염기문자의 빈도와 위치정보를 이용한 DNA 인덱스구조

(A DNA Index Structure using Frequency and Position Information of Genetic Alphabet)

김 우 철 † 박 상 현 †† 원 정 임 ††† 김 상 욱 †††† 윤 지 희 †††††
(Woo-Cheol Kim) (Sang-Hyun Park) (Jung-Im Won) (Sang-Wook Kim) (Jee-Hee Yoon)

요 약 대규모 DNA 데이타베이스를 대상으로 원하는 서열을 빠르게 검색하기 위해 인덱싱 기법을 많이 사용하고 있다. 그러나 대부분의 인덱싱 기법은 원래 데이타베이스보다 더 큰 저장공간을 사용하고 DBMS와의 밀 결합이 어렵다는 문제점을 가지고 있다. 본 논문에서는 완전 매치, 와일드카드 매치, k-미스매치와 같은 근사 매치 질의 처리를 위해 작은 공간을 사용하는 디스크 기반의 효율적인 인덱싱 기법과 질의 처리 기법을 제안한다. 인덱싱을 위해서 DNA 염기서열에 일정 크기의 슬라이딩 윈도우를 위치시킨 후, 윈도우 내에서 각 문자의 출현 빈도를 이용해 서명을 추출해서 R*-트리와 같은 다차원 공간 인덱스에 저장한다. 특히 윈도우 내의 각 위치에 따라서 가중치를 줌으로써 서명들이 인덱스 공간에 집중되는 현상을 억제한다. 제안된 질의 처리방법은 질의 시퀀스를 다차원 사각형으로 변환하고 그 사각형과 중첩되는 서명들을 인덱스로부터 찾아낸다. 제안된 방법을 실제 생물학자들이 사용하는 데이타를 이용해 실험한 결과 서픽스 트리 기반의 방법에 비해서 완전 매치인 경우 3배 이상, 와일드카드 매치인 경우 2배 이상, k-미스매치인 경우 수십 배 이상의 성능향상을 보였다.

키워드 : DNA 데이타베이스, 인덱싱, 완전 매치, 와일드카드 매치, K-미스매치

Abstract In a large DNA database, indexing techniques are widely used for rapid approximate sequence searching. However, most indexing techniques require a space larger than original databases, and also suffer from difficulties in seamless integration with DBMS. In this paper, we suggest a space-efficient and disk-based indexing and query processing algorithm for approximate DNA sequence searching, specially exact match queries, wildcard match queries, and k-mismatch queries. Our indexing method places a sliding window at every possible location of a DNA sequence and extracts its signature by considering the occurrence frequency of each nucleotide. It then stores a set of signatures using a multi-dimensional index, such as R*-tree. Especially, by assigning a weight to each position of a window, it prevents signatures from being concentrated around a few spots in index space. Our query processing algorithm converts a query sequence into a multi-dimensional rectangle and searches the index for the signatures overlapped with the rectangle. The experiments with real biological data sets revealed that the proposed method is at least three times, twice, and several orders of magnitude faster than the suffix-tree-based method in exact match, wildcard match, and k-mismatch, respectively.

Key words : DNA database, Indexing, Exact match, Wildcard match, K-mismatch

· 본 연구는 2003년도 연세대학교 신진교수지원과제(2003-1-0361)과 2004년도 학술진흥재단의 신진교수 연구과제 지원(KRF-2004-003-D00302), 2003년도 한국학술진흥재단의 선도과학자 연구비 지원(KRF-2003-041-D00486), 제주대학교 IT 연구 센터(텔리매틱스 요소 기술 연구)를 통한 연구비 지원을 받았습니다.

† 비 회 원 : 연세대학교 컴퓨터과학과
twelvepp@cs.yonsei.ac.kr
†† 종신회원 : 연세대학교 컴퓨터과학과 교수
sanghyun@cs.yonsei.ac.kr
††† 정 회 원 : 연세대학교 컴퓨터과학과 교수
jiwon@@cs.yonsei.ac.kr
†††† 종신회원 : 한양대학교 정보통신학부 교수
wook@hanyang.ac.kr
††††† 종신회원 : 한림대학교 정보통신공학부 교수
jhyoon@hallym.ac.kr

논문접수 : 2004년 7월 20일
심사완료 : 2005년 2월 15일

1. 서 론

DNA 서열(DNA sequence)은 생물의 특성을 결정하는 코드로서 A, C, G, T의 네 가지 문자들로 구성된 리스트로 표현된다. 새롭게 발견된 DNA 서열의 생물학적 특성을 파악하기 위한 기본적인 방법은 그것과 염기의 배열이 유사한 다른 DNA 서열의 특성으로부터 유추하는 것이다[1]. 염기 서열 검색 연산은 이미 특성이 판명된 DNA 서열 데이타베이스로부터 주어진 질의 DNA 서열과 염기 배치가 유사한 DNA 서열들을 검색하는 연산이다. 분자 생물학자들은 이 연산을 통하여 새롭게 발견된 DNA 서열의 역할, 진화 과정, 화학적 구조 등을 유추한다. DNA 서열에는 돌연 변이의 발생과 오염의 발생이 가능하므로 염기 서열 검색 연산으로서 완전 매치 질의(exact match query)보다는 근사 매치 질의(approximate match query)가 보다 널리 사용된다[2].

염기 서열 검색 연산의 기초적인 방식은 주어진 두 서열들 간의 최적의 정합을 찾아주는 스미스-워터만 정합 알고리즘(Smith-Waterman alignment algorithm)[3]을 이용하는 것이다. 이 알고리즘에서 사용하는 두 서열 간의 유사성에 대한 모델은 다수의 생물학자들에 의하여 채택됨으로써 이후의 연구에 큰 영향을 미친 바 있다. 그러나 이 알고리즘은 길이가 각각 n과 m인 두 서열들의 정합(alignment)을 계산하는데 O(mn)의 시간을 요구하므로, 서열 비교를 위한 수행 시간이 비교적 오래 걸린다. 또한, 검색 시 전체 데이타 서열을 디스크로부터 액세스해야 하는 단점이 있다.

스미스-워터만 정합 알고리즘을 이용한 방식의 이러한 단점은 전체 데이타 서열들을 질의 서열과 비교한다는 점에서 비롯된 것이다. 이러한 단점을 해결하는 가장 일반적인 아이디어는 여과 및 정제(filtering and refinement) 방식을 채택하는 것이다. 즉, 여과 단계에서 질의 서열과 매치될 가능성이 높은 후보들만을 빠르게 검색한 후, 정제 단계에서 이들 각각에 대하여 실제로 질의 서열과 매치되는지의 여부를 확인하는 것이다. BLAST[4,5]는 이러한 방식을 채택하는 대표적인 기법이다. BLAST는 스미스-워터만 정합 알고리즘이 제시하는 유사 모델을 완벽하게 지원하지는 않는 휴리스틱 방식을 기반으로 하고 있으나, 검색의 성능을 극대화함으로써 현재 생물학자들에 의하여 널리 사용되고 있다.

최근, 참고 문헌 [6]에서는 k-디퍼런스 질의를 처리하기 위해 MR-Index를 제안하였다. k-디퍼런스 질의(k-difference query)는 대치, 삽입, 삭제 등을 k회 이하로 적용함으로써 질의 서열과 매치되는 데이타 서열들의 집합을 찾는 연산이다. 이 기법에서는 먼저 2^n 형태로 표현되는 다양한 길이의 슬라이딩 윈도우(sliding window)들을 데이타 서열로부터 추출한 후, 각 슬라이딩 윈도우에 대하여 각 염기 문자의 출현 빈도를 표현하는 서명(signature)을 작성한다. 다음으로, 동일한 길이의 슬라이딩 윈도우로부터 작성된 서명들을 모두 모아서 하나의 R-트리[7]에 저장한다. k-디퍼런스 질의는 이렇게 구축된 여러 개의 R-트리들을 이용하여 처리된다.

본 논문에서는 염기 서열 검색 연산들 중 완전 매치 질의(exact match query), 와일드카드 매치 질의(wild-card match query), k-미스매치 질의(k-mismatch query)의 효과적인 처리 방안에 관하여 논의하고자 한다. 이들은 전술한 스미스-워터만 정합 알고리즘을 기반으로 하는 근사 매치 질의와 더불어 생물학자들에 의하여 널리 사용된다. 이러한 질의들의 처리를 위하여 먼저 스미스-워터만 정합 알고리즘을 기반으로 하는 염기 서열 검색 기법과 MR-Index를 이용한 검색 기법 등을 고려할 수 있다. 그러나 이러한 기법들이 처리하려는 질의들은 본 논문에서 처리하려는 질의들과 비교하여 검색하고자 하는 대상 자체가 다르다. 즉, 이러한 기법들을 사용하는 경우, 본 논문에서 질의 서열과 유사하다고 판단되는 데이타 부분 서열들보다 훨씬 많은 수의 서열들이 질의 결과로 반환되게 된다.

