서열 정렬 알고리즘의 연구 동향

성종희[1], 김동규
부산대학교 컴퓨터공학과
{jhsung[1], dkkim}@islab.ce.pusan.ac.kr

A Survey of Sequence Alignment Algorithms

Jong Hi Sung[1], Dong Kyue Kim
Dept. of Computer Engineering
Pusan National University, Busan 609-735, Korea

요 약

서열 정렬(sequence alignment)은 새로운 서열의 기능적, 구조적, 진화적 분석을 용이하게 하기 때문에 분자 생물학(molecular biology) 등에서 널리 사용된다. 지금까지 서열 정렬 알고리즘들에 대한 연구는 활발히 진행되어 왔다. 특히, 생물학 데이터양의 기하급수적인 증가와 전체 유전체 서열의 분석이 이루어진 종(species)들이 증가하면서, 보다 빠르고 정확하게 서열 정렬을 수행하는 알고리즘이 필요하게 되었다. 본 논문에서는 동적 프로그래밍 방식에서부터 전체 유전체 서열 정렬 알고리즘에 이르기까지 서열 정렬 알고리즘의 연구 동향을 분석하고자 한다.

1. 서론

서열 정렬(sequence alignment), 혹은 서열 비교(sequence comparison)는 서열들 간의 상관관계를 나타내기 위해서 서열을 정렬하는 것이다. 서열 정렬은 두 서열간의 정렬뿐 아니라, 여러 서열간의 정렬, 그리고 서열 데이터베이스(sequence database)에 대한 검색 등을 모두 포함한 용어이다.

분자 생물학에서는 염기 서열(nucleotide sequence), 아미노산 서열(amino acid sequence)과 같은 생체 분자 서열(biomolecular sequence)을 가지고 서열 정렬을 수행하는데, 새로운 서열에 대한 기능적, 구조적, 진화적 분석을 용이하게 하므로 서열 정렬은 현대 분자 생물학에서 필수적이다.

Gusfield[1]에 따르면, 생체 분자 서열에 대한 분석을 통해서 밝혀진 첫 번째 사실은 유사성(similarity)이 높은 서열일수록 기능적으로나 구조적으로도 유사할 가능성이 높다는 것이다. 한 개체가 생성될 때 이전 세대로부터 정보를 복제하는데, 그 과정에서 돌연변이가 발생할 수 있고 기존의 정보는 수정된다. 이 정보가 다시 다음 세대로 전달되는 과정이 반복되면서 진화가 이루어진다. 즉 이전 세대의 DNA를 복제하기 때문에, 새로 생성된 DNA 서열은 이전 세대의 DNA 서열과 유사하다. 따라서 서열들이 서로 유사성을 가진다면 그 서열들은 어떤 진화적 관계를 가질 가능성이 높아진다. 이와 같은 이유로 서열 정렬은 새로운 서열에 대한 기능적, 구조적, 진화적 분석의 시작이 된다.

분석된 유전자의 수가 늘어나고, 전체 유전체 서열(whole genome sequence)이 분석된 생물의 수도 해마다 빠르게 증가함으로써, 생물학 데이터양은 기하급수적으로 증가했다. 따라서 전체 유전체 서열들을 정렬하거나, 방대한 양의 서열 데이터베이스에서 새로운 서열과 유사한 서열들을 빠르게 추출해 낼 수 있는 도구들이 필요하게 되었다. 이러한 이유로 서열 정렬을 보다 빠르고 정확하게 수행하는 알고리즘들이 많이 연구되어 왔다.

본 논문에서는 서열 정렬 알고리즘들에 관한 연구 동향을 제시한다. 2장에서는 기본적인 용어들을 정의하고, 3장에서는 두 서열간의 정렬을 위한 동적 프로그래밍(dynamic programming) 방식에 대해 살펴본다. 4장에서는 서열 데이터베이스의 빠른 검색을 위한 휴리스틱 알고리즘(heuristic algorithm)들을 살펴보고, 5장에서는 전체 유전체 서열 정렬 알고리즘을 살펴본다. 그리고 6장에서 결론을 맺는다.

