DNA序列复杂比对

季雨昊 23300240010

功能简介

本算法实现了复杂DNA序列比对算法。首先对query的正向和反向序列用多尺度k-mer哈希提取不同长度的锚点，通过动态规划在所有锚点中寻找无重叠的主链区间，并自动合并、延伸主链区间（允许模糊匹配），最终输出唯一、无重叠的高质量比对区间。

目录结构

```
项目目录结构
.
├── README.md
├── input.txt                # "实验二:1"的输入
├── input2.txt               # "实验二:2"的输入
├── output.txt               # "实验二:1"的输出（其中final_ans为答案）
├── output2.txt              # "实验二:2"的输出（其中final_ans为答案）
├── grade.py                 # 评分脚本，和助教的jupyter评分脚本一样
└── src/
    ├── __main__.py          # 算法执行入口
    ├── dna_matcher.py       # 算法实现
    └── visualize.py         # 结果可视化
```

用法

1. 算法主程序运行

- **直接运行（交互输入）**

  ```
  python -m src
  ```

  按提示输入query和reference序列。

- **指定输入文件**

  ```
  python -m src --input input.txt
  ```

  结果会在终端打印。

- **指定输出文件**

```
python -m src --input input.txt --output output.txt
```

建议采用此种方法。

- **常用参数说明**
  可通过命令行调整所有核心参数（如max_gap、alpha、gamma、bonus、slope_eps、min_k、k_ratio、ex_max_edit_ratio等），如：

  ```
  python -m src --input input.txt --output output.txt --max_gap 40 --
  ex_max_edit_ratio 0.12
  ```

  现在的参数已经较优，不建议调整。

2. 结果可视化

```
python src/visualize.py --input output.txt
```

在浏览器绘制可视化图。用的是plotly库，和lab1演示差不多，可以交互。

3. 匹配区间评分

```
python grade.py --input input.txt
```

在终端输入答案，格式与output里面final_ans一致，为中括号套起来的(query_start, query_end, ref_start, ref_end)若干元组。之后即可看到得分。

算法实现说明

1. 多尺度k-mer锚点构建

对query的正向和反向互补序列，采用多尺度k-mer哈希（minhash）在reference中查找完全匹配的锚点（anchor）。kmer长度根据等比数列递减。优先提取长锚点，最后提取短锚点。记录每个锚点在query、reference中的起始位置、长度和方向（正向/反向）。

2. 动态规划寻找主链（图算法）

将所有锚点按query/ref起点和方向排序。用动态规划在所有锚点中寻找query/ref区间均无重叠、得分最高的主链。DP得分考虑gap惩罚、斜率惩罚、延伸的顺滑奖励等。

3. 主链区间合并

对DP主链区间，将相邻、正反向相同的区间合并为更长区间。合并时要求gap、斜率等满足阈值，保证区间连续性和合理性。

4. 区间延伸

由于锚点构建时，kmer选取不够精细，此时找到的主链区间可能不是最长的匹配。因此，对每个主链区间，自动向两端延伸，允许少量错配，用参数`ex_max_edit_ratio`调控错配数量。

5. 最终合并

所有延伸后的区间再次按原有逻辑合并，得到最终答案。

算法伪代码

```
Algorithm DNA-Sequence-Alignment(query, ref)
    anchors ← build_anchors(query, ref)
    chain ← dp(anchors)
    merged_chain ← merge_intervals(chain)
    extended_chain ← extend_intervals(merged_chain, query, ref)
    final_chain ← merge_intervals(extended_chain)
    Output final_chain

Procedure build_anchors(query, ref)
    anchors ← ∅
    for each k in k_list (from large to small)
        for each k-mer in query and reverse_complement(query)
            if k-mer matches in ref then
                anchors ← anchors ∪ (q_pos, r_pos, k, strand)
    return anchors

Procedure dp(anchors)
    Sort anchors by (q_start, r_start, strand)
    for j ← 1 to |anchors|
        for i ← 1 to j-1
            if anchors[i] and anchors[j] are non-overlapping and consistent
                Update dp[j] if better chain found
    return Reconstruct-Chain(dp)

Procedure merge_intervals(chain)
    result ← []
    for interval in chain
        if can_merge(result[-1], interval) then
            result[-1] ← merge(result[-1], interval)
        else
            result.append(interval)
    return result

Procedure extend_intervals(chain, query, ref)
    for each interval in chain
        while can_extend_left(interval, query, ref)
            interval ← extend_left(interval)
        while can_extend_right(interval, query, ref)
            interval ← extend_right(interval)
    return chain
```

运行结果

第一组输入

```
[(0, 6453, 0, 6453), (6453, 23015, 6813, 23376), (23015, 29829, 23016, 29830),
(29829, 29840, 17205, 17216)]
```

Score: 29.82k

第二组输入：

```
 [(0, 297, 0, 297), (297, 400, 397, 500), (400, 494, 505, 599), (494, 694, 594,
794), (694, 768, 694, 768), (768, 799, 618, 649), (799, 913, 699, 813), (913, 999,
700, 786), (999, 1200, 699, 900), (1200, 1299, 900, 999), (1299, 1402, 899, 1002),
(1402, 1500, 402, 500), (1500, 1600, 1000, 1100), (1600, 1694, 1300, 1394), (1694,
1807, 1194, 1307), (1807, 1892, 1107, 1192), (1892, 2011, 1392, 1511), (2011,
2020, 39, 48), (2274, 2499, 1474, 1699)]
```

Score: 2104

复杂度分析

1. 锚点构建

对query和reference分别滑窗提取k-mer，k值从大到小递减，我采取的是等比数列下降，K=O(lg(Lr)),整体复杂度为O(Lq * K + Lr * K) = O(Lq * lg(Lr) + Lr * lg(Lr))，其中Lq、Lr为query和ref长度，K为k_list长度。

2. DP主链提取

假设总共找到A个锚点，DP最长无重叠链的复杂度为O(A^2)。A远小于Lq、Lr。

3. 区间合并与延伸

合并操作为一次线性扫描，复杂度O(N)，N为主链区间数，N远小于A。区间延伸每次最多延伸到query/ref边界或遇到错配比例超限，最坏O(Lq)次比对，但实际每区间延伸长度有限，远小于原本区间长度。延伸后再合并一次，仍为O(N)。

4. 匹配区间输出与去重

区间去重、排序为O(N log N)。最终输出区间数远小于Lq。

5. 总体复杂度

- **时间复杂度**： $O(Lq * lg(Lr) + Lr * lg(Lr) + A^2 + N \log N)$，其中：
  - L_q、L_r 分别为 query 和 reference 的长度。
  - A 为锚点总数，N 为最终输出区间数。A 和 N 远远小于 L_q, L_r，且主链区间数 N 大大小于 A。这两点可以在我的两个output.txt看出来。

    - $O(L_q + L_r)$：k-mer哈希构建锚点。
    - $O(A^2)$：主链DP。
    - $O(N \log N)$：区间排序、去重。
- **空间复杂度**：$O(L_q + L_r + A + N)$，主要为哈希表、锚点、主链区间存储。

代码仓库

https://github.com/LagunaKi/Fudan-Algorithm2025-lab2.git