Report for project 1: Solve MSA problem in three algorithms

Personal Information

Kaixiang Yang (杨凯翔, 519030910240, F1903302), SEIEE, SJTU (flying_feixiang@sjtu.edu.cn)

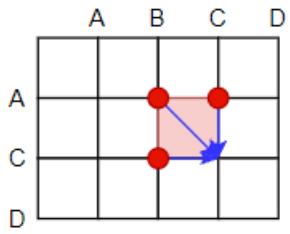
Task Description

In this project, we need to implement three algorithms to solve MSA problem including Dynamic programming, A star search and genetic algorithm. In the former two algorithms, they can find optimal solution, while I only can find a suboptimal solution. In total, there are 5 queries for pairwise alignment and 2 queries for three-sequence alignment, used to align with the sequences in Database, which contains 100 sequences.

Analysis and Implementation

Dynamic Programming

According to the lecture, we can implement dynamic programming on a chessboard of $n \times m$ to find optimal solution for pairwise alignment, as it shows below.



The point with coordinate (i, j) represents the alignment of $S(0, i-1)$ and $T(0, j-1)$ and at each point, we can calculate the minimal cost and finally calculate the cost between $S(0, n-1)$ and $T(0, m-1)$. Then the state transition equation is

$$dp[i][j] = min(dp[i-1][j] + 2, dp[i][j-1] + 2, dp[i-1][j-1] + judge(S[i-1], T[j-1]))$$

$$judge(i, j) \text{ is } (i == j)?0 : 3$$

Point (i, j) may comes from $(i-1, j)$, $(i, j-1)$ and $(i-1, j-1)$. As for the former two situation, they represent that add a **GAP** to T or S , which means plus 2 to the cost. In the latter situation, the cost depends on the two characters (**MATCH or MISMATCH**). On the other hand, if we record all the state transitions of the path from start point to the final point, we can acquire the alignment result of such two sequences easily.

Here is the main part of this algorithm (from 'dp.cpp')

```
for(int i = 0; i <= n; ++i) dp[i][0] = 2 * i; // init
for(int i = 0; i <= m; ++i) dp[0][i] = 2 * i; // init
for(int i = 1; i <= n; ++i)
    for(int j = 1; j <= m; ++j){
        if(s[i-1] == Database[t][j-1])
            dp[i][j] = min_3(dp[i-1][j-1], dp[i][j-1]+2, dp[i-1][j]+2);
         else dp[i][j] = min_3(dp[i-1][j-1]+3, dp[i][j-1]+2, dp[i-1][j]+2);
    }
```

After find the sequence in database that generates the minimal cost with query, we can do similar things again and record the path this time to get the alignment result.

```
for(int i = 0; i <= n; ++i) {DP[i][0] = 2 * i; path[i][0] = 3;}
for(int i = 0; i <= m; ++i) {DP[0][i] = 2 * i; path[0][i] = 1;}
for(int i = 1; i <= n; ++i)
    for(int j = 1; j <= m; ++j){
        if(S[i-1] == T[j-1]) {
             int M = min_3(DP[i-1][j-1], DP[i][j-1]+2, DP[i-1][j]+2);
             DP[i][j] = M;
             if(M == DP[i-1][j-1]) path[i][j] = 2;
             else if(M == DP[i][j-1] + 2) path[i][j] = 1;
             else if(M == DP[i-1][j] + 2) path[i][j] = 3;
         }
         else{
             int M = min_3(DP[i-1][j-1]+3, DP[i][j-1]+2, DP[i-1][j]+2);
             DP[i][j] = M;
             if(M == DP[i-1][j-1] + 3) path[i][j] = 2;
             else if(M == DP[i][j-1] + 2) path[i][j] = 1;
             else if(M == DP[i-1][j] + 2) path[i][j] = 3;
         }
     }
```

Then we can generate the alignment result by **path[i][j]**.

To reduce the time of apply space for the array, here I use std::vector to substitute two dimension array.

