부분서열정렬 개선 기법을 사용한 효율적인 복수서열정렬에 관한 알고리즘

(An Efficient Method for Multiple Sequence Alignment using Subalignment Refinement)

김 진 † 정 우 철 †† 엄 상 용 †††
(Jin Kim) (Woocheol Jung) (Saangyong Uhmn)

요 약 단백질들의 복수서열정렬은 단백질 서열간의 관계를 유추할 수 있는 유용한 도구이다. 최적화된 복수서열정렬을 얻기 위해 사용되는 가장 유용한 방법은 dynamic programming이다. 그러나 dynamic programming은 특정한 비용함수를 사용할 수 없기 때문에 특별한 경우 최소의 비용을 가지는 복수서열정렬을 제공하지 못하는 문제점이 있다. 우리는 이러한 문제점을 해결하기 위하여 부분서열정렬 개선기법을 사용한 알고리즘을 제안하였으며, 이 알고리즘이 dynamic programming의 문제점을 효과적으로 해결함을 보였다.

키워드 : 복수서열정렬, 최적화 알고리즘, 서열정렬, 동적프로그래밍

Abstract Multiple sequence alignment is a useful tool to identify the relationships among protein sequences. Dynamic programming is the most widely used algorithm to obtain multiple sequence alignment with optimal cost. However, dynamic programming cannot be applied to certain cost function due to its drawback and cannot be used to produce optimal multiple sequence alignment. We propose sub-alignment refinement algorithm to overcome the problem of dynamic programming. Also we show proposed algorithm can solve the problem of dynamic programming efficiently.

Key words : multiple sequence alignment, optimization algorithm, dynamic programming

1. 서 론

DNA, RNA 및 단백질들의 서열(sequence)들은 유전에 관한 중요한 정보를 가지고 있다. 서열은 DNA의 경우 {a, t, g, c}, RNA인 경우 {a, u, g, c}, 단백질의 경우 {W, Y, F, V, L, I, M, K, R, H, Q, E, D, N, G, A, P, T, S, C}의 알파벳으로 이루어진 유한한 길이의 스트링으로 정의할 수 있다. 예를 들면 스트링 "gcctaccgagcc"는 DNA의 서열(molecular sequence)이라고 할 수 있다. 서열 순서확인(sequencing)이란 유전 물질로부터 서열의 글자를 읽어내는 작업이라고 정의할 수 있다.

· 본 연구는 정보통신부 정보통신선도기반기술개발사업의 지원에 의하여 이루어진 것임
† 정 회 원 : 한림대학교 정보통신공학부 교수
jinkim@hallym.ac.kr
†† 비 회 원 : 한림대학교 컴퓨터공학과
wcjung@center.cie.hallym.ac.kr
††† 비 회 원 : 한림대학교 교양교육부 교수
suhmn@ekus.ce.hallym.ac.kr
논문접수 : 2003년 1월 10일
심사완료 : 2003년 5월 23일

서열 순서확인에 의해 얻어진 서열들을 서로 비교하는 방법을 서열정렬(sequence alignment)이라고 하는데, 서열정렬 방법은 서열들 간의 유사도를 계산해주는 유용한 도구이다. 두 개의 서열을 서로 비교하는 방법을 쌍정렬(pairwise alignment)이라고 하며, 세 개 이상의 서열을 서로 비교하는 방법을 복수서열정렬(multiple sequence alignment)이라고 한다. 두 개의 서열을 정렬하는 최적의 방법은 Needleman과 Wunch에 의해 소개된 dynamic programming 기법에 의해 해결되었으며 [1], 이후 관심은 여러 서열을 정렬하는 복수서열정렬 문제를 효율적으로 해결하는 알고리즘을 개발하는 방향으로 이동하게 되었다. 복수서열정렬 문제는 계산 생물학(computational biology)내에서도 가장 중요한 문제 중 하나이다. 복수서열정렬은 서열 원소들 사이의 대응관계를 알아내는 것과 관련이 있다. 복수서열정렬은 생물학적 데이타 분석에 있어서 중요한 작업으로, 미지의 서열을 확인하기 위한 데이타베이스 검색, 유사한 분자구조와 관련된 패턴인식, 서열의 기능 및 기능과 진화에 관한 중요한 정보를 획득하기 위하여 사용된다. 이러한

서열정렬을 위해서는 정렬의 양호성(goodness)을 측정할 수 있는 비용함수(f)가 필요하다. 이 비용함수는 서열의 생물학적 특징을 잘 표현할 수 있어야 하며, 이 값이 최소인 서열정렬이 입력으로 주어진 서열 집합에 대한 최적 정렬(optimal alignment)을 의미하여야 한다. 서열을 정렬하는 과정에서 낮은 비용을 가진 정렬을 얻기 위해 서열의 각 원소들 사이에 매치(match)와 삽입(insertion), 교체(substitution), 삭제(deletion) 등이 이용된다.

복수서열정렬을 해결하기 위해 제안된 알고리즘들의 시간 복잡도는 $O(Nn^2)$~ $O(n^N)$(N은 서열의 개수, n은 서열의 길이) 사이에 분포하고 있으며, 거의 모든 알고리즘들이 실용적인 시간 내에 우수한 정렬을 얻는 것을 목표로 하고 있다. 복수서열정렬에 대한 보다 자세한 논의들은 [2-10]에서 잘 이루어지고 있다.