2. 기본 정의

문자 집합 Σ상에서 정의된 길이가 n인 서열 S와 길이가 m인 서열 T가 존재할 때, $1 \le i \le n$인 i에 대해서, $S[i]$는 S의 i번째 문자를 나타내고, $1 \le j \le m$인 j에 대해서, $T[j]$는 T의 j번째 문자를 나타낸다. 이러한 조건이 주어졌을 때, 글로벌 정렬(global alignment), 최적 정렬(optimal alignment), 로컬 정렬(local alignment)은 다음과 같이 정의한다.

2.1 글로벌 정렬

정의 1. 주어진 서열 S와 T의 **글로벌 정렬**은 S와 T에 '-'(공란, space)를 삽입하여 동일한 길이 n'의 서열 S', T'로 변환하고, $1 \le i \le n'$인 i에 대해 $S'[i]$는 반드시 $T'[i]$에 대응되도록 한다.

글로벌 정렬과 로컬 정렬 시에는 Σ상에서 정의되지 않은 특별한 문자인 스페이스를 삽입할 수 있다. S와 T의 정렬 시 S의 어떤 위치에 스페이스를 삽입하는 것은, S의 해당 위치에서 한 문자의 삭제가 일어났음을 나타내고 동시에 T의 해당 위치에서 한 문자의 삽입이 일어났음을 나타낸다. 이것은 생물학적으로 돌연변이에 의해서 한 서열에서 문자의 삽입이나 삭제가 일어난 것을 의미한다.

글로벌 정렬의 결과 값은 다음과 같이 구할 수 있다. 정렬된 서열을 $1 \le i \le n'$인 모든 i에 대해 ($S'[i]$, $T'[i]$)의 짝(pair)으로 나타낸다. ($S'[i]$, $T'[i]$)에 대해서 다음 4가지 경우가 존재한다. $S'[i]$에 나타난 Σ상에서 정의된 문자를 a, $T'[i]$에 나타난 Σ상에서 정의된 문자를 c라고 할 때, [2]에서는 이 4가지 경우에 대해서 특정 값을 주는 점수 함수를 $\sigma(S'[i], T'[i])$로 정의하였다.

- $S'[i]=a$, $T'[i]=c$이고, $a=c$인 경우: $\sigma(a,a)$.
- $S'[i]=a$, $T'[i]=c$이고, $a \neq c$인 경우: $\sigma(a,c)$.

- $S'[i]=a$, $T'[i]=-$ 인 경우: $\sigma(a,-)$.
- $S'[i]=-$, $T'[i]=c$인 경우: $\sigma(-,c)$.

이 4가지 경우에 대해 각각 다른 값을 주는데, 첫 번째 경우는 상대적으로 높은 값을 준다. 예를 들어, $\sigma(a,a)=+2$, $\sigma(a,c)=\sigma(a,-)=\sigma(-,c)=-1$과 같이 점수 함수를 정할 수 있다. 글로벌 정렬의 결과 값은 점수 함수를 이용하여 다음과 같이 계산한다.

- $\sum_{i=0}^{n'}\sigma(S'[i],T'[i])$.

2.2 최적 정렬

정의 2. 주어진 서열 S와 T의 모든 글로벌 정렬들 중에서 그 결과 값이 최대로 되는 글로벌 정렬을 **최적 정렬**로 정의한다.

다음 [그림 1]은 점수 함수가 위의 예와 같이 주어졌을 때, 글로벌 최적 정렬의 예를 보여주고 있다.