Besides, we also need to apply this algorithm to three-sequence alignment, and that is similar to pairwise alignment, with the difference of more state transitions, which is accomplished in the 3-dimension space. It is obvious that it has 7 former states to judge, and the main part is shown below.

```
for(int i = 1; i <= n; ++i)
    for(int j = 1; j <= m; ++j)
        for(int r = 1; r <= p; ++r){
            char a = s[i-1], b = Database[t][j-1], c = Database[k][r-1];
            int A[7] = {dp3[i-1][j-1][r-1]+judge3(a, b, c, 0), dp3[i-1][j-1][r]+judge3(a,       b, c,
1), dp3[i-1][j][r-1]+judge3(a, b, c, 2), dp3[i][j-1][r-1]+judge3(a, b, c, 3),       dp3[i-1][j][r]+4,
dp3[i][j-1][r]+4, dp3[i][j][r-1]+4};
            dp3[i][j][r] = min_7(A).first;
        }
```

Similarly, the process of acquire the alignment result is easy by recording the path.

A Search Algorithm*

1. Understand of A-star Search

In my opinion, A-star Search is similar with BFS adding some pruning, provided by the evaluation function

$$f(x) = g(x) + h(x)$$

$g(x)$ represents the current cost from start node to node x , and $h(x)$ is an estimation of the cost from node x to end, and $h(x)$ is a kind of heuristic function. We can prove that if $h(x)$ is admissible, which means that $h(x)$, a kind of estimation, could not be larger than the real cost, this algorithm will find the optimal solution. Besides, the performance of this algorithm depends on the pruning on BFS and if $h(x)$ is better, the performance will be better.

The core part of this algorithm is according to the value of $f(x)$, finding the node with smallest value, choosing that node to be searched, which is considered most probably have the minimal cost from start, via this node, to the end. So each time, choose the most probable node out from the open list, and then update the adjacent nodes' information and put that node to the closed list. If some point in the closed list has been updated, then put it back to the open list, because that means this node may have another path from start node to itself with less cost and maybe that is the optimal answer. From this perspective, we can also know that the performance of this algorithm depends on the performance of $h(x)$.

So I just implement this algorithm like BFS, and maintain two vector of node: open_list and closed_list. On the other hand, to get the alignment result, we can easily record the optimal path by adding an element to each node, which is its father's coordinates, which is also need to be updated when updating nodes in each loop.

2. Design of heuristic functions

To accomplish the algorithm, I prepare two kinds of heuristic functions for pairwise and three-sequence alignment:

1. Pairwise Alignment:

   From current node (i, j) to the end node (n, m) , we use $h(i, j) = |(n - i) - (m - j)| \times 2$ to estimate. Because considering the minimal cost path from (i, j) to (n, m) , the best situation is several horizontal or vertical steps with the cost of 2, and slant steps with the cost of 0 (the remain parts of two strings are the same). In this situation, the estimation is that $h(i, j)$, so this heuristic function is admissible.

2. Three-sequence Alignment:

   According to the references, there is an obvious conclusion that the cost of alignment of three sequences must be no more that the sum of costs between each two sequences among them. And then I design such a heuristic function. Because the cost from current node (i, j, k) to the end node (n, m, p) is the cost of alignment of the remain parts of the three sequences, then we can calculate the sum of three costs of each two remain parts as the estimation of the smallest cost from current node to the end. It is also admissible, so we will get the optimal solution.

Genetic Algorithm

In genetic algorithm, we can do many optimization to make the result more closed to the optimal answer, and I tried my best to get the best answer, and finally the result converges to suboptimal solution.

1. Encoding for Genes and Initialization

In each test, I calculate the 'smallest' cost of the alignment of two sequences or three sequences. So the gene is encoded by one possible result of alignment and then generate population to apply genetic algorithm. One gene consists of two or three arrays of boolean variables and the length of arrays and the corresponding cost. The array has the same number of 0 as the length of the corresponding sequence and the positions of 1 represent **GAP**. One array can be mapped to a result of alignment of that corresponding sequence. The following is an example.