완전 매치 질의를 처리하기 위한 방안으로는 보이어-무어 알고리즘(Boyer-Moore algorithm)[8]과 KMP 알고리즘(Knuth-Morris-Pratt algorithm)[9] 등이 제안된 바 있으며, 와일드카드 매치 질의를 처리 위한 방안으로는 아호-코라식 알고리즘(Aho-Corasick algorithm)[10]과 스캔 벡터(scan vector)를 이용한 기법이 제안된 바 있다. 이들은 주어진 두 서열들 간의 매치 여부를 효과적으로 판단하기 위하여 CPU 성능을 최적화하는데 목표를 두었다. 반면, 데이타 서열 내의 모든 부분 서열들을 질의 서열과 비교해야 하므로 전체 검색 시간은 크게 증가하며, 또한 순차 검색을 기반으로 하므로 전체 데이타 서열을 디스크로부터 읽어야 한다는 성능 상의 문제점이 있다. 접미어 트리를 이용한 기법(suffix-tree-based method)[11]은 인덱스 구조의 하나인 접미어 트리를 사용함으로써 질의 서열과 매치될 가능성이 높은 후보들을 검색한 후, 이들 각각에 대해서만 실제로 질의 서열과 매치되는지의 여부를 확인한다. 이 기법은 완전 매치 질의, 와일드카드 매치 질의, k-미스매치 질의 등을 모두 처리할 수 있으며, 최종 결과에 포함될 가능성이 높은 후보들만을 디스크로부터 액세스하여, 질의 서열과 비교하게 되므로 상대적으로 좋은 성능을 얻을 수 있다. 반면, 이 기법은 접미어 트리 고유의 특성상 트리를 위한 저장 공간의 오버헤드가 크며[12], 이러한 큰 접미어 트리로 인하여 트리 탐색 시간이 크다는 단점을 갖는다.

본 논문에서는 이러한 문제점들을 해결하는 새로운 근사 질의 처리 기법을 제안한다. 먼저, DNA 데이타베이스를 인덱싱하는 새로운 기법을 제안한다. 제안하는 기법은 데이타 염기 서열로부터 일정 크기를 갖는 슬라이딩 윈도우들을 추출한 후, 각 윈도우 내에서 각 염기의 출현 빈도를 파악하여 구성한 서명을 다차원 공간 인덱스의 하나인 R*-트리[13]에 저장한다. 이때, 각 문자의 출현 위치에 가중치를 부여하여 윈도우들과 대응되는 서명들이 다차원 공간상에 집중되는 것을 억제한다. 다음으로, 제안된 인덱스를 이용하여 완전 매치 질의, 와일드카드 매치 질의, k-미스매치 질의를 효과적으로 처리할 수 있는 질의 처리 기법을 제안한다. 질의 처리 시에는 질의 서열을 다차원 공간내의 직사각형으로 변환한 후, 인덱싱 단계에서 구성된 R*-트리 검색을 통하여 이 직사각형과 중첩된 서명들을 파악함으로써 후보들을 효과적으로 파악할 수 있다.

제안한 기법의 우수성을 검증하기 위하여 기존의 접미어 트리를 이용한 기법과의 다양한 실험을 통한 성능 비교를 수행한다. 실험 결과에 따르면, 제안된 기법은 인덱싱을 위하여 접미어 트리의 1%에 해당하는 저장 공간만을 사용하는 경우 완전 매치 질의인 경우에 3배 이상, 와일드카드 매치인 경우에 2배 이상, k-미스매치인 경우에 수십 배 이상의 성능 향상을 보이는 것으로 나타났다.

2. 문제 정의

본 장에서는 먼저 논의 전개에 필요한 용어 및 기호를 정의하고, 본 논문에서 해결하고자 하는 문제 또한 정의한다.

먼저 염기 서열 T는 T의 i-번째 염기를 표현하는 문자인 t_i ($1 \le i \le n$)들의 리스트이며, $T = <t_1, t_2, \ldots, t_n>$와 같이 정의된다. T내에 포함된 문자의 수 n을 |T|로 표기하며, T의 길이라 부른다. 데이타베이스 내에 저장된 염기 서열을 데이타 염기 서열이라 하며, 질의에 명시되는 염기 서열을 질의 염기 서열이라 한다. 또한, T의 부분 서열 T′은 T로부터 임의의 수의 연속된 문자들을 추출함으로써 만들어진 서열이다.

알파벳 Σ는 염기 서열 내에서 발생 가능한 문자들의 집합으로서 표 1과 같은 문자를 포함한다. 여기서, A, C, G, T는 생명체에서 나타나는 실제 코드와 일대일 대응된다. 그 외의 문자들은 해당 부분의 생명 특성이 완전히 밝혀지지 않아 A, C, G, T중 가능성이 있는 두 가지 이상의 코드들을 표현하기 위하여 사용된다. 표 1은 이러한 각 문자가 표현하는 코드들을 나타낸 것이다.

데이타베이스 내에 저장된 염기 서열들과 질의 내에서 제시되는 염기 서열들은 그 길이가 서로 다르다. 따라서 효과적인 인덱싱을 위하여 윈도우라는 개념을 사용한다. 윈도우는 인덱싱의 단위로서 염기 서열로부터 추출된 고정 길이를 갖는 부분 서열로 정의된다. 윈도우는 W로 표기하며, W의 길이를 |W|로 표기한다. 또한, 염기 서열 T의 i-번째 문자로부터 추출된 윈도우를 W_i로 표기한다.

표 1 문자 코드 표

문자	표현 특성문자	연상기호
A	A	A-denine
C	C	C-ytosine
G	G	G-uanine
T(or U)	T	T-hymine (or U-racil)
R	A or G	pu-R-ine
Y	C or T	p-Y-rimidine
S	G or C	S-tring(3 H-bonds)
W	A or T	W-eak(2 H-bonds)
K	G or T	K-eto
M	A or C	a-M-ino
B	C or G or T	not-A
D	A or G or T	not-C
H	A or C or T	not-G
V	A or C or G	not-(T or U)
N(or X)	any base	a-N-y(or Unknown)

두 문자 s와 q가 존재할 때, s가 표현하는 문자 집합과 q가 표현하는 문자 집합의 교집합이 공집합이 아니면, s와 q는 서로 매치한다고 정의한다. 예를 들어, s=A이고 q=N인 경우, s는 {A}를 표현하고, q는 {A, C, G, T}를 표현하므로, $s \cap q = \{A\}$이다. 따라서 이 경우 s와 q는 서로 매치한다. 이러한 매치에 대한 정의를 통해서 질의 염기 서열 Q와 부분 서열 T′가 주어질 때, |Q|과 |T′|이 같은 경우 Q와 T′간의 다음과 같은 세 가지 매치 관계가 정의된다. (1) 완전 매치(exact match)는 알파벳 Σ로 구성된 T′와 Q에 대해서 모든 i ($1 \le i \le |Q|$)에 대하여 대응되는 q_i와 t'_i의 쌍이 매치하는 경우이다. (2) 와일드카드 매치(wildcard match)는 알파벳 Σ로 구성된 T′과 알파벳 Σ와 와일드 카드 문자 *로 구성된 Q에 대하여 모든 i ($1 \le i \le |Q|$)에 대하여 대응되는 q_i와 t'_i의 쌍이 매치하는 경우이다. 단 여기서 와일드카드 문자 *는 알파벳 Σ내의 어떤 문자와도 매치되는 특수한 문자로 간주된다. (3) k-미스매치(k-mismatch)는 알파벳 Σ로 구성된 T′과 알파벳 Σ와 와일드 카드 문자 *로 구성된 Q에 대하여 모든 i ($1 \le i \le |Q|$)에 대하여 대응되는 q_i와 t'_i의 쌍들 중 |Q|-k개 이상이 매치하는 경우를 의미한다.

염기 서열 검색 문제는 위와 같은 정의를 통해서 데이타베이스에 저장된 염기 서열 T로부터 질의 염기 서열 Q와 매치되는 T의 모든 부분 서열 T′의 위치들을

찾는 문제로 정의할 수 있다. 이 정의는 염기 서열 검색 문제를 데이타베이스에 저장된 하나의 염기 서열을 대상으로 정의하였다. 그러나 실제 응용에서는 데이타베이스 내에 저장된 다수의 염기 서열들을 대상으로 검색을 요구할 수 있다. 이 경우에는 (1) 모든 염기 서열들을 하나의 긴 염기 서열로 연결한 후 염기 서열 검색을 수행하거나, (2) 각 염기 서열에 대하여 독립적으로 염기 서열 검색을 수행한 후 전체 결과들을 병합하는 것이 가능하다. 따라서 본 논문에서는 이후부터 이러한 다수의 염기 서열들을 대상으로 하는 검색에 대해서는 별도로 논의하지 않는다.

3. 관련 연구

본 장에서는 본 연구에서 다루고자 하는 완전 매치 질의, 와일드카드 매치 질의, k-미스매치 질의가 사용되는 각 응용 분야를 소개하고, 이러한 질의를 처리하는 기존의 기법들의 특성과 장단점에 관하여 논의한다.

