

# 项目一：MSA

人工智能 CS410 2021年秋季

姓名：李子龙 学号：518070910095 日期：2021 年 10 月 16 日

## 目录

<b>1</b>	<b>题目</b>	<b>2</b>
1.1	Topic . . . . .	2
1.2	Requirements . . . . .	2
1.3	Rules . . . . .	2
<b>2</b>	<b>动态规划算法</b>	<b>2</b>
2.1	双序列比对 . . . . .	2
2.2	多序列比对 . . . . .	3
2.3	运行时间 . . . . .	5
<b>3</b>	<b>A* 算法</b>	<b>5</b>
3.1	算法描述 . . . . .	5
3.2	运行时间 . . . . .	6

# 1 题目

## 1.1 Topic

Implement three algorithms to solve multiple sequence alignment (MSA) problems.

## 1.2 Requirements

- (1) Implement dynamic programming (DP) algorithm to find the optimal solution.
- (2) Implement A-star (A\*) algorithm to find the optimal solution.
- (3) Implement genetic algorithm to find the optimal/suboptimal solution.

## 1.3 Rules

表 1: Cost Matrix			
	Match $\alpha(p, p)$	Mismatch $\alpha(p, q)$	Gap $\delta$
Cost	0	3	2

The table above shows the pairwise cost matrix. For multiple sequence alignment, the cost should be calculated in a cycle pairwise manner. Note that GAP-GAP is a match and should be considered as 0 cost. For every query, find the best alignment(s) in the database with the lowest cost.

# 2 动态规划算法

## 2.1 双序列比对

在算法与复杂性课程<sup>[1]</sup>里, 已经提到了双序列比对的动态规划算法, 如图 1 所示, 双序列比对对于一个状态只需要考虑三个临近状态的转移, 分别是对齐 $\alpha$ , 间隔  $\delta_x$ 、 $\delta_y$ , 转换行动如表 2 所示。对于每一个状态, 都需要考虑经过哪一条路径消耗最小, 于是就有了如算法 1 的动态规划状态转移方程。

---

### Algorithm 1: 双序列比对动态规划 MSA

---

**Input:**  $x_1x_2 \cdots x_m, y_1y_2 \cdots y_n, \alpha, \delta$

**Output:** minimum cost

```

1 for  $i \leftarrow 0$  to  $m$  do  $M[i, 0] = i\delta$ ;
2 for  $j \leftarrow 0$  to  $n$  do  $M[0, j] = j\delta$ ;
3 for  $i \leftarrow 1$  to  $m$  do
4   for  $j \leftarrow 1$  to  $n$  do
5      $M[i, j] = \min(\alpha[x_i, y_j] + M[i-1, j-1], \delta + M[i-1, j], \delta + M[i, j-1]);$ 
6 return  $M[m, n]$ ;
```

---

表 2: 双序列行动坐标变换表

	$i$	$j$
$\alpha$	+1	+1
$\delta_x$	0	+1
$\delta_y$	+1	0

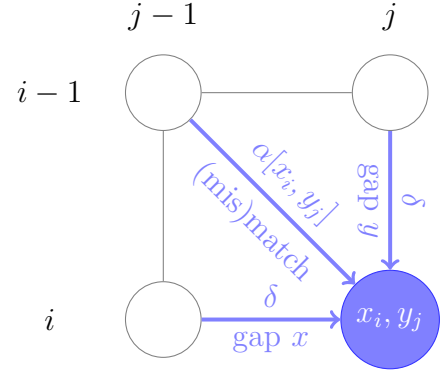


图 1: 动态规划双序列比对

## 2.2 多序列比对

对于三序列比对，情况就复杂地多，需要同时考虑七条路径。

表 3: 三序列行动坐标变换表

	$k$	$j$	$i$
$\alpha_x \delta_y \delta_z$	0	0	1
$\delta_x \alpha_y \delta_z$	0	1	0
$\delta_x \alpha_y \alpha_z$	0	1	1
$\delta_x \delta_y \alpha_z$	1	0	0
$\alpha_x \delta_y \alpha_z$	1	0	1
$\alpha_x \alpha_y \delta_z$	1	1	0
$\alpha_x \alpha_y \alpha_z$	1	1	1