When initializing a population, I randomly choose length between $\max(len1, len2)$ and $1.25 \times \max(len1, len2)$ for one individual in the population.

```
sequences: ABCD, ACD
gene: [0110100] and [0011101], len=7, cost=9
from gene to alignment result:  A--B-CD
                                AC---D-
```

2. Reproduce and Crossover

I design a common crossover for genes. For every two random parents, I randomly select a position to cut one gene and find the appropriate position of arrays in another gene to reproduce two children. In this process, because of the different sequence of 0 and 1, the corresponding positions of arrays in another gene may not aligned and to generate two genes and promises that each array in one gene has the same length, I supplement 1 in the middle of two parts. Then calculate two children's costs and choose the smaller one to put into the population. Here is an example.

```
sequences: ABCD, ACD
gene1: [0110100] and [0011101], len=7, cost=9
gene2: [00010] and [01010], len=5, cost=2
# '|' means the position of random cut
01|10100        00010
00|11101        01010
# then find the appropriate postion in gene2 to cut
01|10100        0|0010
00|11101        010|10
# Generate children, '*' represents the position of cut
01*0010         0 * 10100
00 * 10         010*11101
# Supplement 1 in the cut position to make children aligned
010010          01110100
001110          01011101
# Then calculate the children's costs and choose the smaller one to put into population
```

3. Mutate

For one gene, I design three kinds of mutation: (take pairwise alignment as example)

1. Delete a 1 from each array randomly

```
010010  ->  01000
001110  ->  00110
```

2. Add a 1 to each array randomly

```
010010  ->  0101010
001110  ->  0011101
```

3. Swap a 0 and a 1 in one array

```
010010  ->  011000
001110  ->  001110
```

Define three mutation rates and mutation percentages to finish these kinds of mutation.

4. Select

To avoid the population premature convergence at a worse answer, we need not only to choose the best part of population but also select some gene looks bad to maintain the diversity of the population. And I choose to inherit the top 40% best individual to the next generation, and for the others , I use tournament selection(锦标赛选择法): each time compare two individuals and choose the better one and put that two individuals back to the set.

5. Evolutionary Cycle

To acquire better result, we do not set up the generations of evolution, and use 'evolutionary cycles' to replace. If we find the difference of adjacent two generations' result is less than a constant number (such as 5) we reduce the cycle by 1.

```
while(cyc){
    int t = update3(que, i, j, P);
    if(temp - t < 5) --cyc, temp = t;
}
```

Complexity

Because all of them test one set of sequences to find the solution of alignment every time, so we discuss the complexity for one test here. As for the whole time complexity, just add the time of each test for each query.

1. Dynamic Programming

In one search, this algorithm need to calculate each node in the searching space, so the time complexity is $O(n \times m)$ in pairwise alignment.

So the time complexity of general k-sequence alignment is $O(\prod_{i=1}^{k} S[i].\,size())$ and the space complexity is the same. On the other hand, we can abandon those data of nodes, which will not be used in the following process, to save some space, then the space complexity will be $O(\frac{\prod_{i=1}^{k} S[i].size()}{\max_{1\leqslant i\leqslant k}(S[i].size())})$.

When considering the whole time complexity, just add each test for each query.

2. A-star Search

In this algorithm, I implemented by the method of maintaining open list and closed list. Then it may update the node in the closed list and put that node back into open list, as a consequence of which, we cannot precisely tell how much the pruning it accomplished depending on the heuristic function and in the process of searching, one node may be pushed into open list more than one time. So we can only give out a general upper bound.

Once choose a node with smallest evaluation to pop out of open list and then search no more than 3 nodes (in this process, also need to traverse open list or closed list), so the time complexity is

$$O(L \times 3^{L}) \quad (L = \sum_{i=1}^{k} S[i].\,size())$$

3. Genetic Algorithm

Some notations: C: number of evolution cycle G_{aver}: average number of generations in one cycle

S: Population size r: reproduce rate P_m: mutation percentage r_m: mutation rate L: extension rate in one gene

T_{gen}: represents the time for an evolution (iteration) $length$: the maximum length of the sequences to be aligned

(Here, the mutation means the sum of three kinds of mutation).

