바이오스핀

alignment

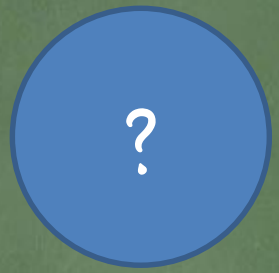
강양제

이 문서는 나눔글꼴로 작성되었습니다. 설치하기

목차

1. Sequence alignment
2. Dynamic programming
3. Tandem repeat

1. Sequence alignment : 배경설명



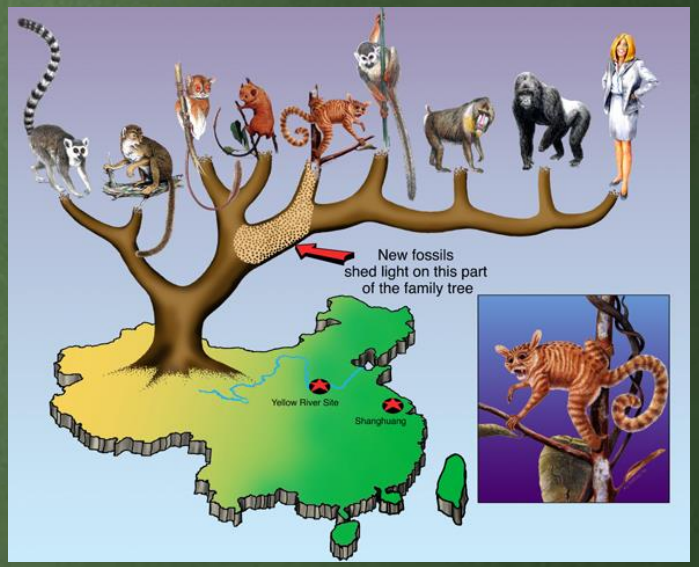
공통 조상

서로 다른 진화의 길

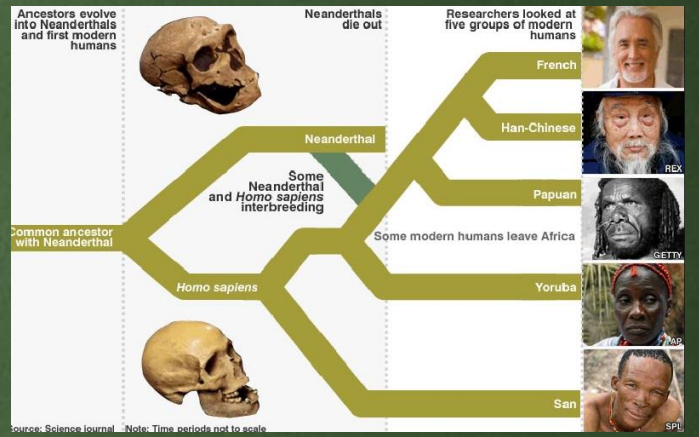
인간

오랑우탄

속 분화



종 분화



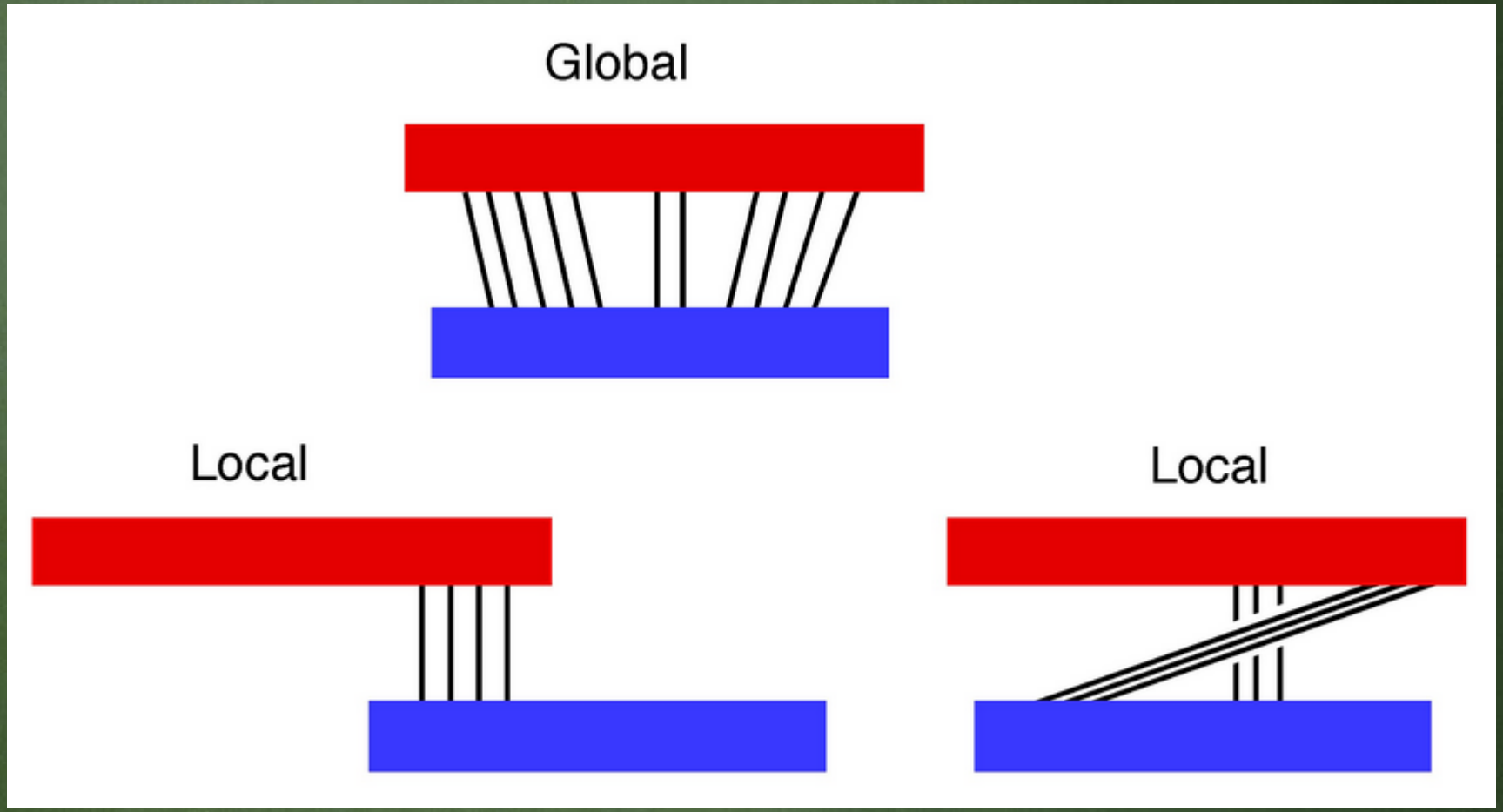
indel

base difference (SNP)

```
gi|237757283|ref|NM_007294.3|          TTGTTGGCTTAGGGCTGGAAGCACAGAGTGGCTTGGCCTCAAGAGAATAG 6200
gi|395749083|ref|XM_003778832.         TTG---GCTTAGGGCTGGAAGCACAGAGTGTCTTGGCTTCAAGAGAATAG 6197
                                       ***   ********************** ****** ************
```

- 인간 유전체에서 알려진 유전자의 기능을 통해서 오랑우탄의 유전체에 존재하는 유전자의 기능을 추정한다.
- 비슷한 유전자를 찾는다 -> Alignment
- 비슷한 정도를 계산한다 -> 통계

1.1 Global alignment vs local alignment



1.2 Direct alignment

RNDKPKFSTARN
RNQKPKWWTATN
++-+++--++-+