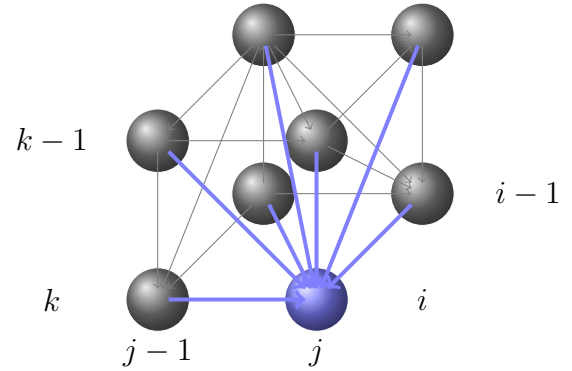


图 2: 动态规划三序列比对

可以统一化为多序列比对问题。对于  $L$  条序列比对，首先需要递归地初始化低维度边缘（如图 3 所示，注意附加高维度的间隙），之后余下空间其行动转换方法可以被表示为二进制从  $(\underbrace{0 \cdots 01}_L)_2$  到  $(\underbrace{1 \cdots 11}_L)_2$  内所有的数（最低位为第一维度），计算损耗使用上三角成对比较，

规则统一为

$$\text{compare} = \begin{cases} 0, & (-, -) \parallel (p, p) \\ 2, & (p, -) \parallel (-, q) \\ 3, & (p, q) \end{cases}$$

并在确定每一次行动后记录路径，最后回溯路径到原点。

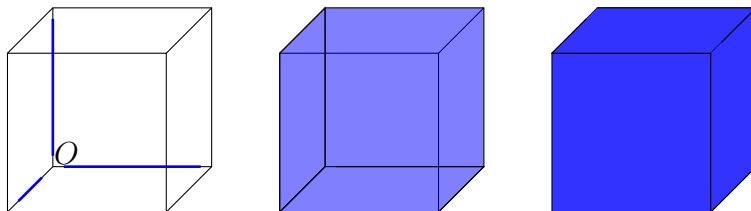


图 3: 降维递归

几乎类似于双序列比对，下面是 numpy 实现版本，虽然其速度没有使用 Python 内置的 list 版本 (`msa_mdp.py`) 的快，但是代码可读性已经与伪代码相当。

Listing 1: `msa_mdp.py`

```

29 def decodeMove(m:np.uint8,dim):
30     return tuple(1 if m & (2**v) > 0 else 0 for v in range(dim))
31
32 def editDistanceNDP(S,dist:np.array=np.array([]),move:np.array=np.array([])):
33     L = len(S)
34     if L == 0:
35         return np.array([0]), np.array([0])
36     global fdim
37     if len(dist)==0:
38         # initialize dist and move
39         shape = tuple(len(S[l])+1 for l in range(L))
40         dist = np.ones(shape=shape, dtype=np.int32)
41         dist = -1 * dist # negative means no data
42         move = np.zeros(shape=shape, dtype=np.uint8)
43         fdim = L
44     # calculate the lower dimension (edges)
45     for s in range(L):
46         slicer = tuple(0 if i==s else slice(None) for i in range(L)) # slice(
None) stands for : symbol
47         dist[slicer], move[slicer] = editDistanceNDP(S[0:s]+S[s+1:L], dist[
slicer], move[slicer]) # skip S[s]
48         # configure move, insert 0 in the corresponding bit
49         # Example: 4-dim xyzw xyw cube z(2) = 0, get an move 111(wyx), but with
that be zero, it should be 1011.
50         # REMEMBER to place the right end in the same level!
51         move[slicer] = (move[slicer] >> s << (s+1)) + (move[slicer] & (2**s-1))
52     # Spread the remaining space, since the edge case has been considered, the
remaining space will have the same action set.
53     it = np.nditer(dist, flags=['multi_index'], op_flags=["readwrite"])
54     while not it.finished:
55         pos = it.multi_index
56         if 0 in pos:
57             it.iternext()
58             continue # calculated
59         ## The range of available move is 1~(2^L-1)
60         minmove = np.uint8(0)
61         minvalue = np.inf
62         for m in range(1,2**L):
63             move_vec = decodeMove(m,L)
64             prev_pos = tuple(a-b for a,b in zip(pos,move_vec))
65             penalty = comparelist([S[a][p] if move_vec[a]==1 else "-" for a,p in
enumerate(prev_pos)]+["-" for i in range(fdim - L)]) # the term is
required since the higher dim will be gapped.
66             moved_dist = dist[prev_pos] + penalty
67             if moved_dist < minvalue:
68                 minmove = m
69                 minvalue = moved_dist
70             it[0] = minvalue
71             move[pos] = minmove
72             it.iternext()
73     return dist, move

