DNA시퀀스 데이터베이스를 위한 효율적인 유사 검색 기법

An Efficient Homology Search Method for DNA Sequence Databases

홍상균(Sang-Kyoon Hong)*, 원정임(Jung-Im Won)**, 윤지희(Jee-Hee Yoon)*,
박상현(Sang-Hyun Park)**, 김상욱(Sang-Wook Kim)***

요 약

유사 검색은 분자 생물학 분야에서 사용되는 매우 중요한 연산이다. 본 논문에서는 DNA 시퀀스 데이터베이스를 대상으로 하는 효율적인 유사 검색 기법을 제안한다. 제안된 인덱스는 트라이를 기본 구조로 채택하며, 포인터 없이 트라이를 비트 스트링으로 표현하는 이진 접미어 트라이 구조를 갖는다. 유사 검색 알고리즘은 다이나믹 프로그래밍 기법에 근거하여 이진 접미어 트라이를 루트로부터 너비 우선(breadth-first) 방식으로 운행하며 경로 상에 존재하는 유사 서브시퀀스를 검색해 낸다. 또한 트라이 운행에 있어 검색 대상의 단말 노드를 빠르게 찾기 위하여 별도의 다차원 인덱스 구조를 사용한다. 제안된 기법의 우수성을 검증하기 위하여, 실험을 통한 성능 평가를 수행하였다. 실험 결과에 의하면, 제안된 유사 검색 기법은 기존의 접미어 트리 기반의 유사 검색 기법과 비교하여 더 작은 저장 공간을 가지고도 수배에서 수십배의 검색 성능의 개선 효과를 가지는 것으로 나타났다.

주제어 : DNA 시퀀스 데이터베이스, 인덱싱, 트라이

ABSTRACT

In molecular biology, DNA sequence searching is one of the most crucial operations. In this paper, we suggest an efficient homology search method for DNA sequence databases. The proposed index consists of two parts: the primary part represents the trie as bit strings without any pointers, and the secondary part helps fast accesses of the leaf nodes of the trie that need to be accessed for post processing. We also suggest an efficient approximate subsequence match algorithm based on that index. Our method employs a dynamic programming search technique which is driven by traversing the trie index in the breadth-first order. To verify the superiority of the proposed approach, we conducted a performance evaluation via a series of experiments. The results revealed that the proposed approach, which requires smaller storage space, can be a few orders of magnitude faster than the suffix tree.

Key words : DNA Sequence Database, Indexing, Trie

본 연구는 한국 과학재단의 목적 기초 연구 지원 사업(R04-2003-000-10048-0)과 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성, 지원 사업의 연구 지원으로 수행되었음.

* 한림대학교 정보통신공학부
** 연세대학교 컴퓨터과학과
*** 한양대학교 정보통신대학 정보통신학부, 교신저자

+ 논문접수 : 2005년 2월 24일, 심사완료 : 2005년 4월 14일

1. 서 론

DNA 시퀀스는 모든 생물의 생명 특성을 결정하는 코드로서 A, C, G, T의 네 가지 문자들로 구성된 스트링으로 표현된다. 분자 생물학에서 어떤 DNA 시퀀스를 해석하기 위한 가장 기본적인 방법은 그것과 염기 배열이 유사한 다른 DNA 시퀀스의 특성을 이용해 구조 및 기능 등을 유추하는 것이다[8].

DNA 시퀀스를 대상으로 하는 유사 검색(homology search) 문제는 다음과 같이 정의된다[24, 27]. DNA 시퀀스 데이터베이스 S와 질의 시퀀스 Q, 유사 허용치(tolerance) T가 주어지면, Q의 서브시퀀스 Q'과 유사 허용치가 T 보다 작거나 같은 서브시퀀스 S'을 S 내에서 검색해 낸다. 이때 Q'과 S'사이의 유사도를 계산하기 위하여 응용 목적에 적합한 유사도 함수(similarity function)를 정의 사용하게 되며, 에디트 거리(edit distance) 등을 예로 들 수 있다.