3.1 완전 매치 질의

완전 매치 질의의 응용 분야로 EST의 검색[14]을 들 수 있다. EST(expressed sequence tag)는 평균 길이 200~300의 DNA 서열이며, 하나의 유전자 서열(gene sequence) 내에서 단 한번만 발생한다. 분자생물학의 연구 결과로 이미 수백만 개의 EST들이 발견되어 현재 데이타베이스 내에 저장되어 있다. 따라서 새로운 DNA 서열이 발견되었을 때, 이 서열 내에 과연 어떠한 EST가 포함되어 있는가를 파악하는 것은 매우 중요하다. 이를 위하여 필요한 연산이 바로 주어진 집합 내의 각 EST를 대상으로 수행되는 완전 매치 질의이다. 이를 위한 기존의 방법으로 보이어-무어 알고리즘(Boyer-Moore algorithm)[8]과 KMP 알고리즘(Knuth-Morris-Pratt algorithm)[9]과 같은 하나의 데이타 서열 내에서 질의 서열의 발생 위치들을 효과적으로 파악하는 알고리즘들이 사용되었다. 비교 대상이 되는 서열의 길이를 n, 질의 서열의 길이를 m이라 할 때, 최악의 경우에도 O(n+m)의 시간 내에 완전 매치 질의를 처리할 수 있다. 단, 이 두 기법들은 순차 검색을 기본 방식을 사용하므로 전체 서열을 디스크로부터 읽어 들여야 하며, 또한 데이타 서열 내의 모든 부분 서열들을 질의 서열과 비교해야 한다는 성능 상의 문제점을 갖는다.

3.2 와일드카드 매치 질의

DNA 전사 요소(DNA transcription factor)의 검색[14]은 와일드카드 매치 질의가 필요한 대표적인 문제이다. DNA 전사 요소는 DNA가 RNA로 전사될 때 영향을 주는 단백질이다. 최근의 연구들로 인하여 많은 DNA 전사 요소들이 발견되었으며, 이들은 여러 개의 그룹들로 분류된다. 각 그룹 내의 다수의 DNA 전사 요소들은 와일드카드들을 포함하는 하나의 서열로 표현된다. 따라서 새로운 서열이 발견되었을 때, 이 서열이 어떠한 DNA 전사 요소들을 포함하는지 판단하는 연산은 이 서열의 특성을 유추할 수 있는 좋은 수단이 된다. 이러한 질의를 처리하기 위해 아호-코라식 알고리즘(Aho-Corasick algorithm)[10]과 스캔 벡터(scan vector)를 이용한 기법이 사용된다[14]. 이 기법은 먼저 질의 서열에서 와일드카드를 제외했을 때 생기는 나머지 부분 패턴들을 하나의 집합으로 설정한다. 그런 다음, 데이타 서열로부터 이 집합 내의 부분 패턴들과 일치하는 모든 위치들을 찾는다. 그 다음, 스캔 벡터라 부르는 일차원 배열을 이용하여 발생한 모든 부분 패턴들을 포함하는 부분 서열들을 데이타 서열로부터 찾아낸다. 이 부분 서열들이 바로 질의 서열과 와일드카드 매치되는 것들이다. 이 기법은 서열 길이에 해당되는 크기의 스캔 벡터를 사용해야 하므로 저장 공간의 오버헤드가 크다. 또한, 기본적으로 순차 검색 방식을 사용하므로 전체 서열을 디스크로부터 읽어야 하는 오버헤드가 있다.

3.3 k-미스매처 질의

앞서서 완전 매치 질의에 대한 응용으로서 EST의 검색을 보였다. 그러나 일반적으로 새롭게 발견된 DNA 서열은 오염으로 인하여 변형된 염기를 포함하고 있는 경우가 많다. 따라서 이와 같은 변형된 부분까지 고려하는 것이 바람직한 경우에는 완전 매치 질의 보다는 k-미스매치 질의를 사용하게 된다. k-미스매치 질의를 이용함으로써 서열 내에서 EST 내의 요소들 중 k개까지의 변형을 허용할 수 있게 된다. 기존의 방법으로 접미어 트리를 이용한 기법(suffix-tree-based method)[11]을 들 수 있다. 이 방법은 먼저 질의 서열과 데이타 서열을 대상으로 하나의 접미어 트리를 구성한다. 그 다음, 접미어 트리 내에서 두 서열들의 공통 조상 노드들 중 가장 하위에 존재하는 노드를 찾은 후, 그 노드로부터 k개의 불일치를 찾을 때까지 트리를 검색해 나가는 방식을 사용한다. 이 기법은 순차 검색 방식이 아닌 여과 및 정제 방식이므로 앞서 언급한 다른 기법들과 비교하여 좋은 검색 성능을 제공할 수 있다. 또한, 접미어 트리를 이용한 기법은 앞서 언급한 완전 매치 질의 및 와일드카드 매치 질의의 처리에도 유사한 방식으로 적용될 수 있다. 그러나 이 기법은 접미어 트리의 특성상 트리를 위한 저장 공간의 오버헤드가 크며, 이러한 큰 접미어 트리로 인하여 트리 탐색 시간이 크다는 단점을 갖는다.

4. BSI(Basic Signature Index)

본 장에서는 대규모 염기 서열 데이타베이스를 대상으로 완전 매치, 와일드카드 매치, k-미스매치 문제를

효과적으로 지원하는 BSI(Basic Signature Index) 기법을 제안하고, 이를 이용한 질의 처리 방식을 보인다.

4.1 기본 전략

먼저, 데이타 염기 서열 T의 모든 문자 위치에 크기가 |W|인 슬라이딩 윈도우를 위치시키고 '데이타 윈도우'라고 정의한다. 다음, 각 데이타 윈도우에 대하여 윈도우 내에 출현하는 n개의 특성 문자에 대하여 출현 빈도의 최소 값과 최대 값을 고려하여 각 윈도우를 n차원 공간상의 직사각형으로 표현한다. 본 연구에서는 특성 문자로 A, C, G, T의 4개 문자를 사용한다. 윈도우에 출현하는 특성 문자의 출현 빈도를 최소 값과 최대 값으로 표현하는 이유는 다음과 같다.

염기 서열 T 상에는 A, C, G, T의 4가지 문자 외에 B, N 등 그 특성이 정확히 밝혀지지 않은 문자가 출현할 수 있다. 예를 들어 문자 B는 A가 아닌 C, G, T 중 임의의 하나의 문자를 의미한다. 따라서 윈도우 상에 문자 B가 출현하는 경우, C, G, T의 최소 출현 빈도는 변화가 없으나, 최대 출현 빈도는 하나씩 증가한다. 데이타 염기 서열 T 내에서 추출된 슬라이딩 윈도우들의 수는 매우 많으므로, 이와 대응되는 4차원의 직사각형들은 R*-트리[13]와 X-트리[15] 같은 페이지 기반의 다차원 인덱스를 이용하여 저장 관리된다.

이러한 다차원 인덱스를 이용한 질의 처리 과정은 다음과 같다. 여기에서는 직관적인 이해를 위하여 |Q|=|W|의 경우를 가정하고, 4.3절에서 일반적인 경우를 보인다. 질의 염기 서열이 주어지면 질의 유형에 따라 각 특성 문자의 출현 빈도, 와일드카드 문자의 출현 빈도, 허용되는 최대 미스매치 수(=k)등을 고려하여 질의 염기 서열을 4차원 상의 직사각형으로 표현한다. 이 직사각형을 '질의 직사각형'이라 정의한다. 다음으로, 질의 직사각형과 중첩된 직사각형들을 다차원 인덱스로부터 검색한다. 이렇게 검색된 직사각형을 '후보 직사각형'이라고 정의한다. 최종적으로, 각 후보 직사각형이 표현하는 데이타 윈도우를 데이타 염기 서열로부터 직접 액세스해서 착오 채택(false alarm)[16,17]을 제거한다.

4.2 인덱스 구성

염기 서열 T 상의 각 데이타 윈도우를 기본 서명(Basic Signature: BS)으로 표현한다. BS는 윈도우 상에 출현하는 문자의 출현 빈도를 이용하여 윈도우의 특성을 나타내며, 그 정의는 다음과 같다.

정의 1. 윈도우 Wi의 기본 서명(Basic Signature: BS)

염기 서열 T의 i-번째 문자로부터 추출된 윈도우 W_i의 BS를 아래와 같이 정의한다. 특성 추출을 위한 문자 집합으로서 알파벳 Σ의 부분집합인 $\Sigma'=\{S_1, S_2, \cdots, S_n\}$을 사용한다.

$$BS(W_i) = ((S_1[min_1, max_1], S_2[min_2, max_2], \cdots, S_n[min_n, max_n]), i)$$

여기에서 min_i는 윈도우 상에서의 문자 S_i의 최소 출현 빈도를 나타내고, max_i는 문자 S_i의 최대 출현 빈도를 나타낸다. □

본 연구에서는 염기 서열의 기본 특성을 고려하여 Σ' = {A, C, G, T}로 설정한다. 즉 4개의 문자를 BS의 출현 빈도 추출에 사용한다. 윈도우의 서명 BS를 구하는 방식을 예를 들어 설명해 보자. 윈도우 W_{100}이 'ACTGGT'라면 윈도우에 대한 서명 $BS(W_{100})$는 ((A[1,1], C[1,1], G[2,2], T[2,2]), 100)이 된다. W_{100} 내에서 A와 C는 각각 단 한번 출현하므로 min_1, max_1, min_2, max_2는 모두 1의 값을 갖는다. 마찬가지로, G, T는 각각 두 번 출현하므로 min_3, max_3, min_4, max_4는 모두 2의 값을 갖는다. 윈도우 W_{100}이 'ACTBGT'라면, 윈도우에 대한 서명 $BS(W_{100})$는 ((A[1,1], C[1,2], G[1,2], T[2,3]), 100)이 된다. 이 경우, 문자 B는 A가 아닌 C, G, T 중 임의의 하나의 문자를 의미하므로 C, G, T의 최대 출현 빈도는 염기 서열상에서 출현하는 B의 개수만큼 증가되어 표현된다. 이는 B의 위치에 C, G, T가 올 수 있음을 의미한다.