A G G - C - G	A G G - C G -
- G G A C G G	- G G A C G G

[그림 1] $S=AGGCG$, $T=GGACGG$의 글로벌 최적 정렬의 예

2.3 로컬 정렬

정의 3. 주어진 서열 S와 T가 존재할 때, S의 부분서열(subsequence) x, T의 부분서열 y는 다음과 같은 성질을 가진다. x, y의 최적 정렬 값은 S와 T의 모든 부분 서열 짝들의 최적 정렬 값들 중 가장 큰 값을 가진다. 이 때 x, y는 S와 T에서 로컬 유사성을 갖는다고 하고, **로컬 정렬**은 이 두 부분 서열 x, y를 찾는다.

3. 동적 프로그래밍 방식

글로벌 정렬과 로컬 정렬은 동적 프로그래밍 방식을 통해서 최적 정렬을 구할 수 있다.

3.1 글로벌 정렬을 위한 동적 프로그래밍 방식

Needleman과 Wunsch의 알고리즘[3]은 가장 많이 사용되는 글로벌 정렬 알고리즘 중의 하나이다. 이 알고리즘은 동적 프로그래밍 방식을 가지고 글로벌 최적 정렬을 구한다. [2]에서 Needleman과 Wunsch의 알고리즘을 자세히 설명하고 있다.

주어진 서열 S와 T의 최적 정렬 값을 계산하기 위해서, $V(i,j)$를 $S[1]\cdots S[i]$와 $T[1]\cdots T[j]$의 최적 정렬의 값으로 정의한다. 그러면 S, T의 최적 정렬 값은 $V(n,m)$이 된다. 동적 프로그래밍 방식은 $0\le i\le n$, $0\le j\le m$인 모든 $V(i,j)$의 값을 계산한다. $V(i,j)$를 계산할 때, $V(i,j)$를 제외한 이전 값들, 즉, $V(0,0)\cdots V(i-1,j-1)$, $V(i-1,j)$, $V(i,j-1)$은 모두 최적 정렬된 값으로 계산되어 있다고 가정한다. 앞서 설명한 4가지 경우를 모두 고려하면 $V(i,j)$ 값은 다음과 같이 계산된다.

- $i>0, j>0$에 대해,

$$V(i,j)=\max(V(i-1,j-1)+\sigma(S[i],T[j]),\ V(i-1,j)+\sigma(S[i],-),\ V(i,j-1)+\sigma(-,T[j]))$$

$i=0$이거나 $j=0$인 경우는 다음과 같이 초기화 해 준다.

- $V(0,0)=0$
- $i>0$ 일 때, $V(i,0)=V(i-1,0)+\sigma(S[i],-)$
- $j>0$ 일 때, $V(0,j)=V(0,j-1)+\sigma(-,T[j])$

S, T의 글로벌 최적 정렬은 최적 정렬 값 $V(n,m)$으로부터 $V(0,0)$까지 역추적해서(retrace) 얻어진다. [그림 2]는 [그림 1]의 글로벌 최적 정렬을 동적 프로그래밍 방식을 통해서 계산한 예를 보여주고 있다.

i \ j		0	1 G	2 G	3 A	4 C	5 G	6 G
0		0	-1	-2	-3	-4	-5	-6
1	A	-1	-1	-2	0	-1	-2	-3
2	G	-2	1	1	0	-1	1	0
3	G	-3	0	3	2	1	1	3
4	C	-4	-1	2	2	4	3	2
5	G	-5	-2	1	1	3	6	5

[그림 2] $S=AGGCG$, $T=GGACGG$의 글로벌 정렬을 위한 동적 프로그래밍의 예

3.2 로컬 정렬을 위한 동적 프로그래밍 방식

Smith와 Waterman의 알고리즘[4]은 동적 프로그래밍 방식을 통해서 로컬 정렬을 구하는 알고리즘이다. 이 알고리즘은 Needleman과 Wunsch의 알고리즘에 기반을 둔다.

주어진 서열 S와 T의 로컬 정렬을 구하기 위해서, $V(i,j)$의 값을 $S[1]\cdots S[i]$와 $T[1]\cdots T[j]$의 로컬 최적 정렬의 값으로 정의한다. $V(i,j)$의 값은 다음과 같이 계산된다.