이러한 알고리즘들은 대체로 세 개의 그룹으로 나뉠 수 있다. 첫 번째 방법은 정렬하고자하는 서열들 사이의 유사도를 측정하고, 측정된 유사도를 바탕으로 정렬하는 순서를 결정하고 그 순서대로 서열들을 합병하는 방법이다[11]. 이 방법의 대부분은 [11]의 논문이 변형된 형태로 이 방법의 시간 복잡도는 $O(Nn^2)$이다. 이 방법은 많은 서열들을 짧은 시간 내에 정렬할 수 있는 장점이 있으나, 최적의 정렬을 제공하지 못하는 단점이 있다. 두 번째의 방법은 n-차원의 표를 사용하여 n개의 서열을 동시에 비교하는 dynamic programming기법이다[12,13]. 이 방법의 시간복잡도는 $O(n^N)$으로, 이 방법은 주어진 서열들을 사용하여 만들 수 있는 모든 경우의 서열정렬들을 조사하여 이중 최적의 비용을 가지는 서열 정렬을 제공하는 것이다. 이 방법은 첫 번째 방법과는 반대로 최적의(최저, 혹은 최고)의(optimal) 비용을 가지는 정렬을 제공하지만[1,9] 많은 서열을 정렬할 수 없다는 단점이 있다. 또한 dynamic programming기법을 사용할 때, 특정 비용함수를 적용시킬 수 없어 최적의 정렬을 제공하지 못하는 단점도 가지고 있다[14]. 세 번째 방법은 추정통계적인 방법으로 대표적으로 simulated annealing을 사용한 방법이 있다[15,16]. 이 방법은 주어진 서열들을 이용하여 여러 개의 서열정렬을 만들고 이중 최적에 가까운(near-optimal) 서열 정렬을 찾아내는 것으로 몇 개의 서열정렬을 만드는 가에 따라 시간복잡도가 결정되는데, 일반적으로 시간복잡도는 $O(\delta * l)$(δ는 하나의 서열정렬을 만드는 시간, l은 만들어지는 서열 정렬의 회수)이다. 이때 l을 크게 할수록 최적에 가까운 서열정렬을 얻을 수 있다. 추정통계적인 방법의 장점으로는 dynamic programming기법보다는 많은 서열을 정렬할 수 있으나, 최적의 정렬을 보장하지 못한다는 단점이 있다.

이 논문에서는 dynamic programming기법의 단점인 특정 비용함수를 사용하지 못하는 문제점을 해결하기 위한 방법을 제시하였다. 2장에서는 복수서열정렬 문제를 보다 공식적으로 정의하고, 복수서열정렬에 사용되는 비용함수를 소개하였으며, dynamic programming기법의 문제점을 기술하였다. 3장에서는 이러한 문제점을 해결하기 위한 알고리즘을 기술하였으며, 4장에서는 새로운 알고리즘에 의해 얻어진 실험 결과들을 보였고 5장에서 결론을 내렸다.

2. 복수서열정렬

2.1 서열정렬을 위한 정의

복수서열정렬 문제를 보다 공식적으로 논의하기 위하여 다음과 같은 정의를 사용하기로 한다.

• 알파벳(alphabet) Σ은 문자들과 널(null,'-')로 이루어진 유한 집합이다($\Sigma = \Sigma \cup '-'$). 여기서 문자집합으로 DNA에서는 Σ={a, t, g, c}를, RNA인 경우 Σ={a, u, g, c}를 이용하며, 단백질의 경우 Σ={W, Y, F, V, L, I, M, K, R, H, Q, E, D, N, G, A, P, T, S, C}을 이용한다. 널의 생물학적인 의미는 서열의 해당위치에서 삽입이 발생했거나, 혹은 비교된 다른 서열의 해당위치에서 삭제가 발생했음을 뜻한다.

• 서열(sequence)은 Σ로 이루어진 유한한 길이의 스트링이다.

• 갭(gap)은 유한한 길이의 연속된 널이다.

• 입력서열 $S_1, S_2, \cdots, S_k$는 각각 길이가 $n_1, n_2, \cdots, n_k$인 k개의 서열들이며($S_1 = s_{11}s_{12}\cdots s_{1n_1}$, $S_2 = s_{21}s_{22}\cdots s_{2n_2}$, $\cdots$, $S_n = s_{k1}s_{k2}\cdots s_{kn_k}$), 알파벳은 Σ이고, $s_{ij} \in \Sigma$는 i번째 서열의 j번째 원소를 나타낸다($1 \le i \le k, 1 \le j \le n_i$). 서열정렬은 주어진 입력서열에서부터 Σ'으로 이루어진 동일한 길이 l의 의사 서열 $S'_1 = s'_{11}s'_{12}\cdots s'_{1l}$, $S'_2 = s'_{21}s'_{22}\cdots s'_{2l}$, $\cdots$, $S'_k = s'_{k1}s'_{k2}\cdots s'_{kl}$을 얻는 것이다. 이 때 널이 포함된 의사 서열 S'_i로부터 널들을 제거하면 원래의 서열 S_i를 얻을 수 있다. 서열정렬은 (1)처럼 같은 크기 $k \cdot l$인 2차원 배열로 표시한다.

$$\begin{array}{l} S'_1 = s'_{11}s'_{12}\cdots s'_{1l} \\ S'_2 = s'_{21}s'_{22}\cdots s'_{2l} \\ \quad\vdots \\ S'_k = s'_{k1}s'_{k2}\cdots s'_{kl} \end{array} \quad (1)$$

이때 동일한 입력서열의 집합으로부터 유한히 많은 서로 다른 서열정렬을 얻을 수 있으며, 그 개수는 [16]에 자세히 설명되어 있다.

[정의] 복수서열정렬 문제는 정렬에 대한 최적도의 기준이 주어졌을 때, 최적 비용(optimal cost)를 가지는 복수서열정렬을 찾는 문제이다.

이 문제는 NP-complete문제군에 속하는 것으로 알려져 있으므로 최적의 해를 실용적인 시간내에 얻는 것은 불가능하다고 추론된다.

2.2 Dynamic Programming 기법을 사용한 MSA 프로그램

복수서열정렬을 얻기 위한 최적화 알고리즘 가운데 가장 표준적인 알고리즘은 Needleman과 Wunsch에 의해 소개된 dynamic programming기법이다[1]. Dynamic programming 기법은 문제를 크기가 작은 부분 문제로 분할한 뒤 부분 문제에 대한 답을 구하기 위해 같은 부분 문제를 몇 번이고 반복해서 해결할 필요가 있는 경우에 사용한다. 이때 한번 해결한 부분 문제의 해답을 표에 기억해 두었다가 필요한 때에 그 표를 참조하게 되면 새로 계산하는데 걸리는 시간을 절약할 수가 있어 효율적이다. 본질적으로 dynamic programming은 모든 부분 문제에 대한 답을 계산하는데, 계산은 크기가 작은 부분 문제로부터 보다 큰 부분 문제로 답을 표에 기록하고, 모든 기록을 마치면 최적의 해를 얻게 된다.

