단백질 이차 구조에 대한 패턴 검색을 위한 클러스터 세그먼트 인덱싱

서민구[O] 박상현 원정임
연세대학교 컴퓨터과학과
{mkseo[O], sanghyun, jiwon}@cs.yonsei.ac.kr

Clustered Segment Indexing for Pattern Searching on the Secondary Structure of Protein Sequences

Minkoo Seo[O], Sanghyun Park and JungIm Won
Dept. of Computer Science, Yonsei University

요 약

바이오 인포메틱스에서의 데이터 검색은 DNA와 단백질 시퀀스에 대해서 주로 이루어지며, 지금까지의 연구는 주로 DNA와 단백질 1차 구조의 검색에 대해서만 수행되었다. 단백질 2차 구조는 1차 구조 내 인접한 아미노산들의 공간적인 배열을 나타내며, 단백질의 기능을 예측하는데 중요한 3차구조의 지역적 아미노산의 특성을 나타낸다. 따라서 2차 구조에 대한 검색은 단백질의 기능을 이해하는데 매우 중요한 역할을 한다. 본 논문에서는 단백질 2차 시퀀스의 효율적 검색 위하여 새로운 질의 언어를 정의하고, 세그먼트를 조합한 클러스터 구조 및 Look Ahead를 사용해 Exact Match, Range Match, Wildcard Match 질의를 효율적으로 처리할 수 있는 기법을 제시하였다. 또한 단백질 2차 데이터 집합에 대해 인덱싱의 효율성을 검증하였으며, 실험 결과 제안한 방식을 사용하여 패턴 질의의 고속 검색이 가능함을 확인하였다.

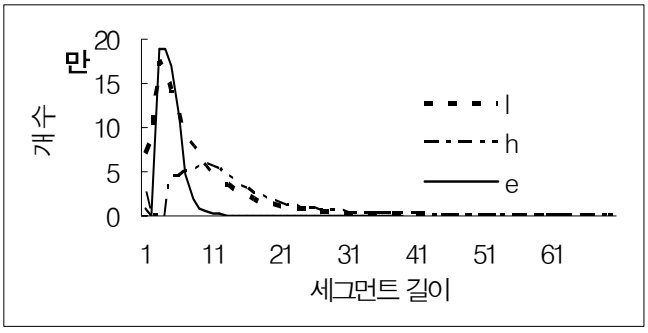
1. 서론

바이오 인포메틱스에서 처리할 데이터의 양은 15~16개월에 2배씩 증가하고 있으며, DNA 또는 단백질 시퀀스에 대한 검색은 이미 특성이 판명된 시퀀스 데이터베이스로부터 미지의 질의 시퀀스와 염기 혹은 아미노산 서열이 유사한 시퀀스를 찾는 연산이다. 이러한 연산을 통해 미지의 시퀀스의 기능, 역할, 구조, 분류 등을 예측할 수 있다[1][9].

DNA 시퀀스 혹은 단백질 1차 구조를 위한 인덱싱 및 검색 기법에 관한 연구 결과가 최근 많이 보고되고 있으나[7][8][10][14], 이에 비해 단백질 2차 구조에 관한 연구는 부족한 실정이다.

단백질 2차 구조 시퀀스는 e (beta sheets), h (alpha helices), l (turns 또는 loop)의 세 가지 기본 구조를 나타내는 문자로 구성되며, 각 문자는 연속되어 나타나는 경향이 있다. 예를 들어, 'helehleh' 와 같은 시퀀스 보다는 'hhhhheeeeellleeeeeehhh' 와 같이 문자가 연속적으로 나타나는 경향이 있다.

[6]에서는 이러한 단백질 2차 구조 시퀀스 검색을 위한 질의 언어를 제안하고 세그먼트 개념을 사용한 인덱싱 방법을 제안하였다. 세그먼트는 각 문자의 연속을 Type(문자의 종류), Len(세그먼트의 길이)로 나타낸 개념이다. 예를 들어, 앞서의 시퀀스는 'hhhhh/eeeee/lll/eeeeee/hhh' 로 분할되어 (h,5)(e,5)(l,3)(e,6)(h,3) 의 5개 세그먼트로 표현된다. 해당 논문에서는 이들 각각의 세그먼트를 인덱싱 하였으나, 5장에서 사용한 83,072개의 시퀀스를 분석한 결과, <그림 1>에서 볼 수 있듯이 e 세그먼트의 경우 세그먼트 길이 3~6사이에 전체 세그먼트의 87%, h세그먼트는 5~14사이에 62%, l 세그먼트는 3~6사이에 41%가 모여 있어 대부분의 세그먼트 길이가 특정 범위에 편중되어 있다.



<그림 1> 세그먼트의 길이 분포

따라서 세그먼트만으로는 인덱스 정선도에 한계가 있다. 반면 중첩 간격(overlapping interval) 기반의 인덱스 구조[2][8][10][14]는 일정개수의 문자를 인덱싱 대상으로 하여 좋은 검색 성능을 얻고 있다.

또한 [6]에서 제안한 질의 언어에서는 갭(Gap)을 세그먼트 단위로 제한하여 질의 처리 능력에 한계가 있다.