- $i>0, j>0$에 대해,

$$V(i,j)=\max(0,\ V(i-1,j-1)+\sigma(S[i],T[j]),\ V(i-1,j)+\sigma(S[i],-),\ V(i,j-1)+\sigma(-,T[j]))$$

Needleman과 Wunsch의 알고리즘에서 $V(i,j)$값을 계산하는 방법과 다른 점은 0을 고려한다는 것이다. 이것은 $S[i]$, $T[j]$으로부터 새로운 부분 서열이 시작한 경우를 고려한 것이다. 초기화 작업은 다음과 같이 해 준다.

- $V(i,0)=0$, $V(0,j)=0$.

S와 T의 로컬 정렬은 $0\le i\le n$, $0\le j\le m$인 모든 $V(i,j)$ 값들 중 가장 큰 값을 가지는 곳에서 역추적을 시작하여, 0을 값으로 가지는 곳에서 역추적을 멈춤으로써 얻어진다. [그림 3]은 로컬 정렬을 동적 프로그래밍 방식을 통해서 계산한 예를 보여주고 있다.

i \ j		0	1 C	2 C	3 G	4 C	5 T	6 C
0		0	0	0	0	0	0	0
1	G	0	0	0	2	1	0	0
2	C	0	2	2	1	4	3	2
3	A	0	1	1	1	3	3	2
4	T	0	0	0	0	2	5	4
5	C	0	2	2	1	2	4	4
6	G	0	1	1	4	3	3	3

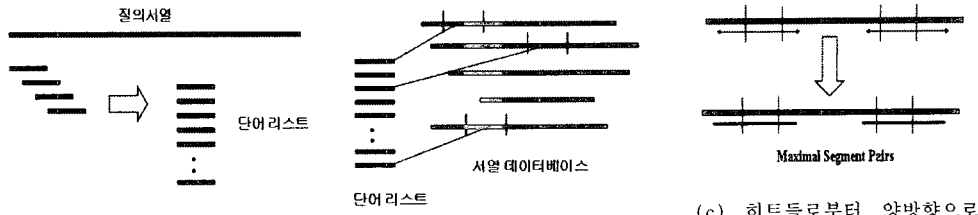
G	C	A	T
G	C	-	T

[그림 3] $S=GCATCG$, $T=CCGCTC$의 로컬 정렬의 예

4. 서열 데이터베이스 검색을 위한 휴리스틱 알고리즘

서열 데이터베이스 검색은 질의 서열(query sequence)이 입력되면, 질의 서열과 유사한 서열을 데이터베이스에서 검색하는 것을 말한다. 본 논문에서는 가장 대표적인 서열 데이터베이스 검색 도구인 BLAST와 최근 발표된 서열 데이터베이스 검색 알고리즘들에 대해 다루고 있다.

4.1 BLAST



(a) 질의 서열에 대해서, 길이가 w이고 점수가 한계값 이상인 모든 단어 리스트를 만든다.

(b) 단어 리스트를 가지고 서열 데이터베이스와 비교하여 정확하게 매칭하는 부분들(히트들)을 찾는다.

(c) 히트들로부터 양방향으로 확장해 나가면서 질의 서열과 데이터베이스 서열을 비교한다.

[그림 4] BLAST의 기본 알고리즘

BLAST(Basic Local Alignment Search Tool)[5]는 서열 데이터베이스 검색을 위해서 가장 널리 사용되고 있는 도구이다. BLAST는 미국 NIH(National Institute of Health) 산하의 NCBI(National Center for Biotechnology Information)에서 연구 개발되었다.

BLAST는 질의 서열과 로컬 유사성을 가지는 부분 서열을 서열 데이터베이스 내에서 찾는다. BLAST는 이 부분 서열을 찾기 위해서, MSP(Maximal Segment Pair) 점수를 정렬의 측정 기준으로 정하고, MSP 점수가 일정 값 이상인 부분 서열들을 찾는다. MSP는 다음과 같이 정의한다.