[17]에서 제안된 MSA 프로그램은 dynamic programming기법을 사용하여 복수개의 서열을 정렬하는 C로 구현된 프로그램이다. Dynamic programming기법을 이용하여 n개의 서열을 정렬하려면 n-차원의 표를 이용하여야 하는데, MSA 프로그램은 n-차원의 표의 일부만을 계산하여 최적 혹은 최적에 가까운 비용을 가진 복수서열정렬을 찾아낸다. MSA 프로그램은 정렬하려는 서열의 개수가 증가함에 따라 필요한 실행시간도 지수함수적으로 증가($O(2l^n)$, l : 서열의 평균길이, n : 서열의 개수)하기 때문에 시간복잡도를 줄이기 위해 몇 가지 단축기법을 사용한다. 먼저 n-차원의 표를 모두 계산하는 것이 아니라, 빠른 경험적인 서열정렬 알고리즘을 사용하여 n-차원의 표에서 모든 서열 쌍(sequence pair)들을 비교하여 대강의 solution path를 계산하고, solution path주위만의 셀들에 대한 값들을 계산하여 solution space를 줄이는 방법을 사용하고 있다[14]. 또한 natural gap cost라는 비용함수 대신 quasi-natural gap cost라는 비용함수를 사용하여 계산시간을 줄이도록 하고 있다[14].

2.3 비용함수

서열정렬에 사용되는 비용함수는 정렬의 질(quality)에 대한 명백한 측정수단이어야 한다. 즉, 작은 비용을 가지는 복수서열정렬은 큰 비용을 가지는 복수서열정렬보다 생물학적 현상을 잘 표현해야 한다. Altschul[14]은 복수서열정렬을 위한 몇 가지 비용함수를 분류하였는데, 이들 비용함수는 교체비용(substitution cost)과 갭비용(gap cost)으로 구성된다. 따라서 특정복수서열정렬 A의 비용은 다음의 식 (2)처럼 교체비용과 갭비용을 계산하여 더하면 된다.

$$\text{비용}(A) = \text{교체비용}(A) + \text{갭의 개수}(A) \cdot \textit{gap penalty} \quad (2)$$

주어진 복수서열에 대해 가능한 모든 복수서열정렬 중에서 이 비용이 가장 작은 정렬이 최적의 정렬이 된다.

2.3.1 교체비용

교체비용은 DNA나 단백질 서열 내에서 분자를 대표하는 문자들을 정렬하는데 드는 비용이다. 이 논문에서 사용된 교체비용은 SP(Sum of Pairs) 교체비용이다. SP 교체비용은 n개의 서열이 정렬되었을 때, $n(n-1)/2$ 개의 쌍에 대한 교체비용의 합이다. 따라서 복수서열정렬 A의 교체비용은 다음의 식 (3)과 같이 계산될 수 있다.

$$\text{교체비용}(A) = \sum_{i=1}^{n-1} \sum_{j=i}^{n} \text{교체비용}(S'_i, S'_j) \cdot \textit{weight}(i,j) \quad (3)$$

이때 단백질 서열간의 교체비용으로 주로 사용되는 비용 매트릭스는 Dayhoff에 의해 제안된 matrix가 있으며[18], $\textit{weight}(i,j)$는 서열 S_i와 S_j를 비교하여 얻은 교체비용에 대한 가중치이다.

2.3.2 갭비용

갭비용은 연속된 널('-')의 개수에 부과되는 비용이다. 따라서 '-', '--', '---' 등은 모두 하나의 갭으로 간주된다. 또한 복수서열정렬내의 두 의사서열 S'_i과 S'_j에서 공통된 위치에 널이 존재한다면 그 널은 서로 상쇄될 수 있다. Altschul은 생물학적으로 자연스러운 갭비용함수로써 natural gap cost를 제안하였다[14]. 이는 정렬된 서열내의 모든 서열 쌍들을 비교하여 갭의 개수를 계산하고, 갭의 개수에 비례하는 벌점을 부여하는 방법을 말하는데, 이는 갭이 많을수록 비교하는 서열들은 유사하지 않다고 여기기 때문이다.

의사서열 $S'_a = a_n a_{n-1} \cdots a_1$과 의사서열 $S'_b = b_n b_{n-1} \cdots b_1$에 대해 갭비용을 계산하고자 할 때 natural gap cost를 이용하기 위해서는 서열 S'_a와 S'_b의 상응하는 위치에서 연속해서 2개의 서열원소 $\begin{pmatrix} a_1 a_2 \\ b_1 b_2 \end{pmatrix}$를 읽어내야 한다. 이때 a_1과 b_1은 갭이 연속된 null 스트링인지 여부를 판단하기 위해 필요한 것으로 실제 계산되어지는 것은 a_2와 b_2이다.

MSA 프로그램에서 비용계산으로 natural gap cost를 사용하기 위해서는 전상태의 정렬기록(alignment history)을 가지고 있어야 한다. n개의 서열을 SP 교체비용과 natural gap cost를 사용하여 정렬할 경우, MSA 프로그램이 보유하고 있어야 하는 정렬기록(history)의 양, $H_A(n)$은 다음의 순환식 (4)로 표현될 수 있다[14].

$$H_A(n) = \begin{cases} 1 & \text{if } n=0 \\ \sum_{i=0}^{n-1} \binom{n}{i} A_i & \text{if } n \geq 1 \end{cases} \quad (4)$$

위의 수식을 이용하여 10개의 서열을 natural gap cost를 적용하여 정렬하는 경우 보유하여야 하는 정렬 기록의 개수를 계산하면 $H_A(10)$=102,247,563이 된다 [14]. 이는 비실용적인 기록 양으로, Altschul은 이 문제를 해결하기 위하여 natural gap cost와 유사한 quasi-natural gap cost를 제안하였다[14]. Quasi-natural gap cost는 표 1에서 볼 수 있는 것처럼 하나의 서열에 존재하는 갭이 다른 서열에 존재하는 갭에 완전히 포함(completely nested)되는 경우, 실제보다 갭을 하나 더 센다(miscalculate)는 잘못된 점을 제외하고는 natural gap cost와 동일하다. quasi-natural gap cost를 사용하는 경우 저장이 필요한 정렬 기록의 양은 2^n-1이다. 이 수식에 따르면 10개의 서열을 quasi-natural gap cost를 적용하여 정렬하려면 1023개의 정렬 기록을 보유하면 되고 이는 위에서 제시한 natural gap cost를 적용하는 경우의 102,247,563보다 훨씬 실용적이다.