DNA 시퀀스 검색을 위하여 일반적으로 사용되는 방식은 데이터베이스 전체를 검색 대상으로 하는 순차 검색 기법이다[22]. BLAST[1, 2]는 DNA 시퀀스 검색 연산을 위하여 가장 널리 사용되는 표준 도구이다. BLAST는 휴리스틱스 기반 알고리즘에 의하여 고속의 시퀀스 검색 기능을 제공하지만, 검색 결과의 완전한 정확도를 보장하지는 못한다. 유사 서브시퀀스 검색 연산의 정확성을 보장할 수 있는 대표적인 알고리즘으로 Smith-Waterman 알고리즘[22]를 들 수 있다. 그러나, 이 방식은 많은 계산량을 필요로 하여 속도가 느리므로 실제적인 사용에 있어 제한적이다.

최근의 DNA 시퀀스 데이터베이스 규모의 급격한 증가 추세를 고려할 때, DNA 시퀀스 검색 연산을 보다 효과적으로 지원할 수 있는 인덱싱 및 질의 처리 기술이 요구된다. 접미어 트리(suffix tree)[23]는 DNA 시퀀스 검색을 위한 좋은 인덱스 구조로 알려져 왔다. 접미어 트리는 주어진 시퀀스의 접미어에 해당되는 서브시퀀스들을 트리 형태로 구성한 것으로서 질의와 정확히 일치하는 서브시퀀스를 고속으로 검색한다. 또한 DNA 시퀀스를 대상으로 하는 효율적인 유사 서브시퀀스 검색 알고리즘들이 제안되어 있다[6, 11, 15, 25]. 그러나 이 들 접미어 트리 기반의 유사 검색 알고리즘은 질의 시퀀스의 길이 혹은 유사 허용치 T의 크기가 커지는 경우, 인덱스의 검색 공간이 증가하여 성능이 저하하는 단점을 갖는다. 이와 같은 단점을 보완하기 위한 방법으로서, 참고 문헌 [18]에서는 주어진 질의 시퀀스를 작은 길이의 서브 질의로 분해하여 각 서브 질의에 대하여 작은 T 값을 대상으로 유사 검색을 수행한 후, 그 결과를 결합하는 하이브리드 인덱싱 기법을 제안하고 있다.

그러나 접미어 트리는 그 구조적 특성으로 인하여 여전히 다음과 같은 문제점을 갖는다.

① 저장 공간 : 접미어 트리는 일반적으로 매우 큰 저장 공간을 필요로 하여, 그 크기는 데이터베이스 크기의 수 십배에 이른다[9, 14, 18]. 예를 들어, 참고 문헌 [11]에 의하면, 286M의 DNA 시퀀스를 대상으로 구성된 접미어 트리의 디스크 상의 실제 크기는 19G 바이트 정도로 보고되어 있다.

② 검색 성능 : 디스크 액세스 패턴이 지역적이지 못하여 검색 성능이 떨어진다. 접미어 트리의 크기가 클수록 이 트리를 순회하는 시간이 길어지며, 따라서 전체 검색 성능이 떨어진다[20].

③ DBMS와의 통합 : 접미어 트리는 그 구조적 특성 상 페이지 단위로 구현하기가 어렵다. 따라서 모든 데이터를 페이지 단위로 구현하는 DBMS와 밀 결합(seamless integration)에 어려움이 있다[23, 26].