서명 BS로 표현된 각 데이타 윈도우는 n차원 직사각형 영역으로 표현되며, 다차원 공간 인덱스에 저장된다. 이 때, 해당 윈도우의 식별자로 i가 함께 저장된다. 이 때 인덱스에 저장되는 윈도우 서명의 총 개수는 |T|-|W|+1과 같다. 일반적인 경우 |T|의 크기는 윈도우 길이 |W|보다 훨씬 크므로 인덱스에 저장되는 윈도우 서명의 총 개수는 |T|와 거의 동일하다. 또한 다차원 인덱스 상에서 각 윈도우 서명을 직사각형 객체로 표현하기 위한 공간은 하나의 염기 문자가 차지하는 공간보다 크다. 따라서 다차원 공간 인덱스의 크기는 데이타 염기 서열의 크기보다 커져서, 데이타 염기 서열의 수배에서 수십 배의 크기가 된다.

본 연구에서는 인덱스의 크기를 줄이기 위한 방안으로 데이타 윈도우를 나타내는 4차원 직사각형들을 영역별로 묶어서 MBR(Minimum Bounding Rectangle)로 저장하는 방법을 사용한다. 제안된 기법에서 인덱스 구성을 위하여 연속적으로 추출된 W_i와 W_{i+1}는 매우 유사한 서명으로 표현되어, 인덱스 공간에서도 매우 근접한 위치에 매핑되게 된다. 이러한 특징을 이용하여 연속된 c개의 데이타 윈도우 서명을 하나의 MBR로 표현한 후 c개의 직사각형을 인덱스에 저장하는 대신 MBR만을 인덱스에 저장하고, 선두 데이타 윈도우의 오프셋 i를 MBR의 식별자로 사용한다. 이 결과, 전체 인덱스 크기는 약 1/c로 줄어들게 된다.

4.3 질의 처리

매치 유형에 따라서 질의 처리를 위한 질의 직사각형을 표현하는 방법은 다음과 같이 달라진다.

(1) 완전 매치 질의: 정의 1에서 보인 데이타 윈도우의 서명 추출과 동일한 방식에 의하여 질의 직사각형을 구한다.

(2) 와일드카드 매치 질의: (1)에서 구한 각각의 max_i (i=1,2,3,4)를 질의 염기 서열에 포함된 와일드카드 문자의 수만큼 증가 시킨다.

(3) k-미스매치 질의: (2)에서 구한 각각의 min_i (i=1,2,3,4)를 k만큼 감소시키고 각각의 max_i (i=1,2,3,4) 를 k만큼 증가시킨다. 이는 각 문자 유형마다 최대 k개의 불일치가 각 문자의 삭제나 삽입의 형태로 표현될 수 있음을 나타낸다. 이 때, (min_i-k)가 0보다 작은 경우에는 최소 출현 빈도로 0을 지정한다.

질의 직사각형을 구성한 후에는 인덱스를 이용하여 질의 직사각형과 중첩되는 모든 후보 직사각형들을 검색한 후 후보 직사각형의 오프셋 정보에 의하여 디스크 내의 데이타 염기 서열을 직접 액세스하여 잘못 채택된 부분 염기 서열을 제거하는 과정을 통해 원하는 결과를 얻을 수 있다. 그러나 4.2절에서 언급되었듯이 제안된 인덱싱 방법에서는 인덱스 저장 공간을 줄이기 위하여 MBR 정보만이 저장되어 있으므로 실제로는 디스크 상에 저장된 염기 서열 T 상의 연속된 c개의 데이타 윈도우에 대하여 후처리를 수행하게 된다.

지금까지 질의 처리 방식에서는 |Q|=|W|를 가정하였다. 그러나 일반적으로 질의 염기 서열의 길이 |Q|는 윈도우의 크기 |W|와 다르다. 다음은 질의 염기 서열 Q의 길이에 따른 각각의 인덱스 검색 방식을 보인다. 먼저 (1) |Q|=|W|의 경우는 앞에서 기술한 방식에 의하여 질의 직사각형을 구성한 후, 인덱스 검색을 수행한다. (2) |Q|<|W|의 경우는 Q의 뒷 부분에 |W|-|Q|개의 와일드카드 문자 *를 채워서 새로운 질의 염기 서열 Q′을 생성한다. |Q′|=|W|의 조건을 만족하므로 Q′에 대하여 (1)의 방식에 의하여 인덱스 검색을 수행한다. (3) |Q|>|W|의 경우는 우선, 그림 1과 같이 Q를 $|Q_i|=|W|$의 조건이 만족되는 $p(=\lceil |Q|/|W| \rceil)$개의 서브 질의 서열 Q_i로 분할한다. 이 때 마지막 서브 질의 서열 Q_p는 길이 |W|의 조건을 만족하기 위하여 Q_p-1과 중첩되어 설정될 수 있다. 다음, 각각의 Q_i에 대하여 (1)의 방식에 의하여 인덱스 검색을 수행한 후, 얻어진 결과들을 병합한다.

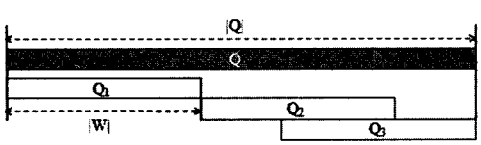


그림 1 윈도우 크기보다 큰 질의 서열을 분할하는 방법

5. WSI(Weighted Signature Index)

5.1 BSI의 문제점

BSI에서는 아래와 같은 두 가지 근본적 문제점을 가지고 있다.

첫 번째 문제점은 서로 다른 윈도우들이 동일한 서명으로 표현된다는 것이다. BSI는 염기 문자의 윈도우 내 출현 빈도만을 이용해 윈도우의 서명을 작성하므로 동일한 서명으로 표현되는 서로 다른 윈도우들의 수가 매우 많아지게 된다. 따라서 착오 채택(false alarm)의 수가 증가하며, 이 결과, 후처리 비용도 증가하여 검색 효율이 저하된다. 계산에 의하면 평균적으로 $4^{|W|}/4H_{|W|}$ 개의 서로 다른 윈도우들이 동일한 서명으로 표현되며, 완전매치 질의의 경우 이 중에서 단 하나 만이 실제로 질의를 만족시키고 나머지는 모두 착오 채택이 된다(자세한 계산 과정은 Appendix 1 참고).

두 번째로 각 서명에 매핑되는 윈도우들의 수가 고르지 않다는 것이다. DNA 염기 서열에 나타나는 특성문자는 일정한 출현 빈도를 갖는다. (예를 들어 18번 염색체의 경우에 A 30%, C 20%, G 20%, T 30%의 분포를 보인다.), 따라서 평균점 (0.3*|W|, 0.2*|W|, 0.2*|W|, 0.3*|W|)에 가까운 서명일수록 많은 윈도우들이 매핑되는 경향이 있다. 따라서 질의의 서명이 평균점에서 가까운 곳에 위치할수록 많은 수의 후보들이 질의의 서명이 중심점에서 멀어질수록 적은 수의 후보들이 인덱스 검색의 결과로 반환된다. 이러한 현상은 질의의 서명에 따라 질의 수행 시간이 고르지 않은 문제점을 초래한다.

위에서 지적된 BSI의 두 가지 문제를 해결하기 위해서는 서로 다른 서명의 총 개수를 늘임과 동시에 윈도우들을 각 서명에 골고루 매핑시키는 방안이 필요하다.

5.2 기본 전략

BSI의 문제점을 해결하기 위한 가장 간단한 방법으로 윈도우로부터 더 많은 정보를 추출하여 서명에 포함시키는 것을 생각할 수 있다. 그러나 이 방법은 서명의 차원과 함께 인덱스의 차원을 증가시켜 인덱스의 검색 효율을 저하시킬 수 있다. 이 논문에서는 서명의 차원을 증가시키지 않으면서도 윈도우를 좀 더 변별력 있게 표현할 수 있는 방법으로 윈도우의 각 문자 위치에 가중치를 부여하는 방법을 제안한다. 이 방법을 사용하면 염기 문자의 윈도우 내 출현 빈도뿐 아니라 출현 위치에 대한 정보도 서명에 포함시킬 수 있다. 이를 위해 먼저 윈도우의 각 위치에 가중치를 부여하는 가중치 함수 w(i) $(1 \le i \le |W|)$를 정의한다. 다음으로 염기 서열 T 상

의 각 데이타 윈도우를 다음과 같은 가중치 서명으로 표현한다.

정의 2. 윈도우 W_i의 가중치 서명(Weighted Signature: WS)

염기 서열 T의 i-번째 문자로부터 추출된 윈도우 Wi의 WS를 아래와 같이 정의한다. 특성 추출을 위한 문자 집합으로서 알파벳 Σ의 부분 집합인 $\Sigma'=\{S_1, S_2, \cdots, S_n\}$을 사용한다.