MSA 프로그램은 시간과 공간복잡도를 줄이기 위해서 quasi-natural gap cost를 갭비용함수로 사용하였다. 우리는 MSA 프로그램에 의해 제공되는 정렬이 가진 비용을 $Opt_{quasi-natural}$로 정의한다. 또한 natural-gap cost를 적용하여 얻은 최적 정렬의 비용을 $Opt_{natural}$로 정의한다. 최적의 정렬이 완전히 포함되는 갭(completely nested gap)을 가지는 경우 앞에서 지적한 것처럼 quasi-natural gap cost를 이용하는 MSA 프로그램은 정확한 비용을 계산할 수 없게 되고, 그 결과 최적이 아닌 정렬을 최적의 결과로 잘못 판단하게 된다. 그림 1은 널('-')과 널이 아닌 알파벳('X')으로 구성된 서열정렬의 일부분으로, (a)는 아래 서열의 갭이 위쪽 서열의 갭에 완전히 포함(completely nested)되는 경우를 보여주고 있다. (a)의 seq2의 길이가 1인 갭은 (a)의 seq1의 길이가 3인 갭에 완전히 포함되어 있는 형태이다. 이 경우 MSA 프로그램은 seq1과 seq2를 비교할 때 그림 1의 (a)처럼 갭의 개수를 1개가 아닌 2개로 잘못 계산하게 된다.

그림 1의 (b)가 올바르게 비용이 계산되는 정렬로, (a)의 seq1과 seq2의 3번째 열의 널('-')부분은 제거되어 실제 두 개의 연속된 널을 가지는 하나의 갭으로 계산하게 된다.

표 1은 natural gap cost와 quasi-natural gap cost를 사용했을 때 계산한 갭의 개수를 보여주고 있다.

```
seq1   X---X      X--X
seq2   XX-XX      XXXX
         (a)        (b)
```

그림 1 (a) 정렬 (b) 비용이 올바르게 계산되는 정렬

표 1 natural gap cost와 quasi-natural gap cost의 차이

서열정렬 / gap cost	seq1 X---X seq2 XXX-X seq3 XXXXX	seq1 X---X seq2 XX-XX seq3 XXXXX
natural gap cost를 적용할 때 갭의 개수	3	3
quasi-natural gap cost를 적용할 때 갭의 개수	3	4

표 1의 복수서열정렬의 갭의 개수는 (seq1, seq2), (seq1, seq3), (seq2, seq3)를 비교하여 각각의 갭의 개수를 계산하고, 얻어진 갭의 개수들을 더하면 정렬내의 총 갭의 개수를 얻게된다. 이때 서열정렬 내에 하나의 긴 갭에 완전히 포함되는(completely nested) 갭들이 존재하지 않는 경우(표 1의 첫 번째 서열정렬) $Opt_{quasi-natural}$과 $Opt_{natural}$은 같은 결과를 제공하지만, 완전히 포함되는 갭들이 존재할 경우(표 1의 두 번째 서열정렬의 seq1과 seq2사이의 갭개수 계산 시) 갭의 개수를 하나 더 계산하기 때문에 MSA 프로그램에서 제공하는 $Opt_{quasi-natural}$은 $Opt_{natural}$보다 추가된 갭의 개수에 갭 패널티를 곱한 값만큼의 비용이 추가된다 ($Opt_{quasi-natural} = Opt_{natural} + n \cdot g$, n은 잘못 추가된 갭의 개수 g는 갭 패널티). 따라서 실제로는 두 번째 형태의 서열정렬이 가장 작은 비용을 가지고 있지만, MSA 프로그램은 첫 번째 서열정렬을 가장 작은 비용을 가진 정렬로 제공하는 경우가 발생한다.

3. Dynamic programming의 서열정렬을 개선(refine)하는 알고리즘

Quasi-natural gap cost를 이용하는 MSA 프로그램을 사용하여 복수서열정렬을 얻는다면 $Opt_{quasi-natural}$을 가지는 서열정렬을 얻을 수 있다. 그러나 우리의 최종목표는 $Opt_{natural}$을 가지는 서열정렬을 얻는 것이다. 이 장에서는 MSA 프로그램이 제공하는 $Opt_{quasi-natural}$을 가지는 서열정렬 중 완전히 포함되는 형태의 갭을 가질 수 있는 부분서열(subsequence)만을 선택하고, 그 부분서열에 대하여 natural gap cost 비용함수를 적용하여 $Opt_{natural}$을 가지는 서열 정렬을 만들어내는 Sub-alignment Refinement Algorithm(SRA)방법에 대하여 설명한다.

3.1 Realignment

MSA 프로그램에 의해 제공되는 서열정렬은 최적의 서열정렬과 매우 유사하다. 위의 표 1에서 두 개의 서열정렬은 널이 위치한 부분만을 제외한 전체 서열의 나머지 부분은 같다는 것을 쉽게 알 수 있다. 따라서 MSA 프로그램이 만들어 낸 서열정렬에서 있어서 널이 위치

한 부분서열을 최적 정렬로 만들 수 있다면 쉽게 전체 서열을 최적 혹은 보다 최적에 가깝게 정렬할 수 있다. 이 때 개선(refinement)을 시도할 부분서열은 다음 조건 (1)과 (2)를 만족하여야 한다.