따라서 본 논문에서는 [6]에서 단백질 2차 구조 검색을 위해 정의한 질의 언어를 수정하여 갭(Gap)을 세그먼트가 아닌 문자 단위로 지정할 수 있는 질의 언어를 제안한다. 그 뒤, 일정 개수의 문자를 인덱싱 대상으로 하기 위한 클러스터 세그먼트 인덱싱 기법을 제안하고, 이를 활용한 질의 처리 기법을 제시한다. 또한 검색 성능의 향상을 위해 Look Ahead를 개념을 추가 적용한다.

2. 관련 연구

DNA 및 단백질 시퀀스 검색에 가장 널리 사용되는 검색 시스템은 BLAST와 FASTA가 있다. 이 두 시스템은 시퀀스를 휴리스틱 기법을 사용해 탐색 대상을 줄이는 방법을 사용한다. 그러나 시퀀스 데이터의 양이 급속도로 증가함에 따라, 대량의 메모리와 병렬 컴퓨터를 필요로 하기 때문에 검색 비용에 대한 오버헤드가 증가될 수 있다[9]. 따라서 최근 데이터베이스 분야에서는 인덱싱 기법을 적용한 연구가 활발히 진행되고 있다.

접미어 트리(suffix tree)[15]는 시퀀스 검색을 위한 좋은 구조로 사용되어 왔으며 [7]에서는 메인 메모리보다 큰 대용량 데이터에 대한 접미어 트리 구성 알고리즘을 제시하였다. 그러나 접미어 트리는 저장 공간 오버헤드가 크고, 페이지 단위의 저장이 어렵다는 단점이 있다[12][15].

[14]에서는 짧은 길이의 패턴 검색을 위한 기법인 RAMdb를 제안하였다. 이 방식은 휴리스틱 기법에 비해 0~800배의 검색 성능 향상을 보였으나, 인덱싱에 사용한 간격(interval)보다 조금 짧거나 긴 패턴만 검색이 가능하다는 한계가 있다[9]. 또한 해당 방식은 패턴 검색의 대상이 단백질 1차 시퀀스와 DNA이었던 이유로 시퀀스 내 문자가 반복하여 나타나는 특성을 가진 단백질 2차 시퀀스에 적용할 경우 검색 효율성이 떨어지게 된다.

단백질 2차 구조의 패턴 검색에 대한 직접적인 연구로, 참고문헌 [6]에서는 단백질 2차 시퀀스가 e(beta sheets), h(alpha helics), l(turns or loops)의 3가지 문자의 연속으로 구성된다는 점에 착안하여 시퀀스에 출현하는 각 문자와 각 문자의 출현 빈도를 활용한 B+ 트리 기반의 세그먼트 인덱싱 기법을 제안하였다. 그러나 [6]의 질의 언어에서는 갭(Gap)이 단일 문자로 구성된 세그먼트 단위로만 가능하다는 제약이 있으며, 동일 길이를 갖는 동일 문자의 행이 세그먼트 테이블에 존재할 가능성이 많아져 검색 속도가 저하될 수 있다는 단점이 있다.

본 논문에서는 [6]에서 제안한 질의 언어의 갭(Gap)을 단일 문자로 구성된 세그먼트 단위가 아닌 문자 단위로 지정할 수 있는 새로운 질의 언어를 정의하고, 클러스터 및 Look Ahead에 기반한 인덱싱 기법 및 고속의 패턴 검색 알고리즘을 제안한다.

3. 질의 언어

단백질 2차 구조의 검색을 위해 본 논문에서 제안하고 있는 질의 언어는 [6]의 질의 언어를 기반으로 한다. 그러나 제안된 질의 언어는 보다 유연성 있는 Wildcard Match등의 질의를 지원하기 위해 [6]에서 제안된 모델을 확장, 적용한다. <표 1>에 제안된 질의 언어를 보인다.

<표 1> 질의 언어 정의

```
Query → (SegmentAndGap | GapAndSegment |
          Segment)+
SegmentAndGap → Segment Gap
GapAndSegment → Gap Segment
Segment → <type lb ub>
Gap → <? lb ub>
type → e | h | l
lb → 0이상의 임의의 정수
ub → 0이상의 임의의 정수 | ∞ (단, lb ≤ ub)
```

제안된 질의 언어는 단백질 2차 구조가 갖는 다음과 같은 특성을 반영한다. (1) 세그먼트 내에는 e, h, l의 세 가지 문자만 나타날 수 있다. (2) 세그먼트 내 각 문자는 연속적으로 나타난다. 예를 들어, ‘ehleheh’와 같은 시퀀스 보다는 ‘eeehhhhlll’와 같이 문자가 반복적으로 나타나는 시퀀스가 많다.

