) / ( x - lastPoint[0] );
          // 通过斜率计算 0.5 位置的点坐标
          const fakeX = currentBP + 0.5;
          const fakeY = lastPoint[1] + (fakeX - lastPoint[0]) * k;
          const fakeHalf = [fakeX , fakeY];
          currentBPPoints.push(fakeHalf);
        }
        const area = calcAreaByPoints(currentBPPoints);
        areas.push([currentBP, area]);
        // 要计算的 bp 向后挪一位
        currentBP += 1;
        // 从当前点作为下次计算的点
        currentBPPoints = [point];
        // 保留上次点，可能要用于计算斜率
        lastPoint = point;
      }
    }
    // 计算最后一个点面积
    const area = calcAreaByPoints(currentBPPoints);
    areas.push([currentBP, area]);
    return areas;
  });
}
// 杂合突变概率估计
function calcHeterozygousMutationRate(lineAreas) {
  const dimRows = lineAreas.length;
  const dimCols = lineAreas[0].length;
  const ratePostions = new Array(dimCols);
  const rates = new Array(dimCols);
  for(let col = 0; col < dimCols; col++) {
    const group = new Array(dimRows);
    for(let row = 0; row < dimRows; row++) {
      group[row] = lineAreas[row][col][1];
    }
    // 组内从大到小排序
    group.sort((a, b) => b - a);
    // 在每一组 4 个面积相比时，有如下几种情况：
    // 1、该组内只有一个较大的数，概率小
    // 2、改组内有两个或多个较大的数，概率大
    const rate = group[0] - group[1];
    ratePostions[col] = [col, rate];
    rates[col] = rate;
```

```
  }
  // 将所有的可能值降序排序
  const minRates = rates.sort((a, b) => a - b);
  // 取中位数
  const midRate = minRates[parseInt(minRates.length / 2, 10)];
  // 筛子 比中位数小20%的就是要找出来的可能顺序
  const filterRate = midRate * 0.2;
  // 可能位置
  let possiablePositions = ratePostions.filter(ratePostion =>
ratePostion[1] <= filterRate);
  return possiablePositions;
}
```

五、使用 Phred 分数去除噪声数据