Because in each generation the average length of genes will not remain unchanged (depending on the number of 1 s in the gene), then we can estimate the average length of gene as $(1 + L) \times length$, because L is an experimental result from the length of general alignment of sequences (I also use this as the upper bound of initialization length). Besides, in this algorithm, I use evolution cycle instead of common number of evolution, then G_{aver} is a statistic variable, because I cannot tell how much generations in each cycle precisely.

Then the time complexity is

$$O(init\_time + update\_time) = O(C \times G_{aver} \times T_{gen})$$

$$T_{gen} = reproduce\_time + sort\_time + select\_time$$

$$\therefore O(T_{gen}) = O(S \times length \times (1+L) \times (1+r) + S \times (1+r) \times \log(S \times (1+r)) + S)$$

$$\therefore \text{It is } O(C \times G_{aver} \times S \times length \times (1+L) \times (1+r) + C \times G_{aver} \times S \times (1+r) \times \log(S \times (1+r)))$$

Because L and r are constant numbers I have defined and they are small. Then,

$$\text{Time Complexity is } O(C \times G_{aver} \times S \times length + C \times G_{aver} \times S \times \log(S))$$

$C \times G_{aver}$ is the number of iterations determines by the speed of convergence. It is determined by the input sequences, so we can use a function $N(\mathbb{S})$ to represent it ($\mathbb{S}$ is the set of input sequences). Then the time complexity is

$$O(N(\mathbb{S}) \times S \times (length + \log(S))$$

1. If $\log S > length$ then $O(N(\mathbb{S}) \times S^2)$.
2. If $\log(S) < length$ then $O(N(\mathbb{S}) \times S \times length)$.

Results and Running time

The programs are compiled with argument -O2. My laptop has an Intel i5-8265U@1.6GHz CPU.