제안된 질의 언어로 처리할 수 있는 질의 형태는 다음과 같다. 가장 간단한 형태의 질의는 Exact Match이다. 예를 들어, <e 3 3><h 2 2><l 4 4>은 eeehhllll 의 시퀀스를 검색한다. 두 번째는 Range Match로, <e 3 5><h 2 5><l 3 6>를 예로 들 수 있다. 질의는 e, h, l의 순서로 세그먼트가 나오되 각 세그먼트의 길이가 3~5, 2~5, 3~6인 시퀀스를 검색하기 위한 질의이다. 마지막 형태는 2차 시퀀스내의 갭(Gap)을 표현하기 위한 Wildcard Match로, 문자 ‘?’를 사용한다. 예를 들어, <h 4 6><? 0 ∞><l 5 5>은 h 세그먼트와 l 세그먼트를 찾되 중간에 0~∞의 길이를 갖는 임의개의 문자가 올 수 있음을 표현한다. 제안된 질의 언어에 의해

'hhhhhh/hhee/llll' (=<h 8 8><e 2 2><l 5 5>)가 검색되어 질 수 있다. 이는 갭(Gap) 역시 언제나 세그먼트로 주어진다고 전제하여 <h 4 6><? 0 ∞><l 5 5>의 질의를 길이 4~6의 h 세그먼트, 길이 5의 l 세그먼트를 찾되 그 사이에 임의 길이세그먼트를 허용하는 검색으로 질의 형태로 해석하여 시퀀스 'hhhhhh/hhee/llll'를 검색하지 못하는 [6]에서의 갭 세그먼트의 개념을 문자 단위로 확장한 것이다.

4. 클러스터 세그먼트 인덱스

본 장에서는 단백질 2차 구조의 효율적 검색을 위한 인덱스 구조를 제안한다. 제 4.1절에서는 단백질 데이터를 저장하고 있는 단백질 테이블의 구조에 대해 설명하고, 제 4.2절에서는 본 논문에서 제안하고 있는 클러스터 세그먼트 테이블의 구성 방식과 이를 활용한 인덱싱 방안에 대해 논의한다. 제 4.3절에서는 질의 최적화를 위한 기법으로 히스토그램 활용 방안을 논의하고, 제 4.4절에서는 인덱스 압축 방식에 논의한다.

4.1 단백질 테이블

본 논문에서 단백질 데이터를 저장하기 위해 사용한 단백질 테이블의 구조를 <표 2>에 보인다. 단백질 테이블은 단백질의 ID, 1차 구조, 2차 구조로 구성된다[6].

<표 2> 단백질 테이블의 구조

필드 명	의 미
ID	단백질의 ID
Primary	단백질의 1차 구조
Secondary	단백질의 2차 구조

4.2 클러스터 세그먼트 테이블

참고 문헌 [6]에서는 단백질 2차 구조에 대해 세그먼트 개념을 이용한 인덱스 구성 방법을 제안하고, 이를 기반으로 한 질의 처리 방법을 제안하였다. 그러나, 이 방식은 단백질 시퀀스를 구성하는 각 문자와 그 길이 정보만을 이용한 단일 세그먼트를 인덱싱 대상으로 하기 때문에 문자와 길이 정보가 같은 중복 세그먼트의 수가 많아져 검색 성능을 저하시킬 수 있다는 단점이 있다.

본 논문에서는 이러한 문제점을 해결하기 위한 방안으로 우선, 여러 개의 세그먼트를 클러스터 단위로 그룹핑하는 방식을 적용한 클러스터 세그먼트 인덱스 구성 방안을 제안하고, 이를 활용한 질의처리 방안을 제안한다. 또한, 제안된 인덱스의 정선도(selectivity)를 향상시키기 위한 방안으로 LookAhead개념을 도입한 새로운 인덱싱 방안을 제안한다.

4.2.1 세그먼트의 구성

단백질 2차 구조 검색을 위해 본 논문에서는 단백질 2차 시퀀스를 구성하는 각 문자를 세그먼트 단위로 분할하는 방식을 사용한다[6]. 분할된 세그먼트는 <표 3>에 보인 바와 같이 단백질 ID, 세그먼트의 시작 위치, 유형, 길이, LookAhead 정보로 구성된다. 특히, LookAhead는 인덱스의 정선도와 검색 속도를 향상시키기 위한 추가 정보로, 해당 세그먼트 뒤에 오는 n개의 세그먼트를 나타낸다.

<표 3> 세그먼트의 구조

필드 명	의 미
ID	단백질의 ID
Loc	단백질 2차 구조 내의 세그먼트 시작 위치
Type	세그먼트의 유형(E, H, L 중 하나)
Len	세그먼트의 길이
LookAhead	해당 세그먼트 뒤에 오는 n개 세그먼트

예를 들어 단백질 2차 시퀀스 'eeehhllee'은 n=2인 경우, 다음과 같이 4개의 세그먼트로 분할된다.

<표 4> 세그먼트 테이블의 예

ID	Loc	Type	Len	LookAhead
1	0	e	3	hl
1	3	h	2	le
1	5	l	2	e
1	7	e	2	

세그먼트는 단백질 2차 시퀀스를 연속된 문자 단위로 분할하는 개념이며, 실제 저장은 이렇게 분할된 여러 개의 세그먼트를 그룹핑한 클러스터 단위로 이루어진다.