通过波形分析测序结果的质量决定了所识别碱基的正确性。因为 DNA 测序序列作为模板，通过引物进行了 PCR 扩增，会造成大部分测序结果首尾部分出现噪声区。这些噪声区的特点是波形峰值较高，每个位点上碱基荧光信号强度包络线混淆不清晰。因为知道这种问题的存在，所以测序样本往往会在外显子之外略微包含两端的内含子，从而保证外显子的完整性，能够把最终对于蛋白质的影响分析出来。

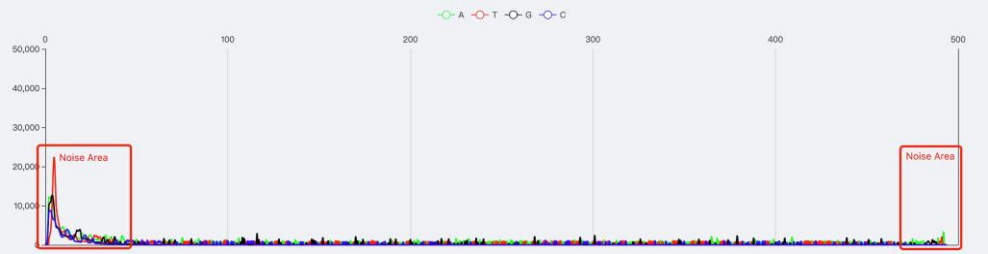

图 6 噪声区和有效测序结果-整体

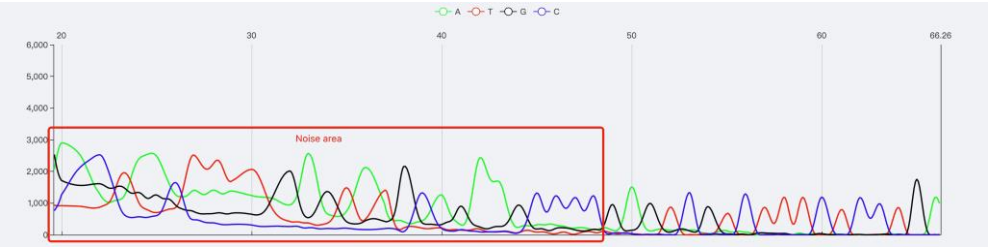

图 7 噪声区和有效测序结果-局部

生物信息学领域使用 phred 分数对于测序质量进行评价，每个碱基的 phred 分数越高说明测序结果越准确。在 ABIF 格式中已经包含了碱基 phred 分数，故参考网上的 Modified Mott trimming algorithm 实现了噪音概率估计，用于求最大 phred 分子序列，这个子序列对应了有效测序结果部分数据。算法核心思路为设 F(n)是最大子序列，i 表示起点下标，n 表示终点下标。那么，要么是前 n-1 项和加 n 项和最大，要么是第 n 项最大，即有：F(n) = Max(F(n-1)+F(n), F(n))。从动态规划角度来思考问题，如果是 F(n-1)+F(n)较大，则有：F(n-1)+F(n) > F(n)，等价于：F(n-1) > 0。

```
// 噪音概率估计 基于 Modified Mott trimming algorithm
function calcNoisyParts(phredScores) {
  if(!phredScores || !phredScores.length) {
    throw new Error('没有有效的 phred 评分！');
  }
  // const { indexMap, dirs } = file;
  // 计算中位数的一半
  const ascSortedPhredScores = phredScores.concat().sort((a, b) => a -
b);
  const halfMidPhredScore = ascSortedPhredScores[Math.round
(ascSortedPhredScores.length / 2)] / 2.0;
  // 减去中位数的一半
  const reducedPhredScores = phredScores.concat().map(v => v -
halfMidPhredScore);
```

```
  // 求最大和子序列
  let sum = reducedPhredScores[0];
  let max = sum;
  let begin = 0;
  let end = reducedPhredScores.length;
  // 求最大和子序列
  for(let i = 1; i < reducedPhredScores.length; i++) {
    // F(n-1) > 0
    if(sum > 0) {
      sum += reducedPhredScores[i];
    }
    else {
      sum = reducedPhredScores[i];
      begin = i;
    }
    if(sum > max) {
      max = sum;
      end = i;
    }
  }
  return {
    leftNoisePosition: begin,
    rightNoisePosition: end
  };
}
```

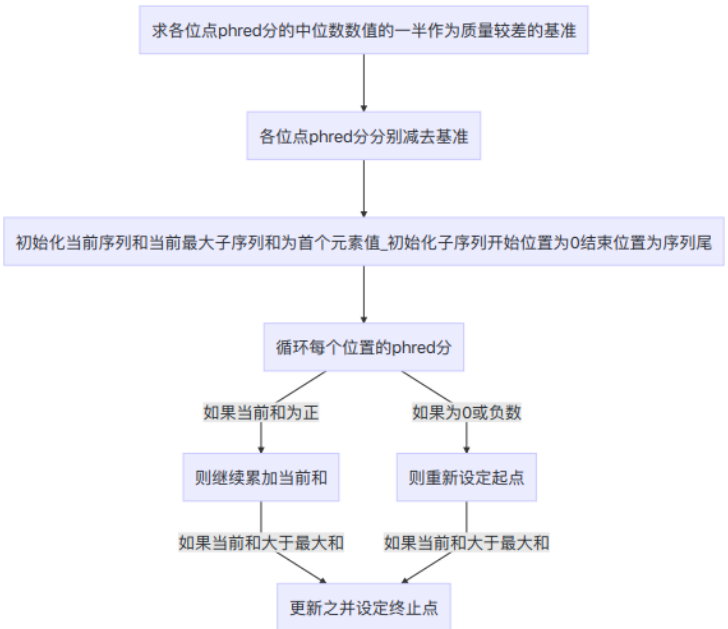

图 8 噪音概率估计算法思路

六、实验

1.建立基于本算法的系统