- 같은 position의 단백질서열을 비교해도 된다고 가정했을 때
- 그러나 보통 직접 비교를 하면 안됨

RNDKPPPPKFSTARN
RNQKP---KWWTATN
++-++---+--++-+

- 특정 서열이 반복되는 진화를 겪었을 경우는 비교대상에 gap을 추가해서 비교해야 할 필요가 있다.

1.2 Direct alignment

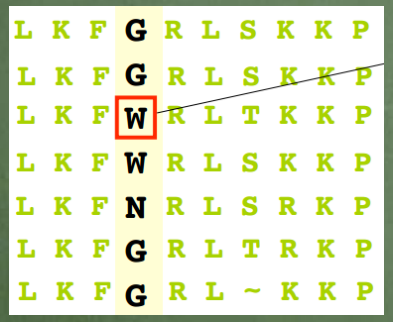
Direct alignment 실습

```
import numpy as np
def simplescore(seq1,seq2):
    conv_seq1 = map(ord,list(seq1))
    conv_seq2 = map(ord,list(seq2))
    # count match
    score = np.equal(conv_seq1,conv_seq2).astype(int).sum()
    score = score - np.not_equal(conv_seq1,conv_seq2).astype(int).sum()
    # gap
    ngaps = seq1.count('-') + seq2.count('-')
    score = score - ngaps
    print(score)
simplescore('AGTCGATCGATT','AGTCGATCGATT')
simplescore('AGTCGATCGATT','AGTCGATCGAAT')
simplescore('AGTCGATCGATT','AGTCGATCGA-T')
print "hi,hello"
```

```
12
10
9
hi,hello
```

1.2 Statistical alignment

$$\text{log odds ratio} = \log_2(\text{observed/expected})$$



How do you calculate log-odds?