4.2.2 클러스터의 구성

본 논문에서는 인덱스 정선도 향상을 위해 단백질 2차 시퀀스를 분할하여 얻어진 여러 개의 세그먼트를 클러스터 단위로 그룹핑하는 방식을 사용한다. 제안된 방식에서는 연속된 2^k (단, $k \geq 0$인 정수)개의 세그먼트를 묶어 클러스터를 구성하고, 이를 클러스터 테이블에 저장한다. <표 5>에 클러스터 테이블의 구조를 보인다.

<표 5> 클러스터 테이블의 구조

필드 명	의 미
ID	표 2 참고
Loc	표 2 참고
TypeStr	2^k개 세그먼트의 유형
TypeLen	2^k개 세그먼트의 총 길이
LookAhead	표 2 참고

예를 들어 단백질 2차 시퀀스 'eeehhllee'는 eee/hh/ll/ee의 세그먼트로 분할 된 뒤, 2^k개의 세그먼트가 묶여 <표 6>와 같이 k=0,1,2인 클러스터 테이블에 각각 저장된다.

<표 6> 클러스터 테이블의 예

k=0

ID	Loc	TypeStr	TypeLen	LookAhead
1	0	e	3	hle
1	3	h	2	le
1	5	l	2	e
1	7	e	2	

k=1

ID	Loc	TypeStr	TypeLen	LookAhead
1	0	eh	5	le
1	3	hl	4	e
1	5	le	4	

k=2

ID	Loc	TypeStr	TypeLen	LookAhead
1	0	ehle	9	

본 논문에서는 구성된 단백질 테이블과 클러스터 테이블을 위한 인덱스 구조로 B+트리를 활용한다.

4.3 히스토그램의 구성

본 논문에서는 클러스터 세그먼트 인덱스 방식이외에도 질의 최적화를 위해 주어진 질의를 만족하는 해의 개수를 미리 예상하는 방식을 추가 적용한다. 이를 위해 클러스터 테이블의 TypeLen를 이용한 히스토그램 테이블을 구성하고, 이를 질의 처리 시 추가 정보로 활용한다. <표 7>에 히스토그램 테이블 구조를 보인다.

<표 7> TypeLen를 이용한 히스토그램 테이블

필드 명	의 미
k	클러스터 테이블의 k
TypeLen	TypeLen길이
Occurrence	해당 TypeLen을 갖는 튜플의 수

4.4 클러스터 세그먼트 테이블의 압축

본 논문에서는 제안된 클러스터 테이블의 압축 표현을 위해 반-정적 제로 오더 문자 수준(semi-static zero-order character-level) 모델을 사용한다. 이 방식은 시퀀스를 구성하는 각 문자의 출현 빈도를 고려한 인코딩 방식으로, 압축을 위해 해당 시퀀스 데이터를 한 번 더 읽어야 하므로 효율이 다소 떨어지는 단점이 있다. 그러나 정적 모델(static model)과 달리 데이터의 특성에 따라 압축 효율이 저하되는 일이 없으며, 적응 모델(adaptive model)과 달리 인코딩과 디코딩 시 많은 메모리를 필요하지 않다는 장점이 있다[11].

<표 8> 각 문자의 이진 표현 예

문 자	이진 표현
l	0
h	10
e	11

본 논문에서는 단백질 2차 시퀀스를 구성하는 각 문자의 출현 빈도를 고려하여, 가장 높은 출현 빈도를 갖는 문자를 '0', 그 외 문자를 각각 '10'와 '11'의 이진으로 표현하는 바이너리 인코딩(Binary Encoding)방식을 사용하여 데이터를 압축 표현한다. <표 8>에 단백질 2차 시퀀스를 구성하는 각 문자에 대한 이진 표현 예를 보인다.

예를 들어 시퀀스 'ehle'는 '11100110'로 표현, 저장된다. 예에서 인코딩된 가장 끝의 0은 바이트 단위로 저장하기 위한 패딩이다.

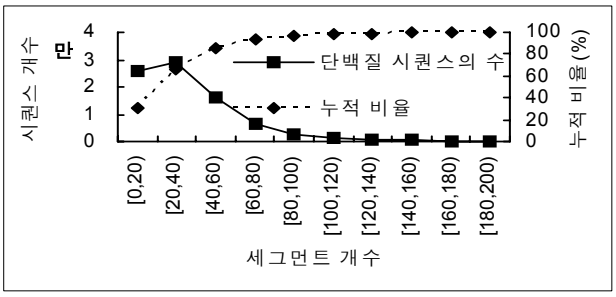
5. 제안된 방식의 효율 분석

PIR[13]내의 단백질 1차 시퀀스 중 83,072개를 PREDATOR[3][4]를 사용해 2차 시퀀스로 변환하여 단백질 테이블에 저장한 뒤, 데이터의 특성을 조사하였다. 구성된 단백질 테이블의 크기는 약 72.12MB이다.