■ 두 서열(질의 서열, 데이터베이스의 한 서열)로부터 선택된 같은 길이의 부분 서열로서 가장 높은 점수를 가지는 서열

■ 길이를 확장하거나 줄임으로써 점수가 향상되지 않는 맥시멀(maximal) 부분 서열

서열 데이터베이스는 방대한 양이고 실제 새로운 서열과 유사성이 높은 서열은 데이터베이스에서 극히 일부분이다. 따라서 일정 값 이상을 가지는 MSP들에 대해 초점을 맞추고 데이터베이스 검색 시 MSP가 될 만한 후보들만 골라내어 속도를 향상 시킨다. BLAST는 서열 비교 시 점수 행렬(scoring matrix)을 사용한다. 점수 행렬은 모든 아미노산들에 대해 각각 서로 치환될 수 있는 가능성에 점수를 주어 행렬 형태로 나타내었다. 대표적인 점수 행렬로 PAM과 BLOSUM이 있다.

BLAST의 기본 알고리즘은 다음 3단계로 구성되어 있다.

[단계 1] 질의 서열의 모든 위치에 대해 해당 위치에서 시작하는 w길이의 부분 서열을 만든다. 점수 행렬을 사용해서 이 부분 서열과 비교했을 때, 점수가 한계 값(threshold value) 이상이 되는 w길이의 모든 단어(w-mers)의 리스트를 생성한다.

[단계 2] 단어 리스트를 가지고 데이터베이스에 대해 검색을 수행하여 정확하게 매칭(matching)하는 부분인 히트들(hits)을 찾는다.

[단계 3] 각 히트들을 질의 서열과 데이터베이스 서열에서 양방향으로 확장해 나가면서 비교한다. 이때 총 점수가 차단 값(cut-off value) 이상이 될 때까지만 확장한다.

[6]에서는 BLAST의 기본 알고리즘을 그림으로 표현했는데, 이것이 [그림 4]에 나타나 있다. [그림 4]의 (a), (b), (c)는 각각 알고리즘의 기본 3단계를 나타내고 있다.

4.2 SSAHA

2001년 발표된 SSAHA(Sequence Search and Alignment by Hashing Algorihtm)[7]는 수 기가(giga) 용량의 서열 데이터베이스에 대해 빠른 검색을 수행하기 위해서 해싱(hashing) 알고리즘을 사용한다.

SSAHA 알고리즘은 크게 다음 3 단계로 나누어진다.

[단계 1] 데이터베이스에 있는 서열들로부터 서로 겹쳐지지 않는(non-overlapping) 연속된 k개의 문자들의 서열인 k-tuple들을 생성하고, 각 k-tuple들의 위치를 저장하는 해쉬 테이블(hash table)을 만든다.

[단계 2] 질의 서열에서 서로 겹쳐지는(overlapping) k-tuple들을 생성하고, 해쉬 테이블에서 검색한다.

[단계 3] 해쉬 테이블 검색 결과 나온 히트들을 소팅(sorting)한다. 소팅된 히트들을 읽어나가면서, 서로 인접한 위치에 있는 히트들을 찾아낸다.

[단계 1]은 다음과 같이 수행된다. 데이터베이스 내의 서열들의 총 개수를 d라고 할 때, 이 서열들은 $1 \le i \le d$ 인 i에 대해 $S_1, \cdots, S_{,,}, \cdots, S_d$로 나타낼 수 있다. 한 k-tuple이 S_i 서열의 j 위치에서 시작할 때, 이 k-tuple은 (i, j)의 짝으로 나타낸다. 염기 서열에서 나올 수 있는 모든 k-tuple은 4^k개이고, 각 k-tuple은 하나의 숫자로 인코딩(encoding)된다. 이 인코딩된 숫자들이 해쉬 테이블의 키(key)가 된다. [그림 5]는 $S_1 = GACTCAAGAA$, $S_2 = ACGGCATTCG$, $S_3 = GGATCAGC$로 구성된 데이터베이스에 대한 해쉬 테이블의 예이다. 이 예에서, w는 k-tuple을 나타내고, E(w)는 k-tuple을 인코딩한 수를 나타낸다. 그리고 Position은 해당 k-tuple의 데이터베이스 내에서의 위치로서 (i, j)의 짝으로 나타낸다.