$WS(W_i) = ((S_1[wmin_1, wmax_1], S_2[wmin_2, wmax_2], \cdots, S_n[wmin_n, wmax_n]), i)$

여기에서 $wmin_i$는 윈도우 상에서 문자 S_i가 반드시 출현하는 모든 위치들의 가중치 합을 나타내며, $wmax_i$는 윈도우 상에서의 문자 S_i가 출현할 가능성이 있는 모든 위치들의 가중치 합을 나타낸다. □

BSI의 경우와 마찬가지로 WSI에서도 염기 서열의 기본 특성을 고려하여 Σ′ = {A, C, G, T}로 설정한다. 이제 윈도우의 가중치 서명 WS를 구하는 방식을 예를 들어 설명해 보자. 가중치 함수로는 w(i) = i를 가정한다. 즉, 윈도우의 i번째 위치에 i를 가중치로 부여한다고 가정한다.

한 예로서, W_{200} = 'ACTBGT'를 생각해보자. 문자 B는 C, G, T 중의 하나로 대체될 수 있으므로 BSI의 경우에는 서명이 $BS(W_{200})$ = ((A[1,1], C[1,2], G[1,2], T[2,3]), 200)으로 표현된다. 그러나 가중치를 고려하면 다음과 같은 이유로 서명이 $WS(W_{100})$ = ((A[1,1], C[2,6], G[5,9], T[9,13]), 200)으로 표현된다. 문자 A는 첫 번째 위치에만 출현 가능하므로 A[1,1]이 된다. 문자 C는 두 번째 위치에만 출현하지만, 네 번째 위치에도 출현할 가능성이 있으므로 C[2,6]이 된다. 문자 G는 다섯 번째 위치에만 출현하지만, 네 번째 위치에도 출현할 가능성이 있으므로 G[5,9]이 된다. 문자 T는 세 번째와 여섯 번째에 출현하고, 네 번째 위치에도 출현할 가능성이 있으므로 T[9,13]이 된다.

위와 같은 방식을 사용함으로써, BSI의 경우에는 동일한 서명으로 표현되었던 서로 다른 윈도우가 이제는 서로 다른 서명으로 표현된다. 예를 들어, W_{100} = 'ACTGGT'와 W_{150} = 'CGAGTT'를 생각해보자. 출현 빈도만 고려하면 두 윈도우의 서명은 모두 (A[1,1], C[1,1], G[2,2], T[2,2])로 표현된다. 그러나 출현 빈도뿐만 아니라 출현 위치도 함께 고려하면 W_{100}의 서명은 (A[1,1], C[2,2], G[9,9], T[9,9])로 표현되고, W_{150}의 서명은 (A[3,3], C[1,1], G[6,6], T[11,11])로 표현된다. 이렇게 출현 빈도와 함께 출현 위치에 대한 정보도 함께 고려하여 인덱스를 구성하는 방식을 본 논문에서는 WSI(Weighted Signature Index)라고 정의한다. WSI를 사용하면 그림 2와 같이 동일한 윈도우 서명으로 표현되었던 윈도우들을 인덱스 공간에 분산시키는 효과를 얻을 수 있다.

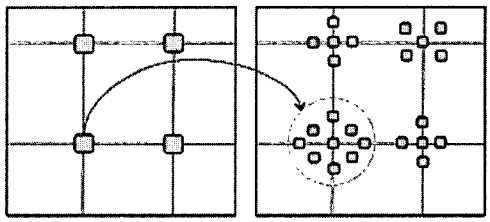


그림 2 가중치를 이용함에 따른 윈도우 서명의 변화

WSI에서 사용하는 질의 처리 방법은 기본적으로 BSI에서 사용한 방법과 동일하다. 하지만 k-미스매치를 처리하는 방법에서는 BSI와 차이를 보인다. BSI에서는 k-미스매치를 질의에 적용할 때 최대값에는 k를 더하고 최소값에는 k를 빼는 방식을 사용하였다. 하지만 WSI 경우에는 불일치 된 문자의 위치를 고려해야 하므로 다음과 같은 방식으로 가중치가 반영된 질의 직사각형을 구할 수 있다.

먼저, 정의 2에서 보인 데이타 윈도우의 가중치 서명 추출과 동일한 방식으로 각 특성 문자 X에 대한 wmin과 wmax를 구한다. 다음으로, 최대 k개의 미스매치를 고려하여 X가 반드시 출현하는 위치들 중에서 가중치가 가장 큰 k개의 위치에서 X가 아닌 다른 문자가 출현한다고 가정하고 문자 X의 최소 가중치 합을 구한다. 즉, 문자 X에 대하여 X가 반드시 출현하는 위치들 중에서 가중치가 가장 큰 k개 위치의 가중치 합이 m 이라면, 문자 X의 최소 가중치 값을 wmin-m로 표현한다. 동일한 방식으로, 최대 k개의 미스매치를 고려하여 X가 출현하지 않은 위치들 중에서 가중치가 가장 큰 k개의 위치에서 X가 출현한다고 가정하고 문자 X의 최대 가중치 합을 구한다. 즉, 문자 X가 출현하지 않는 위치들 중에서 가중치가 가장 큰 k개 위치의 가중치 합이 M 이라면, 문자 X의 최대 가중치 값을 wmax+M로 표현한다.

5.3 효율적인 가중치 함수 w(i)의 선택

WSI에서는 출현 빈도와 함께 출현 위치도 고려하므로, BSI에서는 동일한 서명으로 표현되었던 서열들이 WSI에서는 서로 다른 서명으로 표현되므로 인덱스 검색 결과로 나타나는 착오 채택의 수가 줄어든다. 이때 가중치 함수에 의해서 다차원 공간 상에서의 서명의 분포가 결정된다는 것을 고려하면 효율적인 가중치 함수의 선택은 매우 중요하다. BSI에서 동일한 서명으로 표현되었던 서열들을 가중치 함수를 사용하여 분산시켰을 때 분산된 서명들을 모두 포함하는 MBR이 있을 때

MBR이 크면 클수록 넓게 분산된다는 것을 의미한다. 하지만 MBR이 너무 커지게 되면 인접한 MBR들과 중첩하는 상황이 발생할 수 있다. 따라서 인접한 MBR들과 중첩하지 않으면서 MBR을 최대로 키울 수 있는 가중치 함수를 선택해야 한다. 위의 내용을 공식적으로 설명하기 위해, 다음 용어들을 정의한다.

윈도우 내 가중치 값들을 오름차순으로 정렬했을 때 i번째의 값을 sw(i)로 정의한다. 문자 X를 s개 포함하는 모든 윈도우들의 가중치 서명을 구했을 때, 문자 X의 wmin 값 중 최소값을 $R_{min}(X,s)$라고 하자. 이 값은 가중치 값들 중에서 가장 작은 값 s개의 합으로 표현되므로 $R_{\min}(X,s)=\sum_{k=1}^{s} sw(k)$를 만족한다. 문자 X를 s개 포함하는 모든 윈도우들의 가중치 서명을 구했을 때, 문자 X의 wmax 값 중 최대값을 $R_{max}(X,s)$라고 하자. 이 값은 가중치 값들 중에서 가장 큰 값 s개의 합으로 표현되므로 $R_{\max}(X,s)=\sum_{k=|W|-(s-1)}^{|W|} sw(k)$를 만족한다.

MBR들이 중첩되지 않기 위해서는 0부터 |W|-1까지의 모든 s에 대해 $R_{max}(X,s) < R_{min}(X,s+1)$를 만족해야 하므로 식을 정리하면 다음과 같다.

$$R_{\max}(X,s) < R_{\min}(X,s+1)$$
$$\Leftrightarrow R_{\max}(X,s) - R_{\min}(X,s+1) < 0$$
$$= \sum_{k=|W|-(s-1)}^{|W|} sw(k) - \sum_{k=1}^{s+1} sw(k) < 0$$

즉, 위의 식을 만족시키는 가중치 함수 w(i)를 선택해야 인덱스의 필터링 효과를 최대로 높일 수 있다. 예를 들어, w(i) = i+C라고 놓고 위의 식을 풀어보면 다음과 같다.

$$\sum_{k=|W|-(s-1)}^{|W|} sw(k) - \sum_{k=1}^{s+1} sw(k)$$
$$= \sum_{k=|W|-(s-1)}^{|W|} w(k) - \sum_{k=1}^{s+1} w(k)$$
$$= Cs - C(s+1) + \sum_{k=|W|-(k-1)}^{|W|} k - \sum_{k=1}^{s+1} k$$
$$= -C - (s^2 + (1-|W|)s + 1) < 0$$

이때 위 식은 0부터 |W|-1까지의 모든 s에 대해서 만족해야 하므로 다음과 같이 정리된다.

$$-C - (s^2 + (1-|W|)s + 1) < 0$$
$$\Leftrightarrow C > \max_{s=0}^{|W|}\{-(s^2 + (1-|W|)s + 1)\}$$
$$\Leftrightarrow C > \frac{1}{4}(|W|-1)^2 - 1$$

위 결과를 통해서 C의 값을 결정할 수 있다. 이 논문에서는 계산의 편리함을 위해서 위 조건을 만족하는 C로 $|W|^2$을 선택한다.