根据上述算法，我们使用 ES6 语言实现了采用该算法的系统，可在浏览器环境中运行。系统在浏览器环境下无需安装、使用方便、系统界面友好、运行稳定。

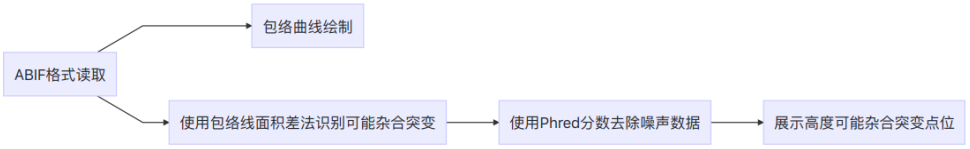

图 9 杂合突变识别算法过程

2.实验过程和记录

我们从东部战区总医院普通外科重症胰腺炎实验室取得了 200 组不同 ABIF 格式的测序数据，对其分别使用人工识别和我们的 SNP 基因分析系统识别，并记录了每次的文件名、人工识别杂合突变位点、系统识别杂合突变位点、人工识别时间、系统识别时间。以下为其中的 25 组记录。

序号	文件名	人工识别杂合突变位点	系统识别杂合突变位点	人工识别时间(s)	系统识别时间(s)	系统识别正确率	倍速
1	AC2-1.ab1	无	无	19.2	1.1	100%	2.0
2	AC2-2.ab1	无	无	22.9	1.4	100%	3.0
3	AC2-3.ab1	无	无	21.2	1.5	100%	2.4
4	AC2-4.ab1	无	无	20.7	1.3	100%	2.6
5	AC2-5.ab1	无	无	20.3	1.6	100%	2.2
6	AC2-6.ab2	无	无	20.4	1.5	100%	2.2
7	AC2-7.ab3	无	无	21.4	1.3	100%	2.7
8	AC2-8.ab4	无	无	20.2	1.4	100%	2.1
9	AC2-9.ab5	无	无	22.2	1.2	100%	3.2
10	AC2-10.ab6	无	无	23.2	1.5	100%	2.8
11	LPL5-161.ab1	276,242,342	276,242,342	44.2	5.8	100%	1.3
12	LPL5-	无	无	46.3	6.1	100%	1.3

	162.ab2						
13	LPL5-163.ab3	无	无	40.5	6.0	100%	1.1
14	LPL5-164.ab4	无	无	46.3	6.3	100%	1.2
15	LPL5-165.ab5	无	无	48.9	5.8	100%	1.4
16	LPL5-166.ab6	无	无	41.7	5.8	100%	1.1
17	LPL5-167.ab7	无	无	43.0	6.0	100%	1.2
18	LPL5-168.ab8	85	85	47.2	5.8	100%	1.4
19	LPL5-169.ab9	无	无	40.3	5.8	100%	1.1
20	LPL5-170.ab10	无	无	45.4	5.4	100%	1.4
21	LPL8-72.ab1	无	无	24.9	2.1	100%	1.9
22	LPL8-198.ab1	74	74	24.8	1.5	100%	2.7
23	LPL8-225.ab1	无	73	23.8	1.9	100%	2.1
24	LPL8-234.ab1	无	无	22.2	2.2	100%	1.6
25	LPL1-204.ab1	无	无	43.5	8.2	100%	0.9

图 10 人工识别和 SNP 基因分析系统识别的部分实验记录

3.实验结果分析

对于这 200 组数据（其中 34 组数据存在杂合突变现象），人工识别时间的平均值为 39.39s，系统识别时间的平均值为 3.62s，系统识别的平均速度为人类的 10.88 倍。通过比对人工识别的杂合突变位点和系统识别的杂合突变位点，我们发现，人工识别杂合突变位点的准确率为 99.5%，也就是说，即使因为位点上出现重合峰的特征较为明显，人工识别可以达到较高的准确率，但这样的大量重复性工作容易造成疲劳，仍然可能出错。而系统识别杂合突变位点的准确率在这次测试中为 100%，即使不排除系统在其他数据上出错的可能性，也很好地验证了我们的 Sanger 测序数据杂合突变识别算法的可靠性。

综合来看，系统的识别效率远高于人工识别（系统效率因设备及其运算速度而异，但绝大多数情况下远高于人工识别），准确率高且稳定可靠，适用范围广。目前医院基因检测中存在的人工识别杂合突变的过程可以被我们的 Sanger 测序数据杂合突变识别算法所取代，所以我们的算法具有极高的应用价值。