```

## 2.3 运行时间

如果字符串平均长度为  $l$ ，该算法  $L$  维字符串的复杂度为：

$$O_S = \prod_{i=1}^L \text{len}(S[i]) = O(l^L)$$

对于该问题，有  $m$  个待比对序列， $n$  个数据库项目，总时间复杂度为：

$$mC_n^{L-1}O_S \approx mC_n^{L-1}l^L$$

实际运行时间如表 4，在服务器上运行时间如下。

表 4: 动态规划运行时间									
双序列					三序列				
	朴素实现	list实现	朴素实现	list实现		list实现	numpy实现	朴素实现	list实现
	<code>msa_dp.py</code>	<code>msa_mdp.py</code>	<code>msa_dp.py</code>	<code>msa_mdp.py</code>		<code>msa_mdp.py</code>	<code>msa_ndp.py</code>	<code>msa_dp.py</code>	<code>msa_mdp.py</code>
运行时间	29s	1min	24h	~36h					

## 3 A\* 算法

### 3.1 算法描述

A\* 算法会从后继结点中首先扩展评估函数  $f(n) = g(n) + h(n)$  最小的结点，如果  $h(n)$  的选择满足可满足启发式和一致性的性质，就可以找到按照贪婪算法的思想找到最优解。

这里将会非常乐观地估计剩下的字符串剩余部分都可以完美匹配，只会剩余间隔损耗。对于状态为  $n$  的启发函数就可以被定义为轮换剩余长度差的和之下界

$$\delta \sum_{cyc} |(l_1 - \text{pos}[i]) - (l_2 - \text{pos}[j])| \geq \delta \left( L \max a_i - \sum_i a_i \right) = h(n)$$

其中

$$a_i = l_i - \text{pos}[i]$$

不等式容易从下面图 4 的可视分析中论证，这样选择的  $h(n)$  满足可满足启发式。

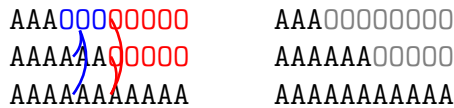


图 4: 每一个间隔都至少贡献了一次

之后来证明一致性。对于 A\* 算法而言，其下一步的定义如图 6 和 7 所示。此处每一步的损耗都会大于等于 0，而这种最好情况只会在全部序列都减少了 1 长度才会产生（超体对角线），这种情况下  $h(n) = h(n')$ ；由于坐标至少在某一维度上增加了 1，一旦产生了间隙，就会有至少 2 的损耗，但是启发函数只会对应地减少 1，所以这个函数将满足一致性：