1. Dynamic Programming & A star Searching

```
Pairwise Alignment:
Query 1 : cost = 112
          alignment with: number (22) in Database
          alignment result:
KJX---XJAJKPXKJ-JX-JKPXKJXXJA--J-KPXKJ-JX-JKPXKJXXJAJKPXKJ-XXJAJ--KH--X-KJ-XXJAJKPXKJXXJAJKHXKJXX--
--XHAPXJAJ-XXXJAJXDJAJX--XXJAPXJAHXXXJAJXDJAJX--XXJAJ-XXXJPPXJAJXXXHAPXJAJXXXJAJ--X--XXJAJ--X--XXJA
Running Time is 0s(dp) 0.640625s(astar)

Query 2 : cost = 107
          alignment with: number (64) in Database
          alignment result:
ILOTGJJLABWTSTG-GONXJ-----MUTUXSJHKWJHCT-OQHWGAGIWLZHWPKZULJTZWAKBWHXMIKLZJGLXBPAHOHVOLZWOSJJLP-O
IPOTWJJLAB-KS-GZG-WJJKSPPOPHT--SJ-EWJHCTOOTH-RAXBKLBHWPKZULJPZKAKVKHXUIKLZJGLXBTGHOHBJLZPPSJJ-PJO
Running Time is 0s(dp) 0.765625s(astar)

Query 3 : cost = 113
          alignment with: number (46) in Database
          alignment result:
IHKKKRKKKKKKXGWGKKKPKSKKKK--KBKKKP-KHKKXKK--BSK-K-PKWKKLKSKRKKWXKPKK-BK-
KKPKTSKHKKKKLADKKYPKKKOPHKKBWWLPPWKK-
I---K-BSKKKK--W-KKK-K-KKKKWWK-KKKPGK-KK-KKXXGGKRKWWKWKKPK-K-KKKXK-KKRWKMKKPK--KWPKKK---KK-PGKK---KLBKW--
--WKKJ
Running Time is 0.015625s(dp) 1.03125s(astar)

Query 4 : cost = 148
          alignment with: number (19) in Database
          alignment result:
MPPPJPX-PGP-JPPPXPPPJPJPPPXPPPPSPPJJJPPXXP-PPPPJPPPXPPXIPJMMMXPKPSVGULMHHZPAWHTHKAAHHUPAONAP-JSWPP-JGA
--OPJPXJP-PMJPPMX-PP-MJ-PPXJPPOXPPXJJPJXXPXJPPOJPPMXPPOGP--PXXP-P----OM---PPXXPPOXPPXJP-QXPPBJ-PPPXPPX
Running Time is 0s(dp) 0.953125s(astar)

Query 5 : cost = 131
          alignment with: number (58) in Database
          alignment result:
IPPVKBKXWXKHSA-PHVXXVOJMRAK--KPJVLLJBWKOLLJKXHGXL--LCPAJOBKPGXBATGXMPOMCVZTAXV-P--AGKXGOMJQO----
LJGWGKXLQ
ITPVKWKSKXKXUAXP-VHXVO-M-MKHYBPABLLOBGKOLLJGXZGXLSOL--AMOGKIGXBATBXMPJTCVMTAXVMPWWA---WOM--
OUPHHZBITKKXLK
Running Time is 0s(dp) 0.8125s(astar)
```

```
Three-sequence Alignment:
Query 1 : cost = 290
          alignment with: number (11) and number (71) in Database
          alignment result:
IPZJJ-LMLTKJULOSTKTJOGLKJOBLTXGKTPLUWWKOMOYJBGALJUKLGLOSVHWBPGWS-LUKOBSOPLOOKU---KSARPPJ
IPZJJPL-LTHUULOSTXTJOGLKJGBLLMMPJPLUWGKOMOYJBZA-YUKOFLOSZHGBP-HXPLXKJBXKJL-AUUOJHW-TWWPQ
IPMJJ-LLLTHOULOSTMAJIGLKJPVLLXGKTPLTWWKOMOYJBZP-YUKLILOSZHGBPGWX-LZKJBSWJLPJUU-MHK-TRAP-
Running Time is 47.9531s(dp) 3002.28s(astar)

Query 2 : cost = 122
          alignment with: number (3) and number (47) in Database
          alignment result:
--IWTJBGTJGJTWGBJTPKHAXHAGJJSJJPPJAPJHJHJHJHJHJHJHJHJPKSTJJUWXHGPHGALKLPJTPJPGVXPLBJHHJPKWPPDJSG
--IWTJBGTJGJTWGBJTPKHAXHAGJJXJJKPJTPJHJHJHJHJHJHJHJHJHKUTJJUWXHGHHGALKLPJTPJPGVXPLBJHH----------
WPIWTJBGTJGJTHGBJOXKHTXHAGJJXJJPPJTP--JHJHJHJHJHJHJHJPKUAJJUWXHGHHGALKLPJTPJPGVXPLBJHHJPK-------
Running Time is 38.6406s(dp) 4420.94s(astar)
```