5.1 세그먼트의 개수 분포

실험 1에서는 단백질 2차 시퀀스에 대해 세그먼트의 분포도를 실험하였다. <그림 2>에서 보인 바와 같이 세

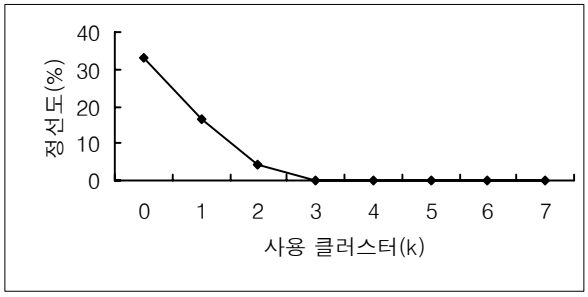
그먼트 개수가 [20,40)인 시퀀스가 가장 많았으며, 약 99%의 시퀀스가 140개미만의 세그먼트를 갖는 것으로 나타났다. 따라서 이후 실험에서는 클러스터의 크기를 최대 $\lfloor \log_2 140 \rfloor$(≒ 7)로 정하여 사용한다.



<그림 2> 세그먼트의 개수 분포

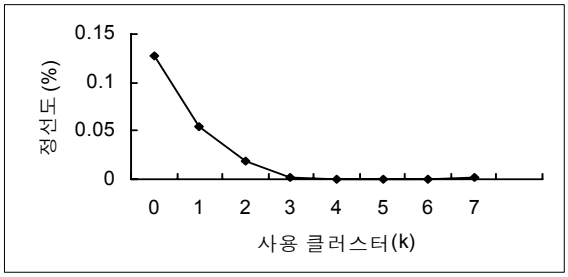
5.2 TypeStr, TypeLen의 정선도

실험 2는 본 논문에서 제안된 인덱스의 정선도를 알아보기 위한 실험으로, (1) TypeStr만을 인덱싱 대상으로 한 경우의 정선도를 <그림 3>에 보인다. k≧3인 경우 TypeStr이 동일한 튜플의 비율이 전체 튜플의 0.5% 미만인 것으로 나타났다.



<그림 3> TypeStr의 정선도

(2) TypeStr와 TypeLen를 모두 인덱싱 대상으로 한 경우의 인덱스 정선도를 <그림 4>에 보인다. k≧3부터 (TypeStr, TypeLen)이 동일한 튜플의 수가 평균 100개 이하로 나타났다.

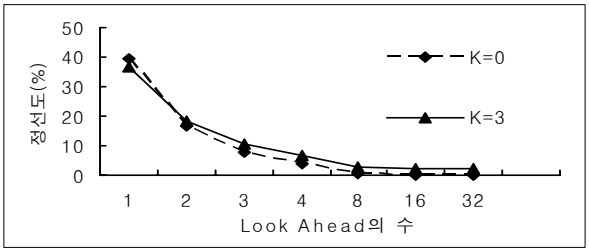


<그림 4> TypeStr+TypeLen의 정선도

즉, 실험 (1), (2)의 결과는 k≧3 인 경우에 제안된 클러스터 인덱스의 정선도가 좋음을 나타낸다.

5.3 LookAhead의 필터링 효과

TypeStr과 TypeLen이 동일할 때, LookAhead에 의해 필터링 되는 튜플의 비율은 k와 거의 무관한 것으로 나타났다(<그림 5> 참조). 특히, LookAhead가 8인 경우 약 97.5%의 튜플이 필터링 되는 것으로 나타났다.



<그림 5> LookAhead의 수와 정선도

5.4 클러스터 테이블의 공간 소요량 분석

LookAhead의 크기를 4로 고정시키고 압축하지 않았을 경우의 클러스터 테이블 인덱스의 크기는 약 1.04GB이다. 이는 단백질 테이블(72.12MB) 내의 ID와 2차 구조 정보인 Secondary 컬럼의 크기가 34.38MB일 때를 고려한다면 약 31.02배 정도이다.

본 논문에서는 클러스터 테이블의 압축을 위해 단백질 시퀀스 내의 각 문자의 발생 빈도를 고려한 데이터 이진 표현에 의한 압축 방식을 제안하였다. 실험 데이터 내의 각 문자의 발생 빈도(<표 9> 참고)를 이용하여, 클러스터 테이블을 압축한 실험 결과에 따르면, 단백질 테이블 34.38MB에 대해 약 22.38배 정도의 클러스터 테이블이 생성되었다. 이는 압축을 하지 않은 경우에 비해 1.39배 정도의 저장 공간 감소 효과를 나타낸다.

<표 9> 시퀀스 내의 각 문자 출현 빈도

문자	발생 빈도	발생률	이진표현
l	14,200,132	0.49	0
h	11,356,220	0.39	10
e	3,587,602	0.12	11

6. 질의처리 방안

본 장에서는 제 3장에서 제안한 클러스터 세그먼트 인덱스 기반의 질의처리 방안을 제안한다. 제안된 질의처리 방안에서는 우선 질의를 세그먼트 단위로 자른

후, 이를 2^k ($0 \leq k \leq 7$, k 는 정수) 클러스터 단위로 그룹핑하여 질의를 처리한다.