본 논문에서는 이와 같은 접미어 트리의 문제점들을 해결하는 DNA 시퀀스 검색을 위한 새로운 인덱스 구조를 제안하고 이를 기반으로 하는 효율적인 질의 처리 방식을 제안한다. 제안하는 인덱스는 이진 접미어 트라이를 기본 구조로 사용한다. 트라이(trie)[10, 23]를 기본 구조로 하여 포인터 없이 트라이를 비트 스트링으로 표현하는 방식을 채택하며, DNA 시퀀스로부터 가능한 모든 접미어를 추출하여 이를 인덱싱의 대상으로 한다. 유사 검색 알고리즘은 기본적으로 다이나믹 프로그래밍 기법을 사용하여 이진 접미어 트라이의 경로 상에 존재하는 모든 유사 서브시퀀스를 검색한다. 접미어 트라이 탐색 방식으로서 너비 우선 방식을 채택하여 인덱스의 각 페이지에 대하여 단 한 번의 디스크 액세스와 단 한 번의 순차적 처리에 의하여 모든 경로 상의 유사 서브시퀀스를 검색한다. 또한 트라이 운행에 있어 검색 대상의 단말 노드 정보를 신속히 찾기 위하여 별도의 다차원 인덱스 구조를 사용하여 검색 성능을 최적화한다.

다양한 실험을 통하여 제안된 유사 검색 기법의 성능을 기존의 방식과 비교하여 정량적으로 검증한다. 실험 결과에 의하면, 제안된 기법은 기존의 접미어 트리 기반의 유사 검색 방식과 비교하여 더 작은 저장 공간을 가지고도 수배에서 수십배까지의 검색 성능의 개선 효과를 가지는 것으로 나타났다.

2. 관련 연구

본 장에서는 DNA 시퀀스 검색에 대한 기존의 관련 연구에 대하여 요약한다.

순차적 검색을 기반으로 하는 대표적인 유사 서브시퀀스 검색 기법으로 Smith-Waterman(S-W) 알고리즘[22]를 들 수 있다. S-W 알고리즘은 다이나믹 프로그래밍(dynamic programming)기법에 의하여 두 시퀀스 S와 Q 사이에 최대 가능 스코어를 갖는 최적의 부분 정렬(optimal local alignment)을 찾는다. 그러나 이 방식은 두 시퀀스의 길이의 곱에 비례하는 계산량($O(|Q|\times|S|)$)를 필요로 하여 속도가 느린 단점이 있다.

BLAST[1, 2]는 현재 DNA 시퀀스 검색 연산을 위하여 가장 널리 사용되는 표준 도구로서 빠른 시간 내에 최적에 가까운(near optimal) 정렬을 얻는 효율적인 알고리즘으로 인정받고 있다. 그러나 BLAST는 관련 데이터베이스가 주기억 장치에 적재되어야 하며, 검색 속도가 데이터베이스 사이즈에 비례하고, 워드의 고정 길이에 따라 검색 결과의 정확도가 영향을 받는 점 등이 문제점으로 지적된다.

DNA 시퀀스 데이터베이스에 대하여 사전에 인덱스를 구성, 활용함으로써 DNA 시퀀스 검색의 속도 향상을 기대할 수 있다. 인덱스 기반의 DNA 시퀀스 검색 기법은 역 인덱스(inverted index) 방식[5, 7, 27], 다차원 인덱스 방식[12], 영속 트리 방식[11, 17, 20] 등으로 분류할 수 있다.

정보 검색 분야에 활용되는 역 인덱스(inverted index)를 기반으로 하는 인덱싱 기법으로 참고 문헌 [5, 7, 27]의 방식을 들 수 있다. 시퀀스 데이터베이스 내에서 일정 길이의 인터벌(interval)을 오버랩핑 시켜가며 추출하여 이 들을 워드로 하여 각 워드의 포스팅 리스트(출현 시퀀스 번호, 오프셋 정보 등 포함)를 구성하는 방식으로 속도 향상 효과를 얻을 수 있다. 참고 문헌 [27]에서는 특히 압축 인덱스 구조를 사용하여 인덱스 파일이 과다하게 커지는 단점을 보완하고 있으나, 검색 결과의 정확도가 떨어지는 점 등이 단점으로 지적되고 있다.