七、总结

我们实现了一种二倍体同源染色体 Sanger 测序中自动识别杂合突变的方法。该方法首先通过计算几何算法，将每个 DNA 位点的四种碱基两侧各 0.5bp 的包络线面积进行计算，得到各位点四种碱基包络面积。结合图形特点，将包络面积中位数的 20%作为固定空包络面积。其次，在每个位点上将四种碱基包络面积按大小降序排列，用最大的面积减去第二大的面积求得面积差。若面积差接近空包络面积，则说明此处存在两个不同碱基信号强度高度一致，有可能是杂合突变。若此处面积差远大于空包络面积，则说明这个位点仅有一个碱基。最后，因 Sanger 测序方法技术限制，在头尾处可能存在噪音信号。为了避免噪音信号对于杂合突变识别的影响，本研究使用了 Phred 碱基质量分，结合动态规划算法，实现了最大有质量序列的识别。实验证明，我们的算法具有高效、高准确率的特点。

八、附录

1.ABIF 格式读取与包络曲线绘制的代码

（1）读取 ANIF 格式目录条目

```
function readABIFDirEntry(reader) {
    const tagName = reader.readString(4);
    const tagNumber = reader.readInt();
    const elementType = reader.readShort();
    const elementSize = reader.readShort();
    const numElements = reader.readInt();
    const dataSize = reader.readInt();
    const dataOffset = reader.readInt();
    const dataHandle = reader.readInt();
    assert(elementSize * numElements <= dataSize, '目录中的元素尺寸数量超出了数据大小，请检查文件完整性');
    return {
        tagName,
        tagNumber,
        elementType,
        elementSize,
        numElements,
        dataSize,
        dataOffset,
        dataHandle
    };
}
// 读取 ABIF 全部目录
function readABIFDirs(reader, dirEntry) {
    const dirCount = dirEntry.numElements;
    const dirs = new Array(dirCount);
    assert(dirEntry.tagName === 'tdir', '文件版本不符合要求，请联系供应商。');
    // 偏移到开始位置
    reader.seek(dirEntry.dataOffset);
    // 读取全部目录结构
    for(let i = 0; i < dirCount; i++) {
        const dir = readABIFDirEntry(reader);
        dirs[i] = dir;
    }
    const indexMap = {};
    // 加载对应目录的数据
    for(let j = 0; j < dirs.length ; j++) {
        const dir = dirs[j];
        // 根据描述获取数据类型
        const dataType = elementTypeDefineMap[dir.elementType > 1024 ?