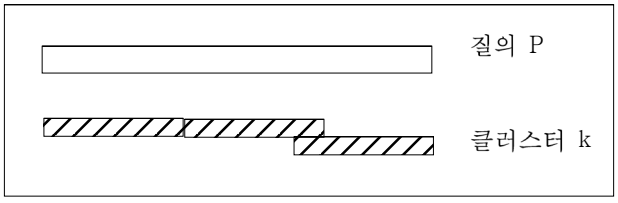
6.1 질의처리 과정

질의 내 세그먼트 개수가 |p|인 질의 p가 주어지면, 질의처리 과정은 다음과 같다.

(1) *k를 계산한다.* 질의를 구성하는 각 세그먼트를 2^k 크기의 클러스터 단위로 그룹핑하기 위해 k를 다음 식을 사용하여 계산한다.

$$k = \min(\lfloor \log2|p| \rfloor, 7)$$

(2) *클러스터 단위로 그룹핑한다.* k가 결정되면, <그림 6>와 같이 질의의 세그먼트를 2^k 크기의 클러스터 단위로 중첩되지 않게 그룹핑해 나간다. 이때, 가장 마지막 클러스터는 질의의 끝에서 2^k 크기만큼을 그룹핑해서 생성한다. 이 과정에서 질의의 마지막 클러스터와 그 직전의 클러스터는 중첩되어 질 수 있다.



<그림 6> 클러스터를 이용한 중첩 검색

(3) *클러스터 단위별로 질의를 처리하여 얻어진 중간 결과를 조인한다.* (2)에서 얻어진 클러스터를 클러스터 세그먼트 인덱스를 이용하여 검색한 후, 그 중간 결과를 클러스터 내의 세그먼트 길이(TypeLen)와 시작위치(Loc) 정보를 기준으로 조인한다.

(4) *후처리 과정을 수행한다.* TypeStr, TypeLen, LookAhead가 모두 일치하더라도 TypeLen는 각 문자의 발생 빈도를 모두 합한 값이므로, 이로 인한 착오 채택(false alarm)이 발생한다. 따라서 (3)의 과정에서 얻어진 시퀀스를 실제의 단백질 테이블에서 가져와 질의와 정확히 일치하는지 확인하는 후처리 단계가 필요하다.

6.2 인덱스 계속 사용 여부 선택

본 논문에서는 검색 효율을 보다 향상시키기 위한 방안으로, 인덱스의 정선도를 고려하여 검색 과정에서 인덱스의 계속 사용 여부를 결정하는 기법을 제안한다. 인덱스 사용 여부를 결정하기 위한 함수 CONT_INDEX(K, L)는 다음과 같다.

$$\text{CONT\_INDEX}(K,L) = (K < 3) \text{ AND } (L < 8)$$

이는 제 4장에서 보인 바와 같이 k < 3인 경우와 LookAhead가 8 미만인 경우, 인덱스의 정선도가 낮아져 인덱스 사용에 따른 검색 효율 향상 효과를 기대할 수 없기 때문이다.

6.3 질의처리 방법

6.3.1 Exact Match

질의 형태 중 가장 간단한 형태로, 예를 들어 질의 <e 3 3><h 2 2><l 1 1>의 경우, 클러스터 <e 3 3><h 2 2>와 클러스터 <h 2 2><l 1 1>로 나누어 검색한 뒤, 앞쪽 클러스터의 시작 위치와 뒤쪽 클러스터의 시작 위치가 5(=3+2)만큼의 차를 갖는 튜플들을 조인하여, 결과로 시퀀스 ID와 Loc를 결과로 얻는다.

6.3.2 Range Match

TypeLen은 각 문자의 발생 빈도를 합한 값이므로 영역 질의 처리 시, TypeLen의 범위가 넓어지는 현상이 발생한다. 예를 들어, 질의 <e 3 5><h 3 6><l 3 7><e 3 8> 는 TypeStr=ehle, TypeLen=12(=3+3+3+3)~26(=5+6+7+8)이 되어 TypeLen 범위의 증가에 따른 인덱스의 검색 공간이 확대되어 검색 효율이 저하되는 요인이 된다. 따라서, 본 논문에서는 이러한 문제점을 해결하기 위한 방안으로 선택적 클러스터링 범위 검색 기법(SCRM: Selective Clustering Range Match)을 제안한다.

제안된 기법에서는 범위 검색을 위해 우선, 주어진 질의를 2^k의 크기를 갖는 클러스터 단위로 분할 한 후, 해당 클러스터를 검색하는 과정에서 제 3.3절에서 제안한 TypeLen 히스토그램 테이블을 이용한다. 즉, 분할된 2^k의 클러스터를 2^k, 2^{k-1}, 2^{k-2}, ..., 2^0의 클러스터로 세분화하여 검색을 수행했을 경우, 결과로 얻어질 수 있는 튜플의 수를 TypeLen히스토그램 테이블을 이용하여 미리 사전에 예측할 수 있다.

예를 들어, 질의 <e 50 60><l 3 3><h 5 6><e 6 8>를 검색하기 위하여 다음과 같이 k=0,1,2의 각각의 경우에 대해 해당되는 히스토그램을 비교한다.