6. 성능 평가

본 장에서는 실험에 의한 성능 평가를 통하여 제안된 기법의 우수성을 규명한다. 제 6.1절에서는 실험 환경을 설명하고, 제 6.2절에서는 실험을 위한 파라미터 값을 설정한다. 제 6.3절에서는 실험 결과를 분석한다.

6.1 실험 환경

실험에 사용된 데이타 염기 서열 T는 NCBI[NCBI]에서 다운로드 받은 2.5Mbp의 Human Chromosome 3, 5Mbp의 Human Chromosome 17, 7.5Mbp의 Human Chromosome 1, 10Mbp의 Human Chromosome 2, 20Mbp의 Human Chromosome 10, 40Mbp의 Human Chromosome 5염기 서열의 6 가지 데이타 셋을 사용한다. 이들 각각의 염기 서열 내에는 Σ ={A, C, G, T}의 네 종류의 문자가 출현한다.

질의 염기 서열 Q는 실험 데이타 셋으로부터 랜덤 방식에 의하여 추출한 256에서 2048의 길이를 갖는 500개의 염기 서열과 실험실 레벨에서 생물학자들에 의해 발견된 실제의 염기 서열 500개[1]를 다운로드 받아 사용한다.

성능 평가는 다음 세 가지의 서로 다른 기법을 대상으로 한다. WSI는 본 논문에서 제안한 위치에 따른 가중치를 이용한 다차원 인덱스 검색 방식으로서, 다차원 인덱스로 사용된 R*-트리는 Maryland 대학의 Faloutsos 교수 팀에서 개발한 R*-tree Version 2.0을 사용한다. 성능 비교를 위한 기존의 기법으로는 위치에 따른 가중치를 고려하지 않는 다차원 인덱스 검색 방식 BSI와 순차 검색 방식 SeqScan, 접미어 트리를 이용한 검색 방식인 Suffix의 세 가지를 사용한다. 또한, 평가 지수로는 인덱스 크기 및 완전 매치 질의, 와일드 카드 매치 질의, k-미스 매치 질의를 수행한 평균 질의 처리 시간을 사용한다.

실험을 위한 하드웨어 플랫폼으로는 Redhat 계열의 배포판인 Fedora core2를 운영체제로 사용하고, 512MB의 주기억장치와 80GB(7200RPM) 디스크를 갖는 Pentium Ⅳ 2.6GHz의 PC를 사용한다. 단, 실험을 위한 버퍼 크기는 2MB로 제한하여 사용한다.

6.2 BSI와 WSI의 기본 파라미터 값 설정

6.2.1 윈도우 크기 설정

BSI와 WSI는 윈도우의 크기가 커질수록 착오 채택되는 윈도우의 개수가 줄어들어 좋은 성능을 보인다. 그

1) ftp://ftp.ensembl.org/pub/current_human/data/fasta/cdna/Homo_sapiens.1834.may.cdna.fa.gz

러나, 4.3절에서 언급한 바와 같이 윈도우의 크기가 질의 염기 서열의 크기(|Q|)보다 큰 경우, 질의 염기 서열에 와일드 카드 문자를 채워서 인덱스 검색을 수행하기 때문에 성능이 저하되는 문제가 발생한다. 따라서, 일반적으로 윈도우 크기를 질의 염기 서열의 크기보다 조금 작은 값을 이용한다. 본 실험에서는 효율적인 윈도우 크기를 결정하기 위하여 [NCBI]에서 다운로드 받은 60에서 104,682의 길이를 갖는 실제의 질의 염기 서열 35,685개에 대하여 그 길이를 분석한 결과를 토대로 한다(그림 6.1 참조). 실험 결과에 따르면, 전체 질의 염기 서열의 62%에 해당하는 서열이 256에서 2048의 길이를 갖는 것으로 나타났다. 따라서 본 실험에서는 윈도우 크기 |W|의 기본 값으로 256을 설정하여 사용한다.

6.2.2 인덱스 공간 압축률 c의 크기 설정

본 논문에서 제안한 BSI, WSI방식은 인덱스의 크기를 줄이기 위한 방안으로 연속적으로 추출된 c개의 데이타 윈도우 서명을 하나의 MBR로 표현하여 인덱스에 저장하는 방식을 사용한다. 그림 3과 그림 4에 실험 데이타 10Mbp의 Human Chromosome 2 염기 서열을 이용하여 인덱스 공간 압축률c의 크기 변화에 따른 인덱스 크기 및 평균 질의 처리 시간을 비교한 결과를 보인다. 평균 질의 처리 시간은 k=10인 k-미스 매치 질의를 수행한 결과로, 인덱스 검색과 후처리 과정을 거쳐 최종 결과로 질의 염기 서열을 만족하는 서열 번호와 오프셋을 반환하는 총 시간을 의미한다.

그림 3에 실험 결과에 의하면 BSI, WSI의 두 방식 모두 c의 크기가 증가함에 따라 인덱스 크기가 급격히 감소됨을 알 수 있다. 또한, 그림 4의 평균 질의 처리 시간은 인덱스의 검색 공간이 축소됨에 따라 다차원 인덱스 검색 시간이 감소되어 전체 검색 시간이 감소하는 것으로 나타났다. 그러나, 어느 정도 이상의 공간 압축률이 되면 착오 채택으로 인한 후보 직사각형의 수가 많아져 전체 질의 처리 시간은 후처리 시간에 많은 영향을 받게 되어 다소 증가하는 것으로 나타났다. 본 실험에서는 인덱스 압축률 c의 기본 값으로 80을 설정하여 사용한다.

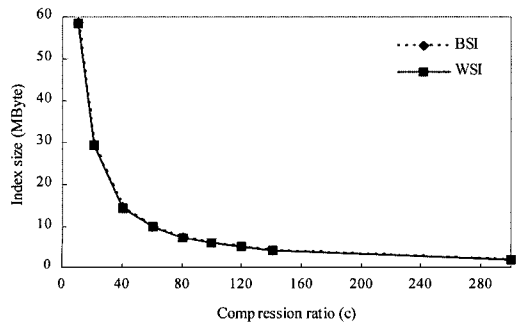


그림 3 인덱스 압축률 c의 크기 변화에 따른 인덱스 크기

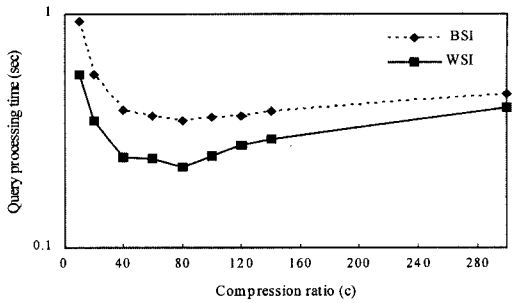


그림 4 인덱스 압축률 c의 크기 변화에 따른 질의 처리 시간

6.3 실험결과 및 분석

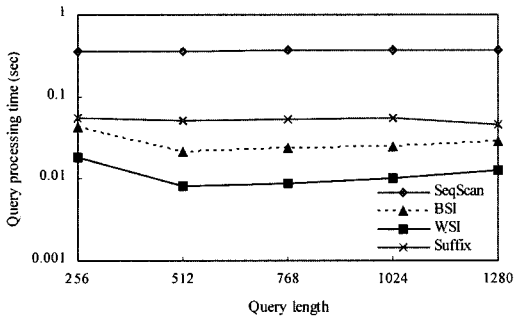
제안된 기법의 성능을 평가하기 위하여 질의 처리 시간을 기존 방식과 실험을 통하여 비교, 분석한다.

실험 1: 질의 길이 변화에 따른 질의 처리 시간 비교

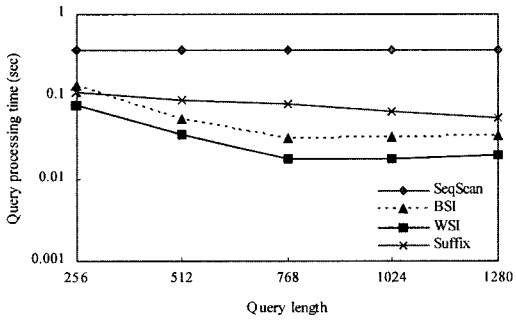
먼저 실험 1에서는 질의 길이 변화에 따른 질의 처리 시간을 실험한다. 그림 5(a)와 그림 5(b), 그림 5(c)에 10Mbp의 Human Chromosome 2 염기 서열을 이용하여 완전 매치, 와일드 카드 매치, k-미스 매치를 수행한 질의처리 시간을 보인다. 이때 와일드 카드 매치와 k-미스 매치를 위한 와일드 카드 문자의 개수와 k의 값은 질의 염기 서열의 평균 길이의 1%에 해당하는 값인 10으로 정하여 사용한다.