$P(G) = 4/7 = 0.57$

Observed probability of G->G

$q_{GG} = P(G->G) = 6/21 = 0.29$

Expected probability of G->G,

$e_{GG} = 0.57*0.57 = 0.33$

$\text{odds ratio} = q_{GG}/e_{GG} = 0.29/0.33$

$\text{log odds ratio} = \log_2(q_{GG}/e_{GG})$

If the 'lod' is < 0., then the mutation is less likely than expected by chance. If it is > 0., it is more likely.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**5**	−2	−1	−2	−1	−1	−1	0	−2	−1	−2	−1	−1	−3	−1	1	0	−3	−2	0
R	−2	**7**	−1	−2	−4	1	0	−3	0	−4	−3	3	−2	−3	−3	−1	−1	−3	−1	−3
N	−1	−1	**7**	2	−2	0	0	0	1	−3	−4	0	−2	−4	−2	1	0	−4	−2	−3
D	−2	−2	2	**8**	−4	0	2	−1	−1	−4	−4	−1	−4	−5	−1	0	−1	−5	−3	−4
C	−1	−4	−2	−4	**13**	−3	−3	−3	−3	−2	−2	−3	−2	−2	−4	−1	−1	−5	−3	−1
Q	−1	i	0	0	−3	**7**	2	−2	1	−3	−2	2	0	−4	−1	0	−1	−1	−1	−3
E	−1	0	0	2	−3	2	**6**	−3	0	−4	−3	1	−2	−3	−1	−1	−1	−3	−2	−3
G	0	−3	0	−1	−3	−2	−3	**8**	−2	−4	−4	−2	−3	−4	−2	0	−2	−3	−3	−4
H	−2	0	1	−1	−3	1	0	−2	**10**	−4	−3	0	−1	−1	−2	−1	−2	−3	2	−4
I	−1	−4	−3	−4	−2	−3	−4	−4	−4	**5**	2	−3	2	0	−3	−3	−1	−3	−1	4
L	−2	−3	−4	−4	−2	−2	−3	−4	−3	2	**5**	−3	3	1	−4	−3	−1	−2	−1	1
K	−1	3	0	−1	−3	2	1	−2	0	−3	−3	**6**	−2	−4	−1	0	−1	−3	−2	−3
M	−1	−2	−2	−4	−2	0	−2	−3	−1	2	3	−2	**7**	0	−3	−2	−1	−1	0	1
F	−3	−3	−4	−5	−2	−4	−3	−4	−1	0	1	−4	0	**8**	−4	−3	−2	1	4	−1
P	−1	−3	−2	−1	−4	−1	−1	−2	−2	−3	−4	−1	−3	−4	**10**	−1	−1	−4	−3	−3
S	1	−1	1	0	−1	0	−1	0	−1	−3	−3	0	−2	−3	−1	**5**	2	−4	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	2	**5**	−3	−2	0
W	−3	−3	−4	−5	−5	−1	−3	−3	−3	−3	−2	−3	−1	1	−4	−4	−3	**15**	2	−3
Y	−2	−1	−2	−3	−3	−1	−2	−3	2	−1	−1	−2	0	4	−3	−2	−2	2	**8**	−1
V	0	−3	−3	−4	−1	−3	−3	−4	−4	4	1	−3	1	−1	−3	−2	0	−3	−1	**5**

Figure 2.2 *The* BLOSUM50 *substitution matrix. The log-odds values have been scaled and rounded to the nearest integer for purposes of computational efficiency. Entries on the main diagonal for identical residue pairs are highlighted in bold.*