$$h(n) \leq c(n, a, n') + h(n')$$

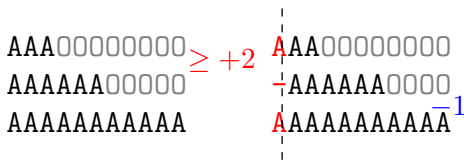


图 5: 前进一步的不等式贡献

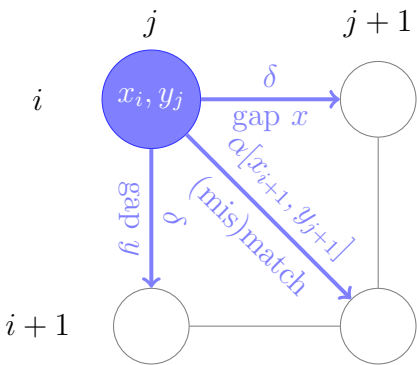


图 6: A\* 双序列比对

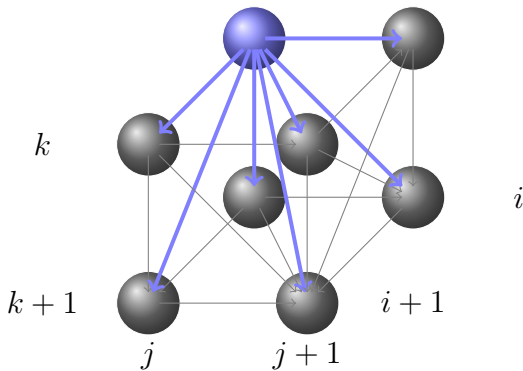


图 7: A\* 三序列比对

伪代码描述如算法 2 所示<sup>[2]</sup>，其中可选行动随着坐标的不同可能会被限制，这样就会首先扩展评估函数最小的结点。

### 3.2 运行时间

该算法的时间复杂度，对于  $L$  个字符串（平均长度为  $l$ ），最多进行  $\sum_i l_i = Ll$  步（如果平均长度为  $l$  的话），每一步的分支因子为  $2^L - 1$ ，单个实例需要花费时间

$$(2^L - 1) \sum_i l_i = L(2^L - 1)l = O(l)$$

因为这是一个树状结构的图，所以一定能够找到路径。当然如果有多个最小节点的情况，这个复杂度会变差，但最多不会超过  $O((2^L - 1)^{Ll})$ （实际上应当多项式时间内即可，这个是 A\* 的最差复杂度）。

表 5: A* 运行时间		
	双序列比对	三序列比对
运行时间	2min	~

### 参考文献

[1] XIAOFENG G. Algorithm & complexity class lab 06[EB/OL]. 2021. <https://github.com/LogCreative/AlgAndComplexity/blob/master/Lab06/Code-SequenceAlignment.cpp>.

[2] Wikipedia contributors. A\* search algorithm — Wikipedia, the free encyclopedia[EB/OL]. 2021. [https://en.wikipedia.org/w/index.php?title=A\\*\\_search\\_algorithm&oldid=1040995101](https://en.wikipedia.org/w/index.php?title=A*_search_algorithm&oldid=1040995101).

---

**Algorithm 2:** A\* 多序列比对

---

**Input:**  $L$ 个字符串列表 $S, \alpha, \delta$ 
**Output:** minimum cost

```

1   $dist[\cdot] \leftarrow \infty$ ;
2   $move[\cdot] \leftarrow 0$ ;
3   $dist[start] \leftarrow 0$ ;
4   $move[start] \leftarrow 0$ ;
5   $openSet \leftarrow \text{MIN-HEAP}()$ ;
6   $openSet[start] = h(start)$ ;
7   $closeSet \leftarrow \{\}$ ;
8  repeat
9       $current \leftarrow openSet.pop()$ ;
10     if  $current = finish$  then
11         return  $dist[current]$ ;
12      $closeSet.add(current)$ ;
13     foreach available move of current do
14          $n \leftarrow pos + \text{available move}$ ;
15          $g(n) = cost + \text{comparelist}(\text{available move})$ ;
16         if  $g(n) < dist[n]$  then
17              $move[n] \leftarrow \text{available move}$ ;
18              $dist[n] \leftarrow g(n)$ ;
19             if  $n$  not in  $closeSet$  then
20                  $openSet[n] \leftarrow g(n) + h(n)$ ;
21 until  $openSet$  is empty;
22 return  $\infty$ ;
```

---