(1) 갭이 존재하는 서열의 수가 2이상이어야 한다.

(2) 가장 많은 널의 개수와 가장 적은 널의 개수의 차가 2이상이어야 한다.

조건 (1)과 (2)는 갭들이 완전히 포함되기 위한 최소한의 조건이다. 위의 조건을 만족한다면 부분서열내의 널들을 재배치하여 완전히 포함된 널을 갖는 부분서열들을 생성하고 정렬하여 더 작은 비용을 가지는 부분서열정렬이 나타날 경우 기존의 부분서열들을 새로이 생성된 부분서열들로 대치하면 보다 적은 비용을 갖는 복수서열정렬을 얻게 된다. 이때 비용 계산에 있어서 MSA 프로그램과 다른 점은 비교하는 두 서열의 갭이 $\binom{x-}{--}$의 형태로 시작하고 $\binom{-x}{--}$의 형태로 끝나면(혹은 $\binom{--}{x-}$로 시작하여 $\binom{--}{-x}$로 끝날 경우), 이를 하나의 갭으로 계산한다는 것이다.

개선하고자 하는 부분서열은 원래의 서열과 비교하여 매우 작은 길이를 갖는다. 이때 최적의 부분서열정렬을 찾는 방법은, 기존의 부분서열정렬이 k개의 갭을 가졌다고 할 때, 기존의 부분서열 집합으로부터 서로 다른 위치에 k개 이하의 갭을 가진 가능한 모든 부분서열정렬을 만들고, 각각의 서열정렬에 대하여 비용을 계산한다. 이들 부분서열정렬 중에서 최소 비용을 갖는 부분서열정렬이 최적의 부분서열정렬이 된다. 이 논문에서는 부분서열정렬의 각 서열에 많아야 하나의 갭을 가지는 부분 서열을 만들고 비용을 계산하였다. 이때 만들어지는 가능한 서로 다른 부분서열 집합의 개수는 다음과 같다. 부분서열의 개수를 n, 길이를 l, i번째 서열에 포함되는 갭의 길이를 G_i라 하면 만들어질 수 있는 가능한 부분서열의 개수 K는 다음 식 (5)와 같다.

$$K = \Pi_{i=1}^{n} (l - G_i + 1) \quad (5)$$

그림 2는 MSA 프로그램에 의해 만들어진 부분서열로부터 가능한 부분서열을 얻는 방법을 나타낸 것이다. 서열들에 포함된 널의 개수에 의해 널의 시작점과 널의 끝점을 알 수 있으므로, 이를 이용하여 서로 다른 부분정렬을 만들어 낼 수 있다. 그림 2(a)는 MSA 프로그램에 의한 정렬이다. 2(a)로부터 각 서열 내에 포함된 널의 개수를 계산한다. 그림 2(b)와 (c)는 다양한 널 위치를 갖는 서로 다른 부분정렬 중에서 최초에 만들어지는 부분정렬(b)이며 마지막으로 만들어지는 정렬(c)이다.

새로이 만들어낸 부분서열정렬에 대하여 natural gap cost를 적용하여 비용을 계산한다. 이때 기존의

MSA 프로그램에 의한 부분서열정렬	널의 개수
XXXX--XXX	2
XXXXX-XXX	1
XX----XXX	4
------XXX	6

(a)

최초에 만들어지는 부분서열정렬	널의 시작점
--XXXXXXX	0
-XXXXXXXX	0
----XXXXX	0
------XXX	0

(b)

마지막으로 만들어지는 부분서열정렬	널의 시작점
XXXXXXX--	7
XXXXXXXX-	8
XXXXX----	5
XXX------	3

(c)

그림 2 (a) MSA 프로그램에 의한 서열정렬 (b) 최초의 부분서열정렬 (c) 마지막 부분서열정렬

$Opt_{quasi-natural}$보다 작은 값을 가지는 부분서열정렬이 발견되면 기존의 부분서열을 새로이 발견된 부분서열로 교체한다. 기존의 부분서열에 완전히 포함된 갭의 개수와 새로이 발견된 부분서열에 완전히 포함된 갭의 개수의 차이가 k_1이라하고, 하나의 갭에 대한 패널티가 g라 하면 교체함에 의해 얻게 되는 $Opt_{natural}$값은 다음의 보조정리 1과 같은 조건을 만족한다.

보조정리 1 :

$$Opt_{quasi-natural} - k_1 \cdot g \le Opt_{natural} \le Opt_{quasi-natural}$$

증명 : 갭의 개수가 k_1개이므로 줄일 수 있는 비용은 최대 $k_1 \cdot g$가 된다. 따라서 새로운 부분서열정렬의 값은 $Opt_{quasi-natural} - k_1 \cdot g$가 된다. □

3.2 Recalculation

만일 MSA 프로그램에 제공되는 정렬에 완전히 포함된 갭들이 k_2개만큼 포함되어 있다면, 해당되는 갭의 개수만큼의 갭비용이 더 부가되어 있는 셈이 된다. 따라서 이러한 경우 natural gap cost를 적용하기 위해서는 해당되는 갭 개수만큼의 패널티를 감하여야 한다. 이때 새로이 얻게되는 비용은 보조정리2와 같다.

보조정리 2 :

$Opt_{natural} = Opt_{quasi-natural} - k_2 \cdot g$

증명 : 완전히 포함되는 갭의 개수가 k_2개이므로 $k_2 \cdot g$만큼 더 부가된 비용을 제거하면 $Opt_{natural}$을 얻을 수 있다. □

위의 보조정리 1과 2를 사용하여 다음과 같은 정리를 얻을 수 있다.

정리 :

$Opt_{quasi-natural} - k_1 \cdot g - k_2 \cdot g \leq Opt_{natural} \leq Opt_{quasi-natural}$

증명 : 보조정리1과 보조정리2에 의해 자명하다. □

위의 정리는 제안한 방법으로 얻을 수 있는 $Opt_{natural}$의 하한 값을 나타낸다.