실험 결과에 의하면, 완전 매치, 와일드 카드 매치, k-미스 매치의 모두 경우에서 질의 길이가 증가함에 따라 SeqScan은 질의 길이와 무관한 질의 처리 시간을 갖는 것으로 나타났다. 이는 염기 서열의 경우 서열의 3에서 6사이의 짧은 길이에서 질의 서열과의 매치 여부가 결정되기 때문에, 질의의 길이와 상관없이 거의 일정한 질의 처리 시간을 갖기 때문이다. Suffix은 질의의 길이가 증가함에 따라 질의 처리 시간이 감소하는 것으로 나타났다. 그 이유는 접미어 트리 방식에서의 전체 질의 처리 시간은 질의와 매치되는 중간 노드 N을 검색한 이후에 중간 노드 N과 연결된 모든 서브 트리를 검색하여 단말 노드를 가져오는 데 걸리는 시간에 의해 좌우되기 때문이다. 즉, 질의의 길이가 증가함에 따라 서브 트리를 검색하는데 걸리는 시간이 감소하기 때문에 전체 질의 처리 시간이 감소하는 것으로 나타났다. 반면에WSI와 BSI는 질의의 길이가 증가하게 되면 해당 질의를 윈도우 크기로 분해하여 서브 질의를 생성하고, 생성된 각각의 서브 질의에 대하여 검색을 수행하여 얻어진 중간 결과를 병합하는 방식을 사용하기 인덱스 검색 시간은 다소 증가하지만, 질의의 길이가 증가함에

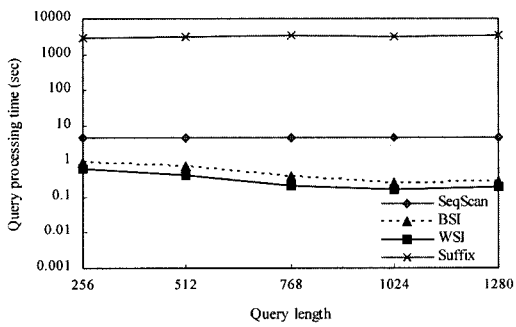
따라 후보 집합의 개수가 급격히 감소하기 때문에 후처리 시간은 감소하게 되어 전체 질의 처리 시간은 질의의 길이가 증가할수록 감소하는 것으로 나타났다. 완전 매치의 경우 WSI는 SeqScan에 비해 19배에서 44배까지, Suffix에 비해 2.9배에서 6.1배까지, BSI에 비해 2.2배에서 2.7배까지의 성능 향상 보였으며, 와일드 카드 매치의 경우 WSI는 SeqScan에 비해 4배에서 21배까지, Suffix에 비해 1.4배에서 4.5배까지, BSI에 비해 1.5배에서 1.8배까지의 성능 향상 보였다. 또한, k-미스 매치의 경우 SeqScan에 비해 7배에서 28배까지, Suffix에 비해 4747배에서 18905배까지, BSI에 비해 1.3배에서 1.6배까지의 성능 향상을 갖는 것으로 나타났다.



(a) 완전 매치의 경우



(b) 와일드 카드 매치의 경우



(c) k-미스 매치의 경우

그림 5 질의 길이 변화에 따른 질의 처리 시간 비교

실험 2: k의 크기 변화에 따른 k-미스 매치 질의 처리 시간 비교

다음 실험 2에서는 k의 크기 변화에 따른 k-미스 매치의 질의 처리 시간을 비교, 실험한다. 그림 6은 10Mbp의 Human Chromosome 2 염기 서열을 이용하여 k의 크기를 질의 길이의 0%부터 3%에 해당하는 값으로 증가시키면서 실험한 것으로, 실험 결과에 따르면 k가 증가함에 따라 네 가지 방식 모두 검색 공간이 확대되어 질의 처리 시간이 급격하게 증가하는 것으로 나타났다. 그러나 접미어 트리 방식이 k가 증가함에 따라 검색 공간의 급격한 증가로 인하여 질의 처리 시간이 급격하게 증가하는 것에 비하여, BSI와 WSI는 k의 크기에 거의 무관하게 기존 방식에 비해 좋은 검색 성능을 갖는 것으로 나타났다. WSI는 SeqScan 에 비해 16배에서 31배까지의 성능 향상을 가졌으며, Suffix 에 비해 최소 3배 이상, BSI에 비해 1.1배에서 2.3배까지의 성능 향상을 갖는 것으로 나타났다.

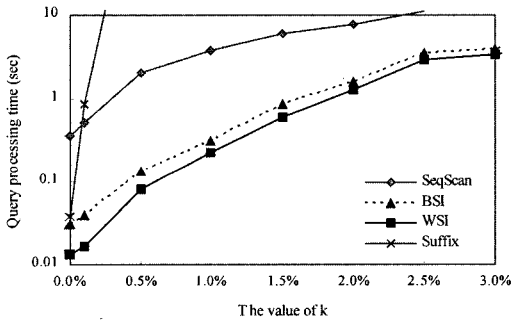


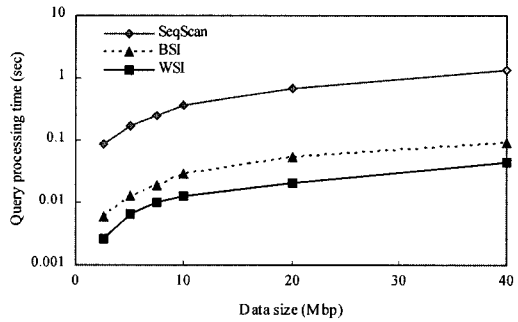
그림 6 k의 크기 변화에 따른 k-미스 매치의 질의 처리 시간 비교

실험 3: 데이타 크기 변화에 따른 질의 처리 시간 비교

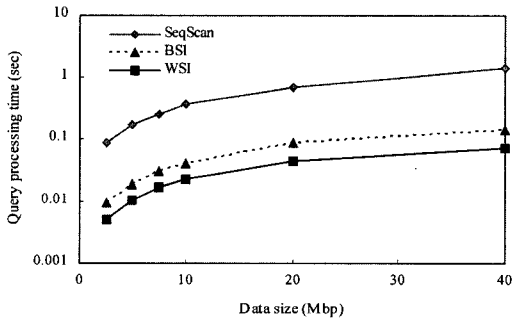
실험 3은 제안된 기법의 Scalability를 실험하기 위한 것으로, 데이타 크기 변화에 따른 질의 처리 시간을 각 기법에 대하여 비교, 분석한다. 단, 실험 1과 실험 2에서 Suffix가 BSI와 WSI에 비하여 모든 질의 처리 방식에서 급격한 성능 저하를 나타냄을 보였으므로 실험 3에서는 제외한다. 그림 7(a)와 그림 7(b), 그림 7(c)는 완전 매치, 와일드 카드 매치, k=10인 k-미스 매치를 각각 수행한 평균 질의 처리 시간을 나타내며, 윈도우 크기(|W|=256)와 인덱스 공간 압축률(c=80)을 고정시키고, 데이타의 크기를 변화시키는 경우의 평균 질의 처리 시간을 측정한 것이다. 이때, 와일드 카드 매치와 k-미스 매치를 위한 와일드 카드 개수와 k는 10이다.

실험 결과에 따르면, 완전 매치, 와일드 카드 매치, k-미스 매치 질의 모두에 대하여 데이타의 크기가 증가함에 따라 SeqScan, BSI, WSI모두 질의 처리 시간이

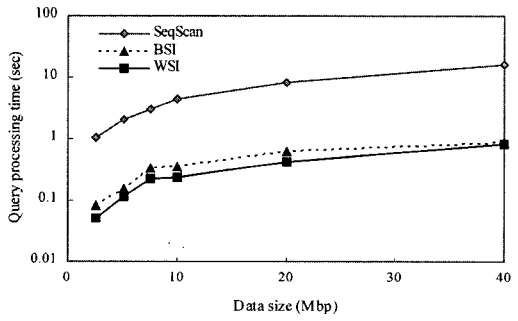
선형적으로 증가하는 것으로 나타났으며, 그 이유는 데이타의 크기 증가에 따라 검색 공간이 확대되기 때문이다. 그러나, BSI와 WSI는 데이타 크기에 거의 무관하게 SeqScan보다 좋은 질의 처리 성능을 보였으며, 완전 매치의 경우 WSI는 SeqScan에 비해 25배에서 33배까지, BSI에 비해 1.8배에서 2.5배까지의 성능 향상을 보였다. 또한 와일드 카드 매치의 경우 WSI는 SeqScan에 비해 15배에서 19배까지, BSI에 비해 1.7배에서 1.9배까지의 성능 향상을 보였으며, k-미스 매치의 경우 WSI는 SeqScan에 비해 13배에서 20배까지, BSI에 비해 1배에서 1.5배까지의 성능 향상을 갖는 것으로 나타났다.



(a) 완전매치의 경우



(b) 와일드 카드 매치의 경우



(c) k-미스 매치의 경우

그림 7 데이타 크기 변화에 따른 질의 처리 시간 비교

7. 결 론

완전 매치 질의, 와일드카드 매치 질의, k-미스매치 질의는 EST(expressed sequence tag)의 검색[14] 및 DNA 전사 요소(DNA transcription factor)의 검색[14] 등 분자 생물학 분야의 염기 서열 검색을 위하여 널리 사용된다. 본 논문에서는 이러한 질의들을 효과적으로 처리하는 방안에 관하여 논의하였다.