참고 문헌 [12]에서는 웨이블렛(wavelet) 변환에 의하여 시퀀스 데이터베이스 내의 각 서브시퀀스를 다차원의 정수 공간으로 매핑한 후, 이 들을 다차원 인덱스 구조로 표현하고, 영역 질의, 최근접 질의를 수행하는 새로운 방식을 제안하고 있다. 제안된 방식은 작은 사이즈의 인덱스 구조를 이용하여 질의 처리 시 데이터베이스의 검색 공간을 축소 시킬 수 있는 효율적인 방식으로 간주될 수 있으나, 유사도를 계산하기 위하여 에디트 거리(edit distance) 외의 일반적인 유사도 함수를 도입하기 어려운 점 등이 단점으로 지적되고 있으며, global alignment에 한정된 검색 기법이다.

영속 트리 기반의 대표적인 인덱싱 방식으로서 접미어 트리(suffix tree)[23]를 들 수 있다. 기존에는 디스크 상에 주기억장치 용량 이상인 대규모의 접미어 트리를 구성하는데 어려움이 있었으나, 참고 문헌 [11]에서 디스크 분할 저장 방식(partitioned suffix tree)에 의하여 이 문제를 해결한 바 있다. 또한 참고 문헌 [20]에서는 인메모리 방식의 압축 접미어 배열[16]을 이용한 유사 시퀀스 검색 방식을 제안하고 있다. 한편, 참고 문헌 [17]에서는 접미어 트리를 기본 인덱스 구조로 사용하여 다이나믹 프로그래밍 A*-탐색 기법[13]을 적용한 유사 검색 방식을 제안하고 있다. 제안된 방식은 질의 길이가 그리 길지 않은 질의 패턴에 대하여 유사도가 높은 순서로 적은 개수의 서브시퀀스만을 검색 대상으로 하는 경우에는 BLAST에 견줄만한 고성능을 보인다. 그러나 접미어 트리 인덱스는 그 구조적 특성으로 인하여 저장 공간의 오버헤드가 매우 크다는 점, 트리를 순회하는 시간이 길다는 점, 구조적 특성 상 페이지 단위로 구현하기가 어려워 DBMS와 밀 결합이 어렵다는 점 등의 문제점을 여전히 가지고 있다.

3. 인덱싱 방안

본 장에서는 대규모 DNA 시퀀스 데이터베이스를 대상으로 하여 효율적인 DNA 시퀀스 검색을 지원하기 위한 인덱싱 방안에 대하여 논의한다.

3.1 이진 접미어 트라이

트라이[10, 23]는 노드가 정보를 가지지 않고 에지에 데이터가 저장되는 트리 구조로서, 각 에지는 하나의 문자를 가지며 루트로부터 단말까지의 경로는 인덱싱의 대상이 되는 하나의 키에 해당된다.

트라이를 구현하는 가장 직접적인 방법은 각 노드를 $|\Sigma|$개의 포인터를 저장하는 배열로 나타내는 것이다. 여기서, Σ는 응용이 대상으로 하는 문자 집합, 즉 알파벳을 나타낸다. 이 방법을 사용하면 검색 연산을 효율적으로 수행할 수 있지만 노드에 널(NULL) 포인터가 많아져서 기억 공간을 낭비하는 경향이 있다. 일반적으로 $|\Sigma|$이 커질수록 기억 공간의 낭비가 커지며, 루트 근처에 있는 노드보다는 단말 근처에 있는 노드가 기억 공간을 더 많이 낭비하게 된다.

트라이의 노드를 링크된 리스트로 구현할 수도 있다. 즉, 각 노드에 바로 오른쪽 형제 노드에 대한 포인터와 가장 왼쪽 자식 노드에 대한 포인터를 저장함으로써 조건을 만족시키는 자식 노드로 손쉽게 이동할 수 있다. 이 방법에서는 꼭 필요한 기억 공간만이 노드에 할당되기 때문에 기억 공간의 낭비가 최소화되는 장점이 있지만, 조건을 만족시키는 자식 노드가 상수 시간(constant time)에 선택될 수 없기 때문에 검색 연산의 효율이 다소 저하되는 단점이 있다.