3.3 알고리즘의 분석

이 논문에서 제안된 알고리즘을 요약하면 그림 3과 같다. 이 부분정렬위의 알고리즘의 시간 복잡도는 [과정 2]에 의존하게 된다. [과정 2]에서 발생하는 하나의 부분서열의 비용을 계산하는데 $O(\ln^2)$이 필요하므로, K개의 부분서열정렬에 대해서는 $O(Kln^2)$의 시간이 필요하다. 이때 [과정 2]에서는 서열을 저장하기 위한 공간이외에 필요한 공간이 없으므로 그 공간복잡도는 $O(\ln)$이다.

[과정 1] MSA에 의해 획득된 서열정렬에서 개선될 수 있는 부분서열을 찾는다. 이때 부분서열은 3.1의 두 조건을 만족해야 한다.
[과정 2] 과정2에서 얻은 부분서열에 대하여 realignment, recalculation 방법을 적용한다.
[과정 3] 더 낮은 비용을 가지는 부분서열정렬을 얻게 되면 기존의 부분서열과 교체한다.

그림 3 알고리즘 요약

4. 실험결과

이 논문에서 제안한 알고리즘은 C로 구현되었으며, linux상에서 컴파일하고 실행시켰다. 메인 메모리의 용량은 256MB였다. 제안한 알고리즘은 많은 메모리 용량을 필요로 하지 않는다. 이 알고리즘을 테스트하기위해 chymotrypsin, trypsin과 elastase family들로부터 단백질 서열들을 선택하여 4개의 서열들로 복수서열정렬을 만들고 개선시킬 수 있는 부분서열을 개선해보았다. 이때 실험에 사용된 교체비용 계산을 위한 matrix는 PAM250 matrix, 갭 페널티는 8이었다. MSA 프로그램과 본 논문의 알고리즘에서는 교체비용의 가중치 $weight(i,j)=1$로 설정하여 비용을 계산하였는데, 이는 MSA와 본 알고리즘의 비교를 용이하게 하기 위함이다.

그림 4(a)는 4개의 서열들을 MSA 프로그램을 사용하여 만들어낸 정렬이며 4(b)는 이 연구에서 제안한 알고리즘을 사용하여 개선한 정렬이다.

MSA 프로그램에 의해 제공된 정렬의 비용은 3473이었고, 개선된 정렬은 3472로 1이 작았으며, (a)에서 없었던 완전히 포함된 갭이 (b)에 하나 존재하는 것을 볼 수 있다.

그림 5(a)는 4개의 서열들을 MSA 프로그램을 사용하여 만들어낸 서열정렬의 부분서열이며 5(b)는 (a)의 부분서열을 이 논문에서 제안한 알고리즘을 사용하여 개선한 부분서열정렬이다. MSA 프로그램에서 만들어낸 서열정렬의 비용은 1651이었으며, 개선된 서열정렬의 비용은 1645였다. 그림 5에서도 그림 4의 예와 같이 5(a)에서는 없었던 완전히 포함된 갭이 5(b)에는 2개가 나타났음을 볼 수 있다. 그림 4의 예에서와 마찬가지로 이 새로이 나타난 2개의 완전히 포함된 갭이 비용을 줄이는 요인임을 알 수 있다.

그림 6은 recalculation이 필요한 예를 보여주고 있다. 그림 6은 MSA 프로그램이 만들어낸 서열정렬의 부분서열로 이 정렬에는 완전히 포함된 갭이 존재함을 볼 수 있다. 이 서열정렬에 대해 MSA 프로그램에서 계산

```
CGPCEPA---SCPPLPPLGCLLGETRDACGCCPMCARGEG
CAPCSAERMALCPPV-PASCPELTRSAGCGCCPMCALPLG
CPGCGQGVQAGCPGGCVEEEDGGSPAEGCAEAEGCLRREG
CPGCGPG---------VQEEDAGSPADGCAETGGCFRREG
```

(a)

```
CGPCEPAS---CPPLPPLGCLLGETRDACGCCPMCARGEG
CAPCSAERMALCPPV-PASCPELTRSAGCGCCPMCALPLG
CPGCGQGVQAGCPGGCVEEEDGGSPAEGCAEAEGCLRREG
CPGCGPG---------VQEEDAGSPADGCAETGGCFRREG
```

(b)

그림 4 (a) MSA 프로그램에 의해 생성된 정렬 (b) 부분서열정렬개선방법에 의한 정렬

```
CRSAYG--NELVANEEICAG
CKKSYP---GQITSNMFCLG
CRRRVN----------VCTL
CLDTLHQQKQETRINIMCIG
```

(a)

```
CRSAYGNE--LVANEEICAG
CKKSYPGQ---ITSNMFCLG
CRRRVN----------VCTL
CLDTLHQQKQETRINIMCIG
```

(b)

그림 5 (a) MSA 프로그램에 의해 생성된 정렬 (b) 부분서열정렬개선방법에 의한 정렬

```
YYG-------YY
YYGG-----YYY
YYGGGGXYYYY
```

그림 6 Recalculation

한 비용은 315였으나, 제안된 알고리즘은 동일한 정렬에 대하여 307로 올바른 값을 제공하였다.

개선된 프로그램을 수행하는데 걸리는 시간은 갭의 개수와, 입력서열의 길이에 비례한다. 위의 예에서 걸리는 시간은 모두 수초이내였다. 그림 4, 5, 6은 제안된 알고리즘이 효율적으로 $Opt_{natural}$을 얻을 수 있음을 보여준다.

그림 7의 (a)와 (b)는 4개의 완전한 단백질 서열들을 MSA로 정렬했을 때와, 제안된 알고리즘을 사용하였을 때의 결과를 나타낸다.

MSA를 사용하여 4개의 서열을 정렬할 때 24745의 값을 가진 정렬을 얻었다. 이 서열의 갭이 있는 지역들을 분리하여 본 알고리즘을 적용하여 개선된 부분을 얻을 수 있었다. 이때 그림 7(a)에서 선택되어 (b)에서 개선된 지역을 '*'로 표시하였다. 개선된 지역들을 원래의 정렬에서 교체한 결과 그림 7(b)를 얻을 수 있었으며 이때 총 비용은 24742였다.