기존의 기법들로는 보이어-무어 알고리즘을 이용한 기법, KMP 알고리즘을 이용한 기법, 아호-코라식 알고리즘과 스캔 벡터를 이용한 기법 등이 제안된 바 있다. 이들은 두 서열간의 정합 시 CPU 성능을 최적화하는데 궁극적인 목표를 두었다. 반면, 질의 서열과의 비교를 위하여 전체 데이타 서열을 디스크로부터 읽어야 한다는 성능 상의 문제점을 갖는다. 접미어 트리를 이용한 기법은 접미어 트리를 사용하는 일종의 여과 및 정제 방식을 취함으로써 다른 기법들과 비교하여 상대적으로 좋은 검색 성능을 얻을 수 있다. 반면, 이 기법은 접미어 트리 고유의 특성으로 인하여 저장 공간의 오버헤드가 크며, 이러한 큰 접미어 트리로 인하여 검색 시간이 크다는 단점을 갖는다.

본 논문에서는 이러한 문제점들을 해결하는 새로운 기법을 제안하였다. 먼저, 염기 서열의 효과적인 인덱싱 기법을 제안하였다. 제안된 기법은 염기 서열로부터 슬라이딩 윈도우들을 추출한 후, 각 윈도우 내에서 각 염기 문자의 출현 빈도를 파악하여 다차원 공간 인덱스의 하나인 R*-트리에 저장한다. 특히, 각 문자의 출현 위치에 따라 서로 다른 가중치를 부여함으로써 추출된 윈도우들이 다차원 공간상에 집중되는 현상을 억제하였다. 또한, 제안된 인덱싱 기법을 기반으로 완전 매치 질의, 와일드카드 질의, k-미스매치 질의를 효과적으로 처리할 수 있는 질의 처리 기법을 제안하였다. 제안된 기법은 질의 서열을 다차원 공간의 한 직사각형으로 변환한 후, 인덱싱 단계에서 생성된 R*-트리를 검색함으로써 이 직사각형과 중첩된 윈도우들을 효과적으로 파악한다. 이 윈도우들이 질의 결과에 포함될 가능성이 높은 후보들을 의미하며, 이 부분들만을 디스크로부터 액세스함으로써 최종 결과에 포함되는지의 여부를 결정한다.

제안한 기법의 우수성을 검증하기 위하여 기존의 접미어 트리를 이용한 기법과의 다양한 실험을 통한 성능 비교를 수행하였다. 실험 결과에 의하면, 제안된 기법은 접미어 트리에 비해서 완전 매치인 경우 3배 이상, 와일드카드 매치인 경우 2배 이상, k-미스매치인 경우 수십 배 이상의 성능 향상을 보이는 것으로 나타났다.

참 고 문 헌

[1] C. Gibas and P. Jambeck, Developing Bioinformatics Computer Skills, O'Reilly and Associates Inc., 2001.

[2] D. W. Mount, Bioinformatics: Sequence and Genome Analysis, Cold Spring Harbor Laboratory Press, 2001.

[3] T. Smith and M. Waterman, "Identification of common molecular subsequences," Journal of Molecular Biology, Vol. 147, pp. 195-197, 1981.

[4] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," Journal of Molecular Biology, Vol. 215, pp. 403-410, 1990.

[5] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: A new generation of protein database search programs," Nucleic Acids Research, 25(17), 1997.

[6] T. Kaheci, A. K. Singh, "An efficient index structure for string databases," VLDB, 2001.

[7] A. Guttman, "R-Trees, A dynamic index structure for spatial searching," ACM SIGMOD, pp. 47-57, 1984.

[8] R.S. Boyer, J. S. Moore, "A fast string searching algorithm," Communications of the ACM, Vol. 20, pp. 762-772, 1977.

[9] D. E. Knuth, J. H. Morris, V. B. Pratt. "Fast pattern matching in strings," SIAM J. Comput., Vol. 6, pp. 323-350, 1977.

[10] A. Aho, M. Corasick, "Efficient string matching: an aid to bibliographic search," Communications of the ACM, Vol. 18, pp. 333-40, 1975.

[11] G. A. Stephen, String Searching Algorithm, World Scientific Publishing, 1994.

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

[13] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," ACM SIGMOD, pp. 322-331, 1990.

[14] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, 1 Edition, January 1997.

[15] S. Berchtold, D. A. Keim, and Hans-Peter Kriegel, "The X-tree: An index structure for high-dimensional data," VLDB pp 28-39, 1996.

[16] R. Agrawal, C. Faloutsos, and A. Swam, "Efficient similarity search in sequence databases," FODO, pp 69-84, 1993.

[17] C. Faloutsos and K. Lin, "FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," ACM SIGMOD, pp. 163-174, 1995.

[18] http://www.ncbi.nlm.nih.gov

부 록

부 록 1

DNA 염기 서열의 길이를 |T|라 하고 윈도우의 크기를 |W|라고 할 때, 슬라이딩 윈도우 방식으로 모든 윈도우를 생성했을 때 윈도우의 개수는 |T|-|W|+1개이다. 일반적인 경우에 |T|가 |W|보다는 훨씬 큰 수이므로 |T|-|W|+1≒|T|를 만족한다. 따라서 계산상의 편리함을 위해서 윈도우의 총 개수를 |T|라고 하면 다음과 같은 결과를 얻을 수 있다.

(1) 윈도우 서열당 평균 윈도우 개수

$$\frac{\text{윈도우 개수}}{\text{서로 다른 윈도우 서열의 개수}} = \frac{|T|}{4^{|W|}}$$

(2) 윈도우 서명당 평균 윈도우 개수

$$\frac{\text{윈도우 개수}}{\text{서로 다른 윈도우 서명의 개수}} = \frac{|T|}{{}_4H_{|W|}}$$

(3) 윈도우 서명당 서로 다른 윈도우 서열의 평균 개수

$$\frac{\text{윈도우 서열의 개수}}{\text{윈도우 서명의 개수}} = \frac{4^{|W|}}{{}_4H_{|W|}}$$

(4) 윈도우 서명당 착오 채택되는 서로 다른 윈도우 서열의 평균 개수

$$\text{윈도우 서명당 서로 다른 윈도우 서열의 개수} - 1 = \frac{4^{|W|}}{{}_4H_{|W|}} - 1$$

(5) 윈도우 서명당 착오 채택되는 평균 윈도우 개수

착오 채택되는 서로 다른 윈도우 서열의 평균 개수

× 윈도우 서열당 평균 윈도우 개수

$$= \left(\frac{4^{|W|}}{{}_4H_{|W|}} - 1\right) \times \frac{|T|}{4^{|W|}} \approx \frac{|T|}{{}_4H_{|W|}}$$

김 우 철

2003년 연세대학교 컴퓨터과학과 졸업(공학사). 2004년~현재 연세대학교 컴퓨터과학과 석사과정 재학중. 관심분야는 바이오인포메틱스, LBS, 데이타베이스 보안, 멀티미디어 데이터베이스 등

박 상 현

1989년 2월 서울대학교 컴퓨터공학과(학사). 1991년 2월 서울대학교 컴퓨터공학과(석사). 2001년 2월 UCLA대학교 전산학과(박사). 1991년 3월~1996년 8월 대우통신 연구원. 2001년 2월~2002년 6월 IBM T. J. Watson Research Center

Post-Doctoral Fellow. 2002년 8월~2003년 8월 포항공과대학교 컴퓨터공학과 조교수. 2003년 9월~현재 연세대학교 컴퓨터과학과 조교수. 관심분야는 데이타베이스, 데이타 마이닝, 바이오인포매틱스, XML

원 정 임

1992년 2월 한림대학교 전자계산학과(학사). 1997년 8월 한림대학교 컴퓨터공학과(석사). 2003년 2월 한림대학교 컴퓨터공학과(박사). 2000년 3월~2004년 2월 한림대학교 교양교육부 강의전담. 2004년 3월~현재 연세대학교 컴퓨터과학과 연구교수. 관심분야는 데이타베이스 시스템, 데이타 마이닝, XML 응용, 바이오 정보공학, 데이타베이스 보안

김 상 욱

1989년 2월 서울대학교 컴퓨터공학과 졸업(학사). 1991년 2월 한국과학기술원 전산학과 졸업(석사). 1994년 2월 한국과학기술원 전산학과 졸업(박사). 1991년 7월~8월 미국 Stanford University, Computer Science Department 방문 연구원. 1994년 2월~1995년 2월 KAIST 정보전자연구소 전문 연구원. 1999년 8월~2000년 8월 미국 IBM T.J. Watson Research Center Post-Doc. 1995년 3월~2000년 8월 강원대학교 컴퓨터정보통신공학부 부교수. 2003년 3월~현재 한양대학교 정보통신대학 정보통신학부 부교수. 관심분야는 데이타베이스 시스템, 저장 시스템, 데이타 마이닝, 바이오 정보공학, 멀티미디어 정보 검색, 공간 데이타베이스/GIS, 주기억장치 데이타베이스, 트랜잭션 관리

윤 지 희

1982년 2월 한양대학교 전자공학과 졸업(학사). 1985년 3월 일본 구주대학교 정보공학과 졸업(석사). 1988년 3월 일본 구주대학교 정보공학과 졸업(박사). 1998년 3월~1999년 2월 미국 UCLA대학교 전산학과 방문교수. 1988년 4월~현재 한림대학교 정보통신공학부 교수. 관심분야는 시계열 데이타베이스, 데이타 마이닝, XML, 공간 데이타베이스/GIS