① <e 50 60><l 3 3><h 5 6><e 6 8>
: k=2, TypeLen=64~77
② <e 50 60><l 3 3>
: k=1, TypeLen=53~63, LookAhead=he
③ <e 50 60>
: k=0, TypeLen=50~60, LookAhead=lhe

위의 3가지 경우 중 히스토그램이 나타내는 튜플의 수가 가장 적은 경우를 사용하여 질의를 처리한다.

그러나 TypeLen만을 사용하여 반환되는 튜플의 수를 예측하는 경우, TypeStr의 길이가 기하급수적으로 짧아져 검색 성능을 저하시킬 수 있다는 문제점이 있다. 위의 예제에서 만약 k=1인 클러스터를 사용하여 검색을 수행하는 경우, TypeStr='elhe' 대신 'el'로 검색하기 때문에 검색 과정에서 효율이 저하될 수 있다. 본 논문에서는 이러한 문제점을 해결하기 위해, 제 3장에서 제안한 클러스터 테이블에서의 LookAhead를 다음과 같이 확장, 적용한다.

k < 3인 경우 LookAhead의 값을 2^3-2^k, k ≧ 3인 경우는 LookAhead의 값을 8로 사용한다. 이는 4장에서의 TypeStr의 길이가 2^3 이상인 경우와 LookAhead의 값이 8이상인 경우 인덱스의 정선도가 좋다는 실험 결과에 의한 것이다.

6.3.3 Wildcard Match

본 논문에서는 Wildcard Match를 Range Match로 변환하여 처리한다. 변환 시, 다음과 같은 사항을 고려하여 질의 형태를 변환한다.

(1) 질의의 중간에 Wildcard가 있는 경우

<a la ua><? lw uw><b lb ub>와 같은 질의 형태로 이는 Wildcard가 a 세그먼트로 시작하거나 b 세그먼트로 끝날 수 있음을 의미하므로, 질의를 <a la ua+uw>와 <b lb ub+uw>로 나누어 검색한 후, 그 결과를 조인한다. 단, 조인 시 Wildcard 전후의 a 세그먼트와 b 세그먼트의 총 길이와 ua+ub의 차는 uw이하가 되어야 하며 두 세그먼트의 간격은 최소 lw가 되어야 한다.

(2) 질의의 처음에 Wildcard가 있는 경우

<? lw uw><b lb ub>와 같은 질의 형태로, Wildcard가 b 세그먼트로 끝날 수 있으므로 <b lb ub+uw>로 검색한다. 단, 후처리 과정에서 b 세그먼트 앞에 최소 lw개의 문자가 존재하는지를 확인해야 한다.

(3) 질의의 끝에 Wildcard가 있는 경우

<a la ua><? lw uw>와 같은 질의 형태로, Wildcard가 a문자로 시작할 수 있으므로, <a la ua+uw>로 검색을 수행한 후, 후처리 과정에서 a 세그먼트 뒤에 최소 lw개의 세그먼트가 존재하는지를 확인해야 한다.

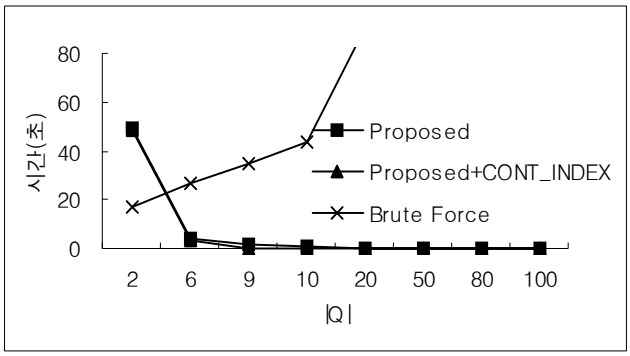
7. 실험

본 장에서는 성능 분석을 위하여 수행한 실험 결과를 보인다. 실험 데이터로는 5절에서 언급한 83,072개의 단백질 2차 시퀀스를 사용하였으며, 실험은 Microsoft Windows XP, RAM 512, HDD 80GB 7200rpm, Oracle 8i에서 수행하였다.

7.1 Exact Match

Exact Match는 주어진 질의와 정확하게 일치하는 시퀀스를 검색하는 연산으로, <그림 7>에 본 논문에서 제안된 Exact Match 알고리즘과 해당 알고리즘에 CONT_INDEX 연산자를 적용한 방식, Brute Force방식을 비교 실험한 결과를 보인다. Brute Force방식은 질의내의 모든 세그먼트를 세그먼트 테이블에서 순차적으로 검색하는 방식이다. |Q|는 질의 내 세그먼트의 개수이며, 알고리즘의 성능 비교 기준은 질의 처리 시간으로 하였다.

실험 결과에 따르면, |Q|=2인 경우에는 Brute Force 방식이 더 좋은 성능을 보였다. 이는 5절에서 언급한바와 같이 제안된 인덱스는 K=0,1인 경우 정선도가 좋지 않기 때문이다. 그러나 |Q|가 커질수록 Brute Force의 질의 처리 시간은 선형적으로 증가하는데 반해, 제안된 방식은 점차 질의 처리 시간이 감소하였으며, 특히 |Q|≥9 인 경우 좋은 검색 성능을 보였다.