In [61]:

```
BLOSUM50 = array([[5,-2,-1,-2,-1,-1,-1, 0,-2,-1,-2,-1,-1,-3,-1, 1, 0,-3,-2, 0],
                  [-2, 7,-1,-2,-1, 1, 0,-3, 0,-4,-3, 3,-2,-3,-3,-1,-1,-3,-1,-3],
                  [-1,-1, 7, 2,-2, 0, 0, 0, 1,-3,-4,-0,-2,-4,-2,-1, 0,-4,-2,-3],
                  [-2,-2, 2, 8,-4, 0, 2,-1,-1,-4,-4,-1,-4,-5,-1, 0,-1,-5,-3,-4],
                  [-1,-4,-2,-4,13,-3,-3,-3,-3,-2,-2,-3,-2,-2,-4,-1,-1,-5,-4,-1],
                  [-1,-1, 0, 0,-3, 7, 2,-2, 1,-3,-2, 2, 0,-4,-1,-0,-1,-1,-1,-3],
                  [-1, 0, 0, 2,-3, 2, 6,-3, 0,-4,-3, 1,-2,-3,-1,-1,-1,-3,-2,-3],
                  [ 0,-3, 0,-1,-3,-2,-3, 8,-2,-4,-4,-2,-3,-4,-2, 0,-2,-3,-3,-4],
                  [-2, 0, 1,-1,-3, 1, 0,-2,10,-4,-3, 0,-1,-1,-2,-1,-2,-3,-1, 4],
                  [-1,-4,-3,-4,-2,-3,-4,-4,-4, 5, 2,-3, 2, 0,-3,-3,-1,-3,-1, 4],
                  [-2,-3,-4,-4,-2,-2,-3,-4,-3, 2, 5,-3, 3, 1,-4,-3,-1,-2,-1, 1],
                  [-1, 3, 0,-1,-3, 2, 1,-2, 0,-3,-3, 6,-2,-4,-1, 0,-1,-3,-2,-3],
                  [-1,-2,-2,-4,-2, 0,-2,-3,-1, 2, 3,-2, 7, 0,-3,-2,-1,-1, 0, 1],
                  [-3,-3,-4,-5,-2,-4,-3,-4,-1, 0, 1,-4, 0, 8,-4,-3,-2, 1, 4,-1],
                  [-1,-3,-2,-1,-4,-1,-1,-2,-2,-3,-4,-1,-3,-4,10,-1,-1,-4,-3,-3],
                  [ 1,-1, 1, 0,-1, 0,-1, 0,-1,-3,-3, 0,-2,-3,-1, 5, 2,-4,-2,-2],
                  [ 0,-1, 0,-1,-1,-1,-1,-2,-2,-1,-1,-1,-1,-2,-1, 2, 5,-3,-2, 0],
                  [-3,-3,-4,-5,-5,-1,-3,-3,-3,-3,-2,-3,-1, 1,-4,-4,-3,15, 2,-3],
                  [-2,-1,-2,-3,-3,-1,-2,-3, 2,-1,-1,-2, 0, 4,-3,-2,-2, 2, 8,-1],
                  [ 0,-3,-3,-4,-1,-3,-3,-4,-4, 4, 1,-3, 1,-1,-3,-2, 0,-3,-1, 5]])


PBET = 'ARNDCQEGHILKMFPSTWYV'


def statistical_algn(mat,abet,seq1,seq2,gap=-8):
    score = 0
    min_length = min([len(seq1),len(seq2)])
    for pos in range(min_length):
        if seq1[pos] == '-' or seq1[pos] == '-' and seq1[pos] != seq2[pos]:
            score += gap
        elif seq1[pos] == '.' or seq2[pos] == '.':
            pass
        else:
            n1 = abet.index(seq1[pos])
            n2 = abet.index(seq2[pos])
            #print n1,n2
            score += mat[n1,n2]
    print score


statistical_algn(BLOSUM50,PBET,"RNDKPKFSTARN","RNQKPKWWTATN")
```

```
49
```

1.3 brute Force alignment

- 한칸씩 밀어보면서 scoring!
- 서열이 길어지면 힘들다.

- The number of possible global alignments between two sequences of length N is

$$\frac{2^{2N}}{\sqrt{\pi N}}$$

Brute Force Alignment

In [67]:
```
def Brute(seq1,seq2):
    len_seq1, len_seq2 = len(seq1), len(seq2)
    for i in range(len_seq1):
        print seq1
        print '.'*i+seq2
        statistical_align(BLOSUM50,PBET,seq1,'.'*i+seq2)
        print '.'*i+seq1
        print seq2
        statistical_align(BLOSUM50,PBET,'.'*i+seq1,seq2)
Brute("RNDKPKFSTARN","RNQKPKWWTATN")
```

```
RNDKPKFSTARN
RNQKPKWWTATN
49
RNDKPKFSTARN
RNQKPKWWTATN
49
RNDKPKFSTARN
.RNQKPKWWTATN
-12
.RNDKPKFSTARN
RNQKPKWWTATN
-5
RNDKPKFSTARN
..RNQKPKWWTATN
-9
..RNDKPKFSTARN
RNQKPKWWTATN
2
RNDKPKFSTARN
...RNQKPKWWTATN
-9
...RNDKPKFSTARN
RNQKPKWWTATN
-9
RNDKPKFSTARN
....RNQKPKWWTATN
-16
```

2. Dynamic Programming

다이나믹 프로그래밍

동적 계획법(Dynamic Programming). 줄여서 DP라고 많이 부른다. 특정 범위까지의 값을 구하기 위해서 그것과 다른 범위까지의 값을 이용하여 효율적으로 값을 구하는~~왠지 모르게 허세가 넘치는~~ 알고리즘이다.