이 논문에서 제안한 알고리즘은 개선되는 부분서열집합의 서열들이 하나의 갭을 가지는 경우에만 적용할 수 있다. 만일 부분정렬내의 서열들이 하나 이상의 갭을 가질 경우, 계산해야 하는 다른 형태의 부분서열이 많아지

```
                                      ************************************
M-ERASLRA--LLFGPAGLLLLLLPLSSSSSSD-TCGPCEPAS---CPPLPPLGCLLGETRDACGCCPMCARGEG
MPEVLAVRAWPLLL-----SLAVQLGATVGAPQPWRCAPCSAERMALCPPV-PASCPELTRSAGCGCCPMCALPLG
M-TPHRLLP-PLLL-----LLALLLAASPGGALA-RCPGCGQGVQAGCPGGCVEEEDGGSPAEGCAEAEGCLRREG
M-TWDGLPTQPLLM-----LLMLLFAAGSESALA-GCPGCGPG----------VQEEDAGSPADGCAETGGCFRREG

EPCGGGGAGRGYCAPGMECVKSRKRRRGKAGAAAGGPGVSGVCVCKSRYP-VCGSDGTTYPSGCQLRAASQRAES
AACGVATAR---CARGLSC-----RALPGEPRPLHALTRGQGACM-TSPCDEATDTKDTTSPENVSPESSEITQEQ
QECGVYTPN---CAPGLQC-----HPPKDDEAPLRALLLGRGRCL-PARAP-----AVAEENPKESKPQAGTARPQD
QPCGVYIPK---CAPGLQC-----QPRENEETPLRALLIGQGRCQ-RARGP------SEETTKESKPHGGASRPRD

                  ********************
RGEKAITQVSKGTCEQGPSIVTPPKDIWNVTGAQVY------LSCEVIGIPTPVLIWNKVKRGHYGVQRTELLPG
LLDN--FHLMAESSEDLPILWNAISNYESLRALEISDVKKWKEPCQRE--LYKVL--DRLAREQQKAGDK-LY--
VNRR--DQQRNPGTSTTPSQPNSAG----VQDTEMG-------PCRRH--LDSVL--QQLQTEVYRGAQT-LY--
RDRQ-----KNPRTSAAPIRPSP------VQDGEMG-------PCRRH--LDSVL--QQLQTEVFRGGANGLY--

DRDNLAIQTRGGPEKHEVTGWVLVSPLSKEDAGEYECHASNS-QGQASASAKITVVDALHEIA-SEKR
KFYLPNCNKNGFYHSKQCE-----TSLEGEPG---LCWCVYPWSGKRILGSVAVRGDPKCQQYFNLQN
---VPNCDHRGFYRKRQCR-----SSQGQRRG---PCWCVDR-MGKSLPGSPDGNGSSSCPT--GSSG
---VPNCDLRGFYRKQQCR-----SSQGNRRG---PCWCVDP-MGQPLPVSPDGQGSSQCSA--RSSG
```

(a)

```
                                      ************************************
M-ERASLRA--LLFGPAGLLLLLLPLSSSSSSD-TCGPCEPA---SCPPLPPLGCLLGETRDACGCCPMCARGEG
MPEVLAVRAWPLLL-----SLAVQLGATVGAPQPWRCAPCSAERMALCPPV-PASCPELTRSAGCGCCPMCALPLG
M-TPHRLLP-PLLL-----LLALLLAASPGGALA-RCPGCGQGVQAGCPGGCVEEEDGGSPAEGCAEAEGCLRREG
M-TWDGLPTQPLLM-----LLMLLFAAGSESALA-GCPGCGPG----------VQEEDAGSPADGCAETGGCFRREG

EPCGGGGAGRGYCAPGMECVKSRKRRRGKAGAAAGGPGVSGVCVCKSRYP-VCGSDGTTYPSGCQLRAASQRAES
AACGVATAR---CARGLSC-----RALPGEPRPLHALTRGQGACM-TSPCDEATDTKDTTSPENVSPESSEITQEQ
QECGVYTPN---CAPGLQC-----HPPKDDEAPLRALLLGRGRCL-PARAP-----AVAEENPKESKPQAGTARPQD
QPCGVYIPK---CAPGLQC-----QPRENEETPLRALLIGQGRCQ-RARGP------SEETTKESKPHGGASRPRD

                  ********************
RGEKAITQVSKGTCEQGPSIVTPPKDIWNVTGAQVY-------LSCEVIGIPTPVLIWNKVKRGHYGVQRTELLPG
LLDN--FHLMAESSEDLPILWNAISNYESLRALEISDVKKWKEPCQRE--LYKVL--DRLAREQQKAGDK-LY--
VNRR--DQQRNPGTSTTPSQPNSA---GVQDTEMG-------PCRRH--LDSVL--QQLQTEVYRGAQT-LY--
RDRQ-----KNPRTSAAPIRPSP-------VQDGEMG-------PCRRH--LDSVL--QQLQTEVFRGGANGLY--

DRDNLAIQTRGGPEKHEVTGWVLVSPLSKEDAGEYECHASNS-QGQASASAKITVVDALHEIA-SEKR
KFYLPNCNKNGFYHSKQCE------TSLEGEPG---LCWCVYPWSGKRILGSVAVRGDPKCQQYFNLQN
---VPNCDHRGFYRKRQCR------SSQGQRRG---PCWCVDR-MGKSLPGSPDGNGSSSCPT--GSSG
---VPNCDLRGFYRKQQCR------SSQGNRRG---PCWCVDP-MGQPLPVSPDGQGSSQCSA--RSSG
```

(b)

그림 7 (a) MSA에 의해 만들어지는 서열정렬. 서열의 총비용은 24745.
(b) 부분서열정렬을 이용하여 MSA의 서열을 개선한 서열정렬.
서열의 총비용은 24742.