```

```
1024: dir.elementType];
        if(!dataType) {
            console.error(`目录${dir.tagName}的目录类型
${dir.elementType}无法识别，跳过！`);
            continue;
        }
        // 序号
        dir.index = j + 1;
        dir.key = `${dir.tagName}_${dir.tagNumber}`;
        // 数据类型
        dir.dataType = dataType;
        // 从文件中读取数据
        dir.data = dataType.read(reader, dir);
        // 增加字段注解，不是所有的字段都有注解
        dir.tag = findTagNameInfo(dir.tagName, dir.tagNumber);
        // 增加索引
        indexMap[dir.key] = j;
        // 显示没有识别的字段
        // if(!tagNameInfo) {
        //     console.log(`${dir.tagName}_${dir.tagNumber}`,
dataType.name, dir.data, tagNameInfo ? tagNameInfo.tagDesc: null);
        // }
    }
    return { dirs, indexMap };
}
// 读取 ABIF 格式数据
export function readABIFFormat(buffer) {
    const reader = new BufferReader(buffer);
    // 文件头 ABIF 四个字
    const fileHeader = reader.readString(4);
    assert(fileHeader === 'ABIF', 'AB1 格式文件头不符合要求，请检查您
是选择了正确的文件类型。');
    // 版本号
    const fileVersion = reader.readShort();
    assert(fileVersion === 101, 'AB1 格式文件版本不符合要求，目前仅支
持 101 版本，请联系供应商。');
    // 目录入口
    const dirEntry = readABIFDirEntry(reader);
    // 读取所有目录
    return readABIFDirs(reader, dirEntry);
}
```

（2）伪代码描述图形数据整理步骤

```
// 获取 Y 轴数据
const aFull = abif['DATA_10'];
const tFull = abif['DATA_11'];
const gFull = abif['DATA_9'];
const cFull = abif['DATA_12'];
// 对应整数位点的荧光强度数据下标，例如 2, 63, 72 ...
const pLoc2 = abif["PLOC_2"];
// 计算荧光强度数据的精确位点坐标
let lastRawIndex = 0;
let position = 0.0;
pLoc2.forEach((rawIndex, currentPos) => {
  // 计算位点下标密度 = 一个位点 / 上次下标 - 本次下标
  const density = 1.0 / (rawIndex - lastRawIndex);
  // 循环下标
  for(let index = lastRawIndex; index < rawIndex; index++){
    // 得到精确位点和对应数据
    console.log(`x=${position} a=${aFull[index]} t=${tFull[index]}
g=${gFull[index]} c=${cFull[index]}`);
    position += density;
  }
  // 去除累加误差
  position = currentPos + 1.0;
  // 更新上次下标，用于下次计算使用
  lastRawIndex = rawIndex;
});
```

（3）**根据数据范围进行裁剪，裁剪的数据两边补齐端点，避免断线现象**

```
function tailSequences(sequences, leftPercent, rightPercent) {
    const tailedSequences = [];
    // 以最大碱基序列长度作为序列长度
    const sequenceBaseLength = Math.max.apply(null,
sequences.map((sequence) => {
        if(sequence.length === 0) {
            return 0;
        }
        return sequence[sequence.length - 1][0];
    }));
    // 计算碱基片段开始，结束
    const sequencePartBaseBegin = Math.floor(sequenceBaseLength *
leftPercent / 100.0);
    const sequencePartBaseEnd = Math.ceil(sequenceBaseLength *
rightPercent / 100.0);
    let maxYValue = 0;
    // 根据下标位置，截取每根曲线，并且如果不能精确截取，就需要补齐开
```

```
始结束点
    for(const sequence of sequences) {
        // 存放裁剪过得序列
        const tailedSequence = [];
        // 序列长度不能为零
        if(sequence.length === 0) {
            tailedSequences.push(tailedSequence);
            continue;
        }
        // 确定碱基片段开始数组下标位置
        const sequenceLength = sequence.length;
        let sequencePartIndexBegin = 0;
        for(let index = 0; index < sequenceLength; index++) {
            const baseData = sequence[index];
            const sequenceBasePosition = baseData[0];
            //如果碱基数据位置与序列裁剪的起始位置完全匹配，直接加入
            if(sequenceBasePosition === sequencePartBaseBegin) {
                // 求 y 最大值
                if(baseData[1] > maxYValue) {
                    maxYValue = baseData[1];
                }
                tailedSequence.push(baseData);
                sequencePartIndexBegin = index + 1;
                break;
            }
            else if(sequenceBasePosition > sequencePartBaseBegin) {
                // 求上一碱基数据和当前碱基数据在裁剪位置的强度值
                // x 为剪切位置碱基坐标
                const x = sequencePartBaseBegin;
                // 取上一点坐标，若为起始，用 0,0 作为上一点坐标
                const lastBaseData= (index === 0)[0,0]: sequence[index
- 1];
                // 取出坐标 x,y 值
                const [ x1, y1 ] = lastBaseData;
                const [ x2, y2 ] = baseData;