w	E(w)	Position	w	E(w)	Position
AA	1	(1,9)	GA	9	(1,1)
AC	2	(2,1)	GC	10	(3,7)
AG	3	(1,7)	GG	11	(2,3) (3,1)
AT	4	(3,3)	GT	12	
CA	5	(1,5) (2,5) (3,5)	TA	13	
CC	6		TC	14	
CG	7	(2,9)	TG	15	
CT	8	(1,3)	TT	16	(2,7)

[그림 5] 해쉬 테이블의 예.

[단계 2]는 질의 서열로부터 서로 겹쳐지는 k-tuple들을 만들고 해쉬 테이블로부터 검색하여 히트들을 찾는다. 해당 k-tuple의 시작 위치를 t라고 할 때, 히트 H는 다음과 같이 표현된다.

■ $H = (i, j - t, j)$.

[단계 3]은 [단계 2]에서 찾아진 모든 히트들을 가지고 소팅하는데, 첫 번째 소팅 기준은 i이고 두 번째 소팅 기준은 $j - t$이다. 소팅되어 있는 상태에서는 서로 인접해 있는 히트들을 쉽게 찾아낼 수 있다. [단계 3]의 마지막 부분에서 연속된 히트들을 찾아 결과로 출력한다.

논문에 나타난 실험 결과에 따르면, SSAHA 알고리즘은 BLAST에 비해서 3~4배 정도 빠르다. 그러나 서열 데이터베이스에 대한 해쉬 테이블을 메모리에 저장하고 있어야 하므로 메모리 소모가 크다.

실제 웹상에서 서비스되고 있는 SSAHA는 각 종들의 전체 유전체 서열 데이터베이스 각각에 대해서 해쉬 테이블을 유지하고, 사용자가 데이터베이스를 검색할 때 특정한 종에 대한 전체 유전체 서열 데이터베이스를 선택하도록 한다.

SSAHA는 SNP(Single Nucleotide Polymorphism)를 감지하는 데 유용하게 사용되고, 매우 큰 크기의 서열 조립(sequence assembly)에 사용된다.

4.3 BLAT

BLAT(BLAST-Like Alignment Tool)[8]은 2002년 발표된 서열 데이터베이스 검색 도구이다. BLAT은 SSAHA와 마찬가지로 전체 유전체 서열 데이터베이스에 대한 인덱스를 만든다.

BLAT 알고리즘은 다음과 같이 동작한다.

[단계 1] 데이터베이스에 있는 서열들로부터 서로 겹쳐지지 않는 k-mer들을 생성하고, k-mer들에 대한 인덱스를 만든다.

[단계 2] 질의 서열에 대해 서로 겹쳐지는 k-mer들을 생성하고, 이미 만들어진 인덱스를 이용해서 k-mer들의 히트들을 찾는다.

[단계 3] 동적 프로그래밍 방식을 이용하여 히트들을 확장하면서 인접한 히트들은 서로 연결한다.

[단계 4] 이전 단계에서 얻어진 연결된 히트들을 한데 묶어서 더 큰 정렬로 만든다.

위에 나타난 BLAT 알고리즘은 대체적으로 BLAST의 알고리즘과 유사하다. BLAT 역시 먼저 히트들을 찾은 다음 확장한다.