2. Genetic Algorithm

```
Pairwise Alignment:
Query 1 : cost = 136
          alignment with: number (22) in Database
          alignment result:
K-JXXJAJKPX-KJJXJKPXKJXXJAJKPXKJJXJKPXKJ---XXJAJKPXKJXXJAJKHXKJXXJAJKPXKJXX--JAJKHXKJXX
XHAPXJAJXXXJAJXDJAJX--XXJAPXJAHXXXJAJXDJAJXXXJAJXXXJPPXJAJXXXHAPXJAJXXXJAJXXXJAJX-XXJ-A
Running Time is 1.375s

Query 2 : cost = 121
          alignment with: number (64) in Database
          alignment result:
ILOTGJJLABWTSTGGONXJMUTU-XSJHKWJHCTOQHWGAGIWLZHWPKZULJTZWAKBWHXMIKLZJGLXBPAHOHVOLZWOSJJLPO
IPOTWJJLABKSGZGWJJKSPPOPHTS-JEWJHCTOOTHRAXBKLBHWPKZULJPZKAKVKHXUIKLZJGLXBTGHOHBJLZPPSJJPJO
Running Time is 1.51562s

Query 3 : cost = 153
          alignment with: number (46) in Database
          alignment result:
IHKK-KRKKKKKKXGWGKKKPKSKK-KKKBKKKPKHKKXKKBS-KKPKWKKLKSKRKKWXKPKKBKKKPKTSKHK-KKKLADKKYPKKKOPHKK-BWWLPPWKK
I-KBS--KKKKWK-K-KKKKKKWWKKKKPGK-KKK--KXXGGKRKWWKWKK-P-KKKKKXKKKRWKM--K----KPK-K--WPK-KKKKPGKKKLBKWW--KKJ
Running Time is 1.73438s

Query 4 : cost = 171
          alignment with: number (19) in Database
          alignment result:
MP-PPJPXPGPJPPPXPPPJPJPPPXPPPPSPPJJJPPXXPPPPPJPPPXPPXIP--JMMMXPKPSVGULMHHZPAW--HTHKAA-HHUPAONAPJSWP-PJGA
OPJPXJP--PMJPPMXPP-M-JP-PXJPPOXPPXJJP-JXXPX--JPPOJPPMXPPOGPPXXP-P----OMP--PXXPPOXPPXJPQXPPB----JPPPXP-PX
Running Time is 1.67188s

Query 5 : cost = 141
          alignment with: number (58) in Database
          alignment result:
IPPVKBKXWXKHSAPHVXXVOJMRAKKPJVLLJBWKOLLJKXHGXLLCPAJOBKPGXBATGXMPOMCVZTAXVP---AGK-XGOMJQOLJGWGKXLQ
ITPVKWKSKXKXUAXPVHXVOMMKHYBPABLLOBGKOLLJGXZGXLSOLAMOGKIGXBATBXMPJTCVMTAXVMPWWAW-OM-OUPHHZBITKKXLK
Running Time is 1.73438s

Three-sequence Alignment:
Query 1 : cost = 317
          alignment with: number (5) and number (71) in Database
          alignment result:
IPZJJLMLTKJULOSTKTJOGLKJOBLTXGKTPLUWWKOMOYJBGALJUKLGLOSVHWBPGWSLUKOBSOPLOOKUKSARPPJ
IPZJILPLTHOUKOSTMPJOGLKJFBLPMJKJTLBWWKOMOYJBJPLJSKLFLOSZHGVPGJSLWGXBHOHLV-WUKWXXTWJ
IPMJJLLLTHOULOSTMAJIGLKJPVLLXGKTPLTWWKOMOYJBZ-PYUKLILOSZHGBPGWXLZKJBSWJLPJUUMHKTRAP
Running Time is 118.578s
```

```
Query 2 : cost = 503
          alignment with: number (41) and number (49) in Database
          alignment result:
IWTJBGTJGJTWGBJTPKHAXHAGJJSJJPPJAP-JHJHJHJHJHJ-HJHJHJPKSTJJUWXHGPHGA---LKLP-JTPJ-PGVXPLBJHH-JPKWPPDJSG
IPPJOJ--PJ-JJPJ-OJ-PP-A--JO-J-P-KJ-----OJP--JJJPPAJOJP--JJJPPA-JOJ-P--------J--J-J-P-PA--J-OJP--J--JJP
IPHJ-JJ-PJJJPAJ--O-JP----J-JJPAJOJPP--J-JJ-KHJ-JJPJJJPJ-JJP-A-J-OJP-JJJ-PHJJJ-PJJJ---PA-J--OJPJJJPHJJ-
Running Time is 121.562s
```

Conclusion

In this project, DP and A star algorithms are optimal while genetic algorithm may not acquire an optimal solution. According to the time complexity, the runtime for A star is much larger than DP. In my implement of genetic algorithm, I tried to acquire the nearest solution, and finally, it performs well in the pairwise alignment and the first query in three-sequence alignment, but the solution for the second query in three-sequence alignment. I guess that it is affected by the real result of the alignment, that is, in the optimal solution that three sequences equal with each other in the most middle part of sequences, and they add several **GAP** at the right side and left side of the sequences. In my implement of **GA** , such situation cannot easy get normally, actually, this form can be generated by special mutation, which is not a general way.