엄밀히, 동적 계획법은 알고리즘이라기보단 문제해결 패러다임에 가깝다. 동적 계획법은 "어떤 문제를 풀기위해 그 문제를 더 작은 문제의 연장선으로 생각하고, 과거에 구한 해를 활용하는" 방식의 알고리즘을 총칭한다. 동적 계획법을 써야 좋은 효과를 얻을 수 있는 문제들은 주로 Optimal Substructure라고 불리는 구조를 가진 것들이다. 쉽게 이야기하면, 뭘 구하기 위해서 했던 계산을 또하고 또하고 계속해야하는 류의 문제를 풀 때 동적 계획법이 효과를 발휘한다는 것.

https://mirror.enha.kr/wiki/%EB%8B%A4%EC%9D%B4%EB%82%98%EB%AF%B9%20%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D

2.1 Scoring matrix

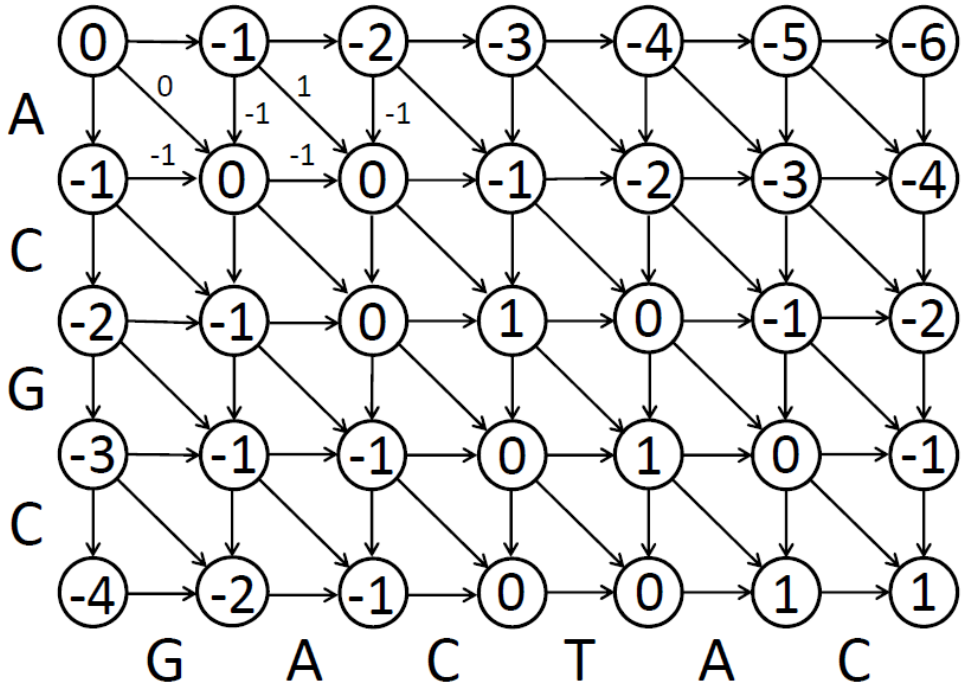
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**5**	−2	−1	−2	−1	−1	−1	0	−2	−1	−2	−1	−1	−3	−1	1	0	−3	−2	0
R	−2	**7**	−1	−2	−4	1	0	−3	0	−4	−3	3	−2	−3	−3	−1	−1	−3	−1	−3
N	−1	−1	**7**	2	−2	0	0	0	1	−3	−4	0	−2	−4	−2	1	0	−4	−2	−3
D	−2	−2	2	**8**	−4	0	2	−1	−1	−4	−4	−1	−4	−5	−1	0	−1	−5	−3	−4
C	−1	−4	−2	−4	**13**	−3	−3	−3	−3	−2	−2	−3	−2	−2	−4	−1	−1	−5	−3	−1
Q	−1	1	0	0	−3	**7**	2	−2	1	−3	−2	2	0	−4	−1	0	−1	−1	−1	−3
E	−1	0	0	2	−3	2	**6**	−3	0	−4	−3	1	−2	−3	−1	−1	−1	−3	−2	−3
G	0	−3	0	−1	−3	−2	−3	**8**	−2	−4	−4	−2	−3	−4	−2	0	−2	−3	−3	−4
H	−2	0	1	−1	−3	1	0	−2	**10**	−4	−3	0	−1	−1	−2	−1	−2	−3	2	−4
I	−1	−4	−3	−4	−2	−3	−4	−4	−4	**5**	2	−3	2	0	−3	−3	−1	−3	−1	4
L	−2	−3	−4	−4	−2	−2	−3	−4	−3	2	**5**	−3	3	1	−4	−3	−1	−2	−1	1
K	−1	3	0	−1	−3	2	1	−2	0	−3	−3	**6**	−2	−4	−1	0	−1	−3	−2	−3
M	−1	−2	−2	−4	−2	0	−2	−3	−1	2	3	−2	**7**	0	−3	−2	−1	−1	0	1
F	−3	−3	−4	−5	−2	−4	−3	−4	−1	0	1	−4	0	**8**	−4	−3	−2	1	4	−1
P	−1	−3	−2	−1	−4	−1	−1	−2	−2	−3	−4	−1	−3	−4	**10**	−1	−1	−4	−3	−3
S	1	−1	1	0	−1	0	−1	0	−1	−3	−3	0	−2	−3	−1	**5**	2	−4	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	2	**5**	−3	−2	0
W	−3	−3	−4	−5	−5	−1	−3	−3	−3	−3	−2	−3	−1	1	−4	−4	−3	**15**	2	−3
Y	−2	−1	−2	−3	−3	−1	−2	−3	2	−1	−1	−2	0	4	−3	−2	−2	2	**8**	−1
V	0	−3	−3	−4	−1	−3	−3	−4	−4	4	1	−3	1	−1	−3	−2	0	−3	−1	**5**

Figure 2.2 *The* BLOSUM50 *substitution matrix. The log-odds values have been scaled and rounded to the nearest integer for purposes of computational efficiency. Entries on the main diagonal for identical residue pairs are highlighted in bold.*

를 이용한다!