그러나 BLAT 알고리즘은 BLAST 알고리즘과 여러 가지 면에서 다르다. 첫 번째로 BLAST는 질의 서열에 대해서 인덱스를 만들고 데이터베이스를 읽어나가면서 히트들을 찾지만, BLAT은 데이터베이스의 인덱스를 만들고 질의 서열을 순차적으로 읽어나간다. 이로 인해 BLAT은 BLAST보다 훨씬 빠른 검색 속도를 가진다. 한번 데이터베이스에 대한 인덱스가 만들어지면 히트들을 찾을 때 질의 서열을 읽는 시간만 소요되기 때문에, 빠른 시간 내에 히트들을 찾아낼 수 있다. 두 번째 차이점은 BLAST는 하나의 완벽하게 매치되는 히트에 대해서 확장하지만, BLAT은 히트들이 서로 인접해서 여러 개 발생하거나 히트에서 한 문자 정도 불일치가 발생하더라도 확장이 일어난다. 세 번째 차이점은 BLAST는 히트들을 확장하여 얻어진 MSP를 각각 독립된 결과로 알려주지만, BLAT은 확장하여 얻어진 부분을 엮어서 더 큰 정렬로 만들고 이것을 결과로 알려준다.

BLAT은 실험 결과 엮기 서열의 데이터베이스 검색 시에는 BLAST에 비해서 500배 빠르고, 아미노산 서열에 대해서는 50배 빠르다. 그러나 SSAHA와 마찬가지로 전체 유전체 서열에 대한 인덱스를 저장하기 때문에 많은 메모리를 필요로 한다.

5. 전체 유전체 서열 정렬 알고리즘

길이가 매우 긴 전체 유전체 서열 정렬 시 기존의 알고리즘들은 매우 느리게 동작하거나 아예 동작하지 않는다. 이러한 이유로 전체 유전체 서열 정렬을 위한 알고리즘들이 개발되었다.

1999년 발표된 논문 [9]는 전체 유전체 서열 정렬 알고리즘이다. 이 알고리즘은 전체 유전체 서열 정렬을 위하여 써픽스 트리(suffix tree)를 사용한다. 써픽스는 서열의 각 위치에서 시작해서 서열의 끝까지 확장되는 부분 서열이다. 써픽스 트리는 한 서열의 모든 써픽스들을 트리형태로 저장한 자료 구조이다. 써픽스 트리는 다음과 같은 몇 가지 특징을 가진다.

- 루트 노드(root node)에서부터 단말 노드(leaf node)까지의 패스(path)에 있는 문자들을 모두 합치면 하나의 써픽스가 된다. 즉, 각 단말 노드는 하나의 써픽스에 대응된다.
- 각 내부 노드(internal node)는 최소 두 개의 자식 노드(child node)를 가진다.
- 각 내부 노드는 내부 노드를 루트 노드로 하는 서브트리(sub-tree)내의 단말 노드들이 나타내는 써픽스들 사이에 중복되는 서열이다. 내부 노드가 나타내는 숫자는 중복되는 서열의 길이다.

써픽스 트리는 선형 시간 내에 생성가능하다. 이 논문은 McCreight[10] 알고리즘을 사용하여 적은 공간을 차지하는 써픽스 트리를 생성한다.

입력으로 들어온 두 유전체 서열을 G_A, G_B라고 할 때, G_A, G_B는 생물학적으로 상관관계가 있는 서열이라고 가정한다. 이 알고리즘은 가장 먼저 다음과 같이 정의된 MUM(Maximal Unique Match)을 찾는다.

- G_A, G_B 사이에 서로 매칭되는 같은 길이의 부분 서열들 중에서 가장 길고 오직 하나 존재하는 것.

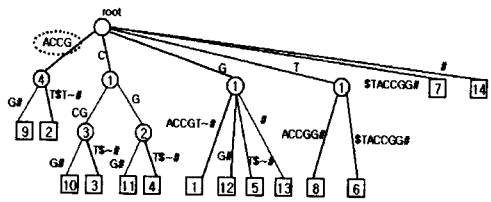
정렬 알고리즘은 다음과 같이 진행된다.