                // 使用投影法，计算 x 位置 y 强度
                const y = y1 + (y2 - y1) / (x2 - x1) * (x - x1);
                const tailedBaseData = [x, y];
                // 求 y 最大值
                if(tailedBaseData[1] > maxYValue) {
                    maxYValue = tailedBaseData[1];
                }
                // 加入裁剪起始处数据
                tailedSequence.push(tailedBaseData);
```

```
                sequencePartIndexBegin = index + 1;
                break;
            }
        }
        // 确定碱基片段结束数组下标位置
        for(let index = sequencePartIndexBegin; index <
sequenceLength; index++) {
            const baseData = sequence[index];
            const sequenceBasePosition = baseData[0];
            // 求上一碱基数据和当前碱基数据在裁剪位置的强度值
            if(sequenceBasePosition > sequencePartBaseEnd) {
                // x 为剪切位置碱基坐标
                const x = sequencePartBaseEnd;
                // 取上一点坐标，若为起始，用 0,0 作为上一点坐标
                const lastBaseData=(index=== 0)?[0, 0]: sequence[index
- 1];
                // 取出坐标 x,y 值
                const [ x1, y1 ] = lastBaseData;
                const [ x2, y2 ] = baseData;
                // 使用投影法，计算 x 位置 y 强度
                const y = y1 + (y2 - y1) / (x2 - x1) * (x - x1);
                const tailedBaseData = [x, y];
                // 求 y 最大值
                if(tailedBaseData[1] > maxYValue) {
                    maxYValue = tailedBaseData[1];
                }
                // 加入裁剪起始处数据
                tailedSequence.push(tailedBaseData);
                break;
            }
            // 求 y 最大值
            if(baseData[1] > maxYValue) {
                maxYValue = baseData[1];
            }
            // 加入裁剪起始处数据
            tailedSequence.push(baseData);
            // 如果完全匹配插入后退出
            if(sequenceBasePosition === sequencePartBaseEnd) {
                break;
            }
        }
        tailedSequences.push(tailedSequence);
    }
    return {
```

```
        sequenceBaseLength, // 最大序列总长度
        sequencePartBaseBegin, // 根据百分比计算出的裁剪后序列起始位置
        sequencePartBaseEnd, // 根据百分比计算出的裁剪后序列终止位置
        sequencePartBaseLength: sequencePartBaseEnd -
sequencePartBaseBegin,
        // 根据百分比计算出的裁剪后序列长度
        tailedSequences, // 裁剪过的序列
        maxYValue: maxYValue * 1.25, // 裁剪区域内的 Y 最大值
    };
}
//绘制线条
function drawLine(
    sequences,
    id,
    color,
    text,
    ctx,
    axisHeight,
    sequencePartBaseLength,
    controlWidth,
    axisFontSize,
    axisMarginTop,
    graphHeight,
    devicePixelRatio,
    paddingLeft,  // 左空
    paddingRight,  // 右空
    maxYValue,
    sequencePartBaseBegin
) {
    // 保存图形上下文
    ctx.save();
    // 缩放比调整，考虑 retina 屏幕的设备绘制清晰度问题
    ctx.scale(devicePixelRatio, devicePixelRatio);
    // 设置设置线条粗细为 1
    const xRefLineWidth = 1;
    // 取到刻度线的总高为 xAxisYEnd
    const xAxisYBegin = axisFontSize + axisMarginTop;
    const xAxisYEnd = xAxisYBegin + axisHeight;
    const xRefYBegin = xAxisYEnd;
    // 计算绘制区域的宽高
    const boundLineWidth = 2;
    // 可以得到 y 轴的最大值为 maxY
    const maxX = controlWidth - paddingRight - 2 * boundLineWidth;
    const maxY = xRefYBegin + graphHeight;
```

```
    // 开始绘制
    ctx.beginPath();
    ctx.moveTo(paddingLeft, maxY);
    ctx.strokeStyle = color;
    ctx.lineWidth = xRefLineWidth;
    // 建立循环的数组
    const sequence = sequences[id];
    //循环遍历数组绘制图片
    if(sequence && sequence.length !== 0)
    {
        for (const point of sequence) {
            const [ x, y ] = point;
            // 定义每次循环的参数值
            let xAxis =  paddingLeft + (x - sequencePartBaseBegin) *
step;
            if(xAxis >= maxX) {
                break;
            }
            // 求出数据折线到 x 轴的距离
            let yAxis = xRefYBegin + graphHeight * (1 - y / maxYValue)
            ctx.lineTo(xAxis,yAxis)
        }
    }
    // 绘制边线
    ctx.stroke();
    // 恢复图形上下文到上次 save()的情景
    ctx.restore();
}
```

参考文献

[1] 邓继忠，林伟森，甘四明. 一种二倍体片段测序中 SNP 检测系统的构建 华南农业大学学报，2016，37（3）：115-120.