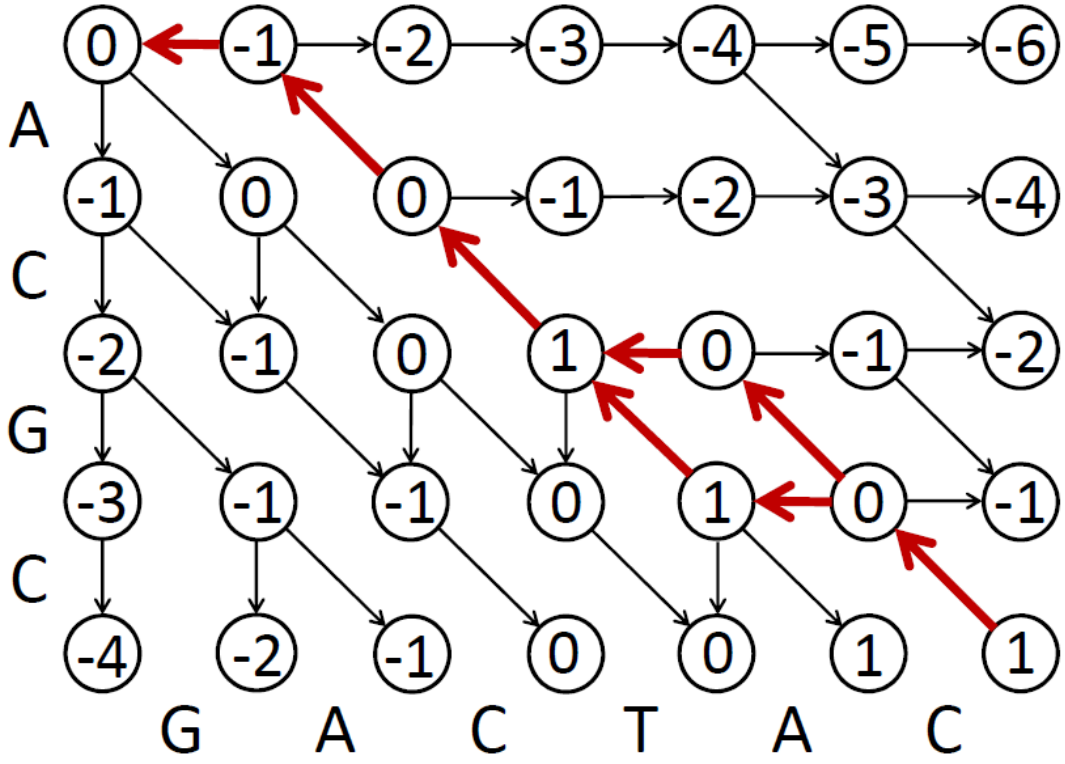
$$SIM(i,j) = max\begin{cases} SIM(i-1, j-1) + s(x_i, y_j) & x_i \text{ and } y_j \text{ aligned} \\ SIM(i-1,\ j) + g & x_i \text{ aligned with a null} \\ SIM(i,\ j-1) + g & y_j \text{ aligned with a null} \end{cases}$$

An Example



Scores: Match +1 Mismatch 0 Gap -1

Follow the traceback edges from the final node

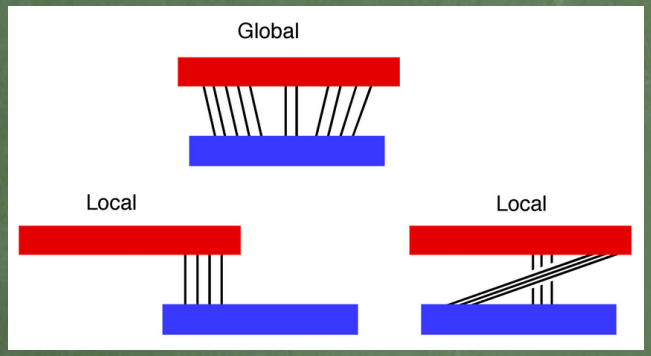


Optimal alignments:

```
-ACG-C
GACTAC
```

and

```
-AC-GC
GACTAC
```



The Smith-Waterman Algorithm: Traceback

Optimal local alignments, or *subalignments*:

AC–G
ATTG

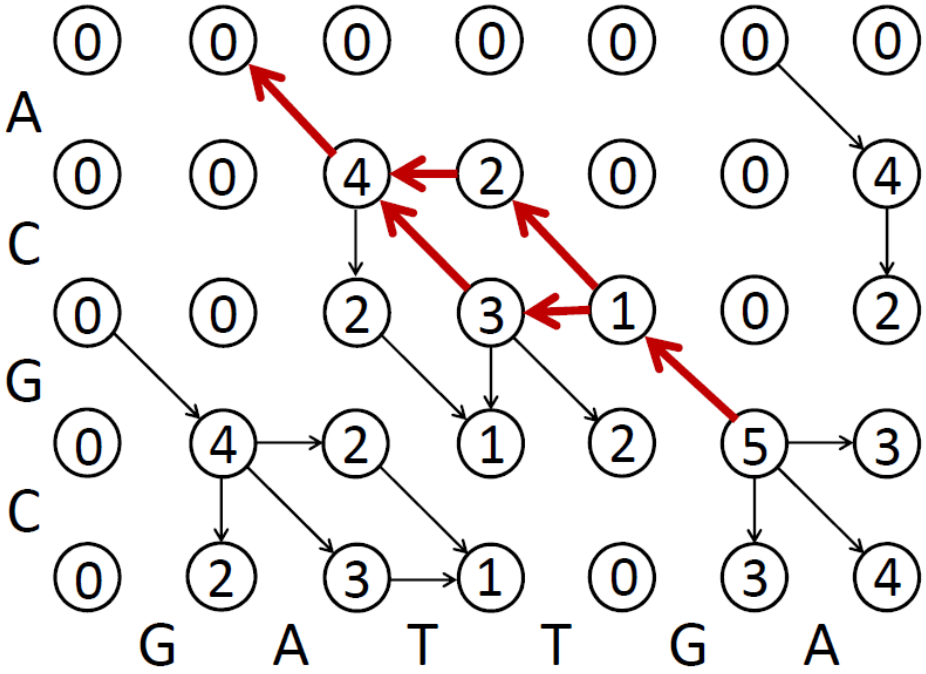
and

A–CG
ATTG

Questions:

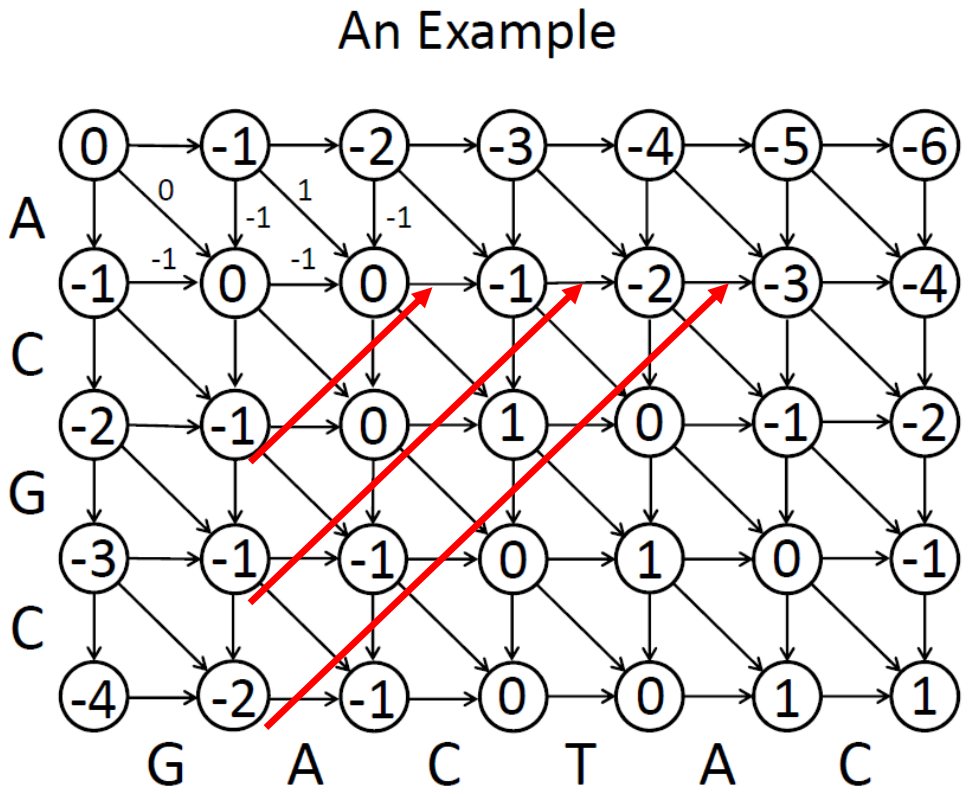
Can one find other *locally optimal* subalignments?