[단계 1] 입력된 두 유전체 서열을 가지고 써픽스 트리를 구축한다. 써픽스 트리를 이용하여 일정 길이 이상의 MUM들을 찾아낸다.

[단계 2] 찾은 MUM들을 소팅한다. 이 때 해당 MUM들의 G_A에서의 위치를 기준으로 소팅한다.

[단계 3] 찾아진 MUM들 사이를 동적 프로그래밍 방식을 이용해서 정렬한다.

[단계 1]은 다음과 같이 수행된다. G_A, G_B 사이에 특수한 문자를 삽입하여 새로운 서열을 만들고 이 서열을 가지고 써픽스 트리를 구축한다. 그리고 G_A, G_B로부터 나온 두 단말 노드들의 내부 노드로서 일정 값 이상을 가지는 노드들을 찾으면, 이 내부 노드가 나타내는 서열이 MUM이 된다. [그림 6]은 G_A, G_B로 만든 써픽스 트리의 예를 보여주고 있다. 그림에서 점선으로 된 동그라미로 둘러싸여진 서열이 G_A, G_B의 MUM이다.



[그림 6] $G_A = GACCGT$, $G_B = TACCGG$로 만든 써픽스 트리의 예

MUM들이 서로 겹쳐지게 되면 [단계 3]에서 큰 정렬로 만들 수 없으므로, [단계 2]는 MUM들을 G_A에서의 위치순으로 소팅하여 MUM들이 서로 겹쳐지지 않도록 한다.

마지막으로 [단계 3]에서 소팅된 MUM들 사이를 동적 프로그래밍 방식을 적용해서 확장해 나가면서 큰 정렬로 만들고 결과를 출력한다.

6. 결론

본 논문에서는 서열 정렬 알고리즘들에 관한 연구 동향을 제시했다. 기하급수적으로 생물학 데이터양이 증가함에 따라 보다 빠르고 정확한 알고리즘들이 개발되었다. 서열 데이터베이스 검색 분야에서는 전체 데이터베이스에 대한 인덱스를 메모리에 저장하여 검색 속도를 향상시키는 알고리즘들이 개발되었다. 또한 전체 유전체 서열이 분석된 종이 증가함에 따라 전체 유전체 서열 정렬 알고리즘들이 개발되었다.

[참고문헌]

[1] D.Gusfield, *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*, Cambridge University Press, 1997

[2] M.Tompa, "Lecture Notes on Biological Sequence Analysis," Tech. Rep. 2000-06-01, Department of Computer Science and Engineering University of Washington, 2000

[3] S.B.Needleman, C.D.Wunsch, "A general method applicable to the search for similarities in the amino acid sequences of two proteins," J. Mol. Biol. 48, pp.443-453, 1970

[4] T.F.Smith, M.S.Waterman, "Comparison of biosequences," Adv. Appl. Math. 2, pp.482-489, 1981

[5] S.Altschul, W.Gish, W.Miller, E.Myers, D.Lipman, "Basic Local Alignment Search Tool," J. Mol. Biol. 215, pp.403-410, 1990

[6] http://bric.postech.ac.kr/topic/12.html

[7] Z.Ning, A.J.Cox, J.C.Mullikin, "SSAHA: A fast search method for large DNA databases," Genome Res. 11, pp.1725-1729, 2001

[8] W. James Kent, "BLAT: The BLAST-like Alignment Tool," Genome Res. 12(4), pp.656-664, 2002

[9] A.L.Delcher, S.Kasif, R.D.Fleischmann, J.Peterson, O.White, S.L.Salzberg, "Alignment of whole genomes," Nucleic Acids Res. 27(11), pp.2369-2376, 1999

[10] E.M.McCreight, "A space-economical suffix tree construction algorithm," J. ACM 23(2), pp.262-272, 1976