게 되는데, 이러한 경우는 갭의 개수가 증가하여 일반적으로 부분서열의 비용이 커지게 되기 때문에 이 논문에서는 고려하지 않았다. 이러한 경우, 모든 경우의 갭 모양을 고려해야 하므로 simulated annealing[16]과 같은 방법을 사용할 수도 있다.

5. 결론 및 향후과제

이 논문에서는 복수서열정렬의 최적화 기법에 대하여 논의하였다. Dynamic programming 알고리즘은 최적의 복수서열정렬을 제공하는 가장 널리 사용되는 알고리즘이나, 이 알고리즘의 속성상 natural gap cost를 적용할 수 없는 것이 가장 결정적인 문제점이었다. 이러한 문제점을 해결하기 위한 다양한 알고리즘이 제안되었지만, 어느 논문에서도 $Opt_{quasi-natural}$과 $Opt_{natural}$의 관계에 대한 이론적인 근거를 제시하지 못하였다. 이 논문에서 제안하는 개선 알고리즘을 사용하여 natural gap cost를 사용한 최적의 복수서열정렬을 얻을 수 있었으며, $Opt_{natural}$의 하한값에 대한 이론적인 근거를 제시하였다. 본 논문에서 제안한 알고리즘을 개선하려는 부분정렬내의 서열들이 하나 이상의 갭을 가질 경우에 대해 적용하려 할 경우, 계산해야 하는 다른 형태의 부분서열이 많아지게 된다. 이러한 경우의 갭 모양을 고려해야 하는 문제를 해결하기 위하여 향후 효율적인 추정통계적인 방법을 적용하려 한다. 현재 고려하고 있는 추정통계적인 방법은 simulated annealing과 genetic algorithm이다. 이때 전통적인 simulated annealing이나 genetic algorithm을 적용할 때 많은 계산시간이 필요하므로, 계산시간을 줄일 수 있는 기법을 개발할 예정이다.

참 고 문 헌

[1] Needleman. S. B and Wunch, C. D. "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. Molec. Biol.*, Vol. 48, pp. 443-453, 1970.

[2] Chan, S. C. C., Wong, A. K., and Chiu, D. K. Y. "A survey of multiple sequence comparison methods," *Bull. Math. Bio.*, Vol.43, pp.563-598, 1992.

[3] Feng, D. F., Johnson, M. S., and Doolittle, R. F. "Aligning amino acid sequences: comparison of commonly used methods," *J. Molec. Evol.*, Vol.21, pp. 112-125, 1982.

[4] Ficket, J. W. "Fast optimal alignment," *Nucl. Acids Res.*, Vol.12 pp.175-180, 1984.

[5] Martinez, H. M. "A flexible multiple sequence alignment program," *Nucl. Acids. Res.*, Vol16, pp.1683-1691, 1988.

[6] Notredame, C. and Higgins, D. "SAGA:sequence alignment by genetic algorithm," *Nucl. Acids. Res.*, Vol.24, No.8, pp.1515-1524, 1996.

[7] Sankoff, D. "Simultaneous comparison of three or more sequences related by a tree," *Addison-Wesley, Reading, MA*, 1983.

[8] Sankoff, D. and Kruskal, J. B. "Time Warps, String Edits and Macromolecules: The theory and practice of Sequence Comparison," *Addison-Wesley, Reading, MA*, 1983.

[9] Smith, T. F and Waterman, M. S. "Identification of common molecular subsequences.", J. Mol. Biol. Vol. 147, pp.195-197. 1981.

[10] Taylor, W. R. "Multiple sequence alignment by a pairwise algorithm," *CABIOS*, Vol.3, pp.81-87, 1987.

[11] Feng, D. F. and Doolittle, R. F. "Progressive sequence alignment as a prerequisite to correct phylogenetic trees," *J. Molec. Evol.*, 25:351-360, 1987.

[12] Altschul, S. F. and Lipman, D. J. "Threes, stars, and multiple biological sequence alignment," *SIAM J. appl. Math.*, Vol.49 pp.197-209, 1989.

[13] Murata, M., Richardson, J. S., and Sussman. J. L. "Simultaneous comparison of three protein sequences," In Proc. *Natl. Acad. Sci.* USA., Vol.82, pp.3073-3077, 1985.

[14] Altschul, S. F. "Gap costs for multiple sequence alignment," *J Theor. Biol.*, Vol.138 pp.297-309, 1989.

[15] Kim, J. and Pramanik, S. "An efficient method for multiple sequence alignment," *In Second International Conference on Intelligent Systems for Molecular Biology*, 1994.

[16] Kim, J. and Pramanik, S. and M. J. Chung. "Multiple sequence alignment using simulated annealing," *CABIOS*, Vol.10, pp.419-426, 1994.

[17] Lipman, D. J., Altschul, S. F. and Kececioglu, J. D. "A tool for multiple sequence alignment," *Proc. Natl. Acad. Sci.* USA., Vol.86, pp. 4412-4415, 1989.

[18] Dayhoff. M. O. "A model of evolutionary change in proteins. matrices for detecting distance relationships," *In Atlas of Protein sequence an Structure*, Vol. 5 suppl.3, pp.354-352. Dayhoff. M. O.(ed) Washington. DC: National Biomedical Research Foundation, 1978.

김 진

1984년 고려대학교 물리학과 이학사 1990년 Michigan State University, 공학석사. 1996년 Michigan State University, 공학박사. 1997년 3월~2000년 8월 건국대학교 자연과학대학 컴퓨터과학전공. 2000년 9월~현재 한림대학교 정보통신공학부. 관심분야는 Bioinformatics, 알고리즘, 데이타베이스, 인공지능

정 우 철

2001년 한림대학교 컴퓨터공학과(학사) 2001년~2002년 한국소프트 스페이스 개발직 근무. 2002년~현재 한림대학교 컴퓨터공학과 석사 과정. 관심분야는 패턴인식, Bioinformatics, 알고리즘, 3D visualization

엄 상 용

1987년 한림대학교 전자계산학과 졸업 1997년 한림대학교 컴퓨터공학부 석사 1999년 한림대학교 컴퓨터공학부 박사과정수료. 1999년~현재 한림대학교 교양교육부 교수