<그림 7> 질의 길이에 따른 질의처리 시간

7.2 Range Match와 Wildcard Match

Range Match와 Wildcard Match는 현재 구현 및 실험 중인 단계로, Range Match를 위한 실험으로 TypeLen의 범위 확대에 따라 제안된 SCRM 알고리즘의 검색 효율을 비교 실험할 계획이다. Wildcard Match의 경우에는 본 논문에서 제안한 질의 언어와 [6]에서 제안한 질의 언어의 검색 능력을 비교하기 위해 <a a_lb a_ub><? 0 n><b b_lb b_ub>의 질의에 대해 n 값을 0~∞로 바꾸면서 검색되는 시퀀스 개수를 비교 실험할 예정으로, 이 실험을 통해 갭(Gap)을 문자 단위로 지정하는 본 논문의 질의 언어가 [6]의 갭(Gap)을 세그먼트 단위로만 지정하는 방법에 비해 더 효율적임을 보일 수 있다.

8. 결론

본 논문에서는 단백질 2차 시퀀스를 세그먼트 단위로 나누고, 세그먼트들을 클러스터로 묶은 클러스터 세그먼트 테이블의 개념과, 이를 사용한 Exact Match, Range Match, Wildcard Match 알고리즘을 제안하였다. 또한, Look Ahead를 사용한 검색 성능 향상 기법을 제안 및 적용하였다.

또한, 본 논문에서 제안한 질의 언어는 PROSITE 데이터베이스 (http://www.expasy.org/prosite)등 에서 제공하는 정규식 형태의 모티프(motif)[5]에 대한 검색을 지원할 수 있으며, 특히 검색 대상이 단백질 2차 구조라는 점에 그 의미가 있다.

향후 연구에서는 각종 정렬(alignment) 알고리즘에 대한 비교 분석을 통해, 단백질 2차 구조에 대한 인덱스 기반의 유사 검색 기법을 제안하고, 시퀀스 데이터의 급격한 증가를 수용할 수 있는 효과적인 인덱싱 기법에 관한 연구를 수행할 예정이다.

참고 문헌

[1] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson. Molecular Biology of the Cell, 3rd ed. Garland Publishing, Inc., 1994

[2] A. Califano and I. Rigoutsos, "FLASH: A Fast Look-up Algorithm for String Homology," Proc. Int'l Conf. Intelligent Systems for Molecular biology, pp. 56-64, 1993

[3] D. Frishman and P. Argos, "75% accuracy in protein secondary structure prediction", in Proteins, 27:329-335, 1997

[4] D. Frishman and P. Argos, "Incorporation of long-Distance interactions into a secondary structure prediction algorithm", in Protein Engineering, 9:133-142, 1996

[5] A. Gattiker, E. Gasteiger and A. Bairoch, "ScanProsite: a reference implementation of a PROSITE scanning tool," Applied Bioinformatics, 1(2):107-108, 2002

[6] L. Hammel and J. M. Patel, "Searching on the Secondary Structure of Protein Sequence," In Proc. VLDB Conference, 2002

[7] E. Hunt, M. P. Atkinson and R. W. Irving, "Database indexing for large DNA and protein sequence collections", The VLDB Journal, Vol. 11, No. 3, pp. 256-271, 2002

[8] T. Kahveci and A. K. Singh, "An Efficient Index Structure for String Databases", in VLDB, 2001.

[9] H.E. Williams, ''Genomic Information Retrieval'' , In K.-D. Scheme and X. Zhou, Proc. Australasian Database Conference, Adelaide, Australia, 27-35, 2003.

[10] H. E. Williams, J. Zobel, "Indexing and Retrieval for Genomic Databases," IEEE Transactions on Knowledge and Data Engineering, Vol. 14 ,1,pp. 63-78, 2002

[11] I. H. Witten , T. C. Bell , A. Moffat, "Managing Gigabytes: Compressing and Indexing Documents and Images," John Wiley & Sons, Inc., New York, NY, 1994

[12] H. Wang, C.-S. Perng, W. Fan, S. Park, and P. S. Yu, "Indexing Weighted Sequences in Large Databases," in Proc. 19th IEEE International Conference on Data Engineering (IEEE ICDE), pp. 63-74, Bangalore, India, March, 2003

[13] C. H. Wu, L-S. L. Yeh, H. Huang, L.Arminski, J. Castro-Alvear, Y. Chen, Z-Z. Hu, Robert S. L., P. Kourtesis, B. E. Suzek, C. R. Vinayaka, J. Zhang, and Winona C. Barker. "The Protein Information Resource", in Nucleic Acids Research, 31: 345-347, 2003.

[14] C. Fondrat and P. Dessen, "A Rapid Access Motif Database(RAMdb) with A Search Algorithm for the Retrieval Patterns in Nucleic Acids or Protein Databanks," Computer Applications in the Biosciences, 11(3): 273-279, 1995.

[15] G. A. Stephen, String Searching Algorithms, World Scientific Publishing, 1994.