[2] 唐立琼，肖层林，王伟平. SNP 分子标记的研究及其应用进展[J]. 中国农学通报，2012，28(12)：154-158.
[3] 许家磊，王宇，后猛，等. SNP 检测方法的研究进展[J]. 分子植物育种，2015，13(2)：475-482.
[4] MATTHEW S，JAMES S，ROBERTSON P D. 自动化基于序列的检测和二倍体样品中单核苷酸多态性（SNP）的基因分型[J]. Nat Genet，2006,38（3）：375-381.
[5] NGAMPHIW C，KULAWONGANUNCHAI S，ASSAWAMAKIN A. VarDetect：核苷酸序列变异探索工具[J]. BMC Bioinformatics，2008,9（S12）：9.

致谢

我们非常感谢我校史钋镭老师对我们研究的无偿帮助和指导。我们感谢东部战区总医院的杨琦博士，因为这个项目的灵感来自于与她的一次谈话，在谈话中她向我们抱怨她单调的“肉眼识别杂合突变”工作。我们提出要求后，杨琦博士很乐意地让我们体验了她的工作。我们发现这项工作不仅很累，而且很耗时，尤其是对于每天要检查成千上万张图片来识别杂合突变的医生来说，是一项艰巨的任务。由于该过程目前已广泛应用于与基因相关研究，特别是在药物生产中，因此识别杂合突变的效率低下的问题是普遍存在的。在她的慷慨帮助下，我们对识别杂合突变的过程和目前的技术有了更多的背景知识，于是在不懈努力下我们找到了帮助解决这个问题的方法。我们还要尤其感谢杨琦博士在东部战区总医院的授权下提供了 ABIF 格式荧光强度信号的真实数据，这是我们研究的基础。我们也感谢南京友谱信息科技有限公司的张未波先生和他的研究团队在研究、组织讨论和论文撰写方面给予的帮助。

王禹听负责生物背景的研究，以及 ABIF 格式荧光强度信号的读取和预处理数据。李滕昊设计了从 Sanger 测序结果中识别杂合突变的算法和程序，并使用 Phred 评分去除了噪声数据。金川杨基于之前的方法设计了一个在线识别杂合突变的网站，并进行了实验的主要部分，包括将我们的方法应用于荧光强度信号的检测数据，以及将我们的方法与目前使用的方法进行比较。金川杨撰写了这篇论文，所有作者共同参与了论文的讨论和审查。史钋镭老师和张未波先生对论文进行了修改。

学术诚信申明

本参赛团队声明所提交的论文是在指导老师指导下进行的研究工作和取得的研究成果。尽本团队所知,除了文中特别加以标注和致谢中所罗列的内容以外论文中不包含其他人已经发表或撰写过的研究成果。若有不实之处，本人愿意承担一切相关责任。

参赛队员：金川杨，王禹听，李滕昊　指导老师：史钋镭

2019 年 9 月 1 4 日