How can they be defined?



Scores: Match +4 Mismatch -1 Gap -2

2.1 Efficient programming



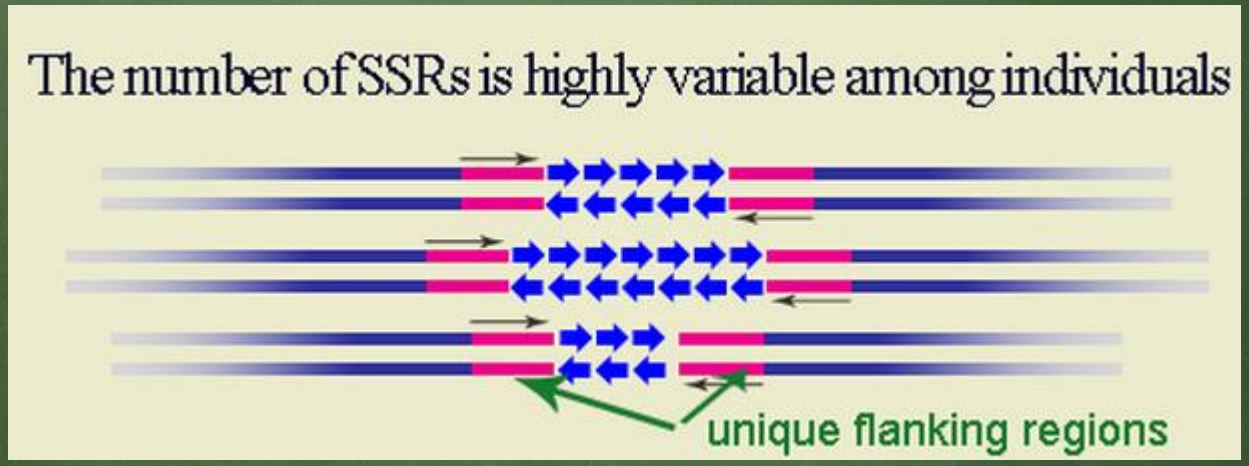
한번에 여러 개를 계산한다는 장점

1. Diagonal position 계산
2. 각 대각선을 한번에 계산
3. 빠르다

3. Tandem repeat

TC TC TC TC $(TC)_4$
ATTC ATTC ATTC $(ATTC)_3$
TGTAT ACACAC TGTAT ACACACAC TGTAT AC $((TGTAT)_1(AC)_*)_3$

Simple sequence repeat !



3.1. Hauth solution

1. 어떤 서열에서
2. word 추출
3. location 추출
4. 거리 추출
5. 거리 별 분포
6. repeat 추출

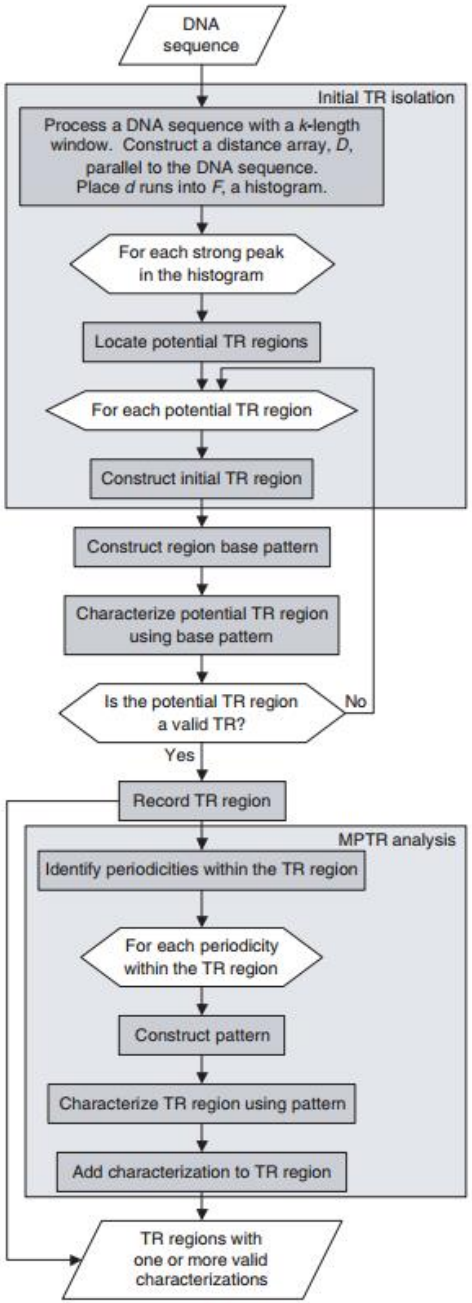


Fig. 1. Basic identification algorithm locates TR regions in a DNA sequence and characterizes the region using one or more patterns.

이 문서는 나눔글꼴로 작성되었습니다. 설치하기