꼭 필요한 기억 공간만을 사용하면서도 검색 연산을 효율적으로 수행할 수 있는 방안으로 포인터 없는 이진 트라이(pointerless binary trie)[21]를 생각해 볼 수 있다. 포인터 없는 이진 트라이는 알파벳 Σ를 {0,1}로 제한하여 각 노드가 최대 2개의 에지를 가지도록 하며, 0의 값을 가지는 에지는 노드의 왼편에, 1의 값을 가지는 에지는 노드의 오른편에 연결하는 규칙을 적용하여 에지 정보를 생략 표현한다. 즉, 노드 당 두 비트를 할당하여 그 값이 '10'이면 노드에 왼쪽 에지만이 연결된 형태를 표현하고, '01'은 노드에 오른쪽 에지만이 연결된 형태를 표현하고, '11'은 노드에 왼쪽 에지와 오른쪽 에지가 모두 연결된 형태를 표현하고, '00'은 에지가 연결되지 않은 단말 노드의 형태를 표현한다.

본 연구에서는 포인터를 사용하지 않는 이진 트라이의 기본 개념을 활용하여 DNA 시퀀스 검색을 지원하는 인덱싱 구조를 제안한다. 이를 위해 DNA 시퀀스로부터 가능한 모든 접미어를 추출하여 이를 인덱싱의 대상으로 한다. 따라서 인덱스의 기본 구조로서 접미어 트라이[23] 구조를 사용한다.

접미어 트라이는 인덱싱 대상이 되는 접미어들이 많은 공통 접두어를 가질 때 좋은 압축 효과를 갖는다. DNA 시퀀스는 A, C, G, T 네 개의 문자로 구성된 매우 긴 시퀀스로 볼 수 있다. DNA 시퀀스로부터 추출된 접미어는 소수의 문자 집합으로 생성되는 특성에 의하여 많은 공통 접두어를 가질 확률이 높아지므로, DNA 시퀀스를 대상으로 접미어 트라이 인덱스를 생성하면 기본적으로 높은 압축 효과를 기대할 수 있다.

본 연구에서는 보다 높은 인덱스 압축 효율을 구현하기 위하여 DNA 시퀀스가 소수의 문자만으로 구성된 시퀀스라는 점에 착안하여 시퀀스 내에 출현하는 각 문자를 8비트가

문자	이진표현
$	000
A	001
C	010
G	011
N	100
T	101
S	110
Y	111

<그림 1> 각 출현 문자의 이진 표현 예

추출된 접미어 시퀀스	이진 변환된 접미어 시퀀스
S1: ACGT$	001010011101000
CGT$	010011101000
GT$	011101000
T$	101000
S2: ACT$	001010101000
CT$	010101000
T$	101000

<그림 2> DNA 시퀀스로부터 추출된 접미어 시퀀스의 이진 표현 예

아닌 최소의 비트량으로 표현한다. 실제의 DNA 시퀀스에는 A, C, G, T 네 개의 문자 외에 출현 빈도가 높지 않은 11개의 와일드 카드 문자가 출현할 수 있다. 예를 들어 N은 A, C, G, T 중 임의의 하나의 문자를 나타내며, B는 A가 아닌 C, G, T 중 임의의 하나의 문자를 의미한다. 따라서 예를 들어 인덱싱 대상의 DNA 시퀀스에 7개 이하의 서로 다른 문자가 출현하는 경우, 각 문자에 3비트를 할당하여, 이를 구별 표현할 수 있다. 이와 같은 방식에 의하여 DNA 시퀀스를 축약된 이진 시퀀스로 변환하여 접미어 트라이를 구성하면 공통 접두어가 생성될 확률이 더욱 높아지며, 따라서 생성된 트라이 인덱스 구조로부터 더욱 높은 압축율을 기대할 수 있다.

<그림 1>에 DNA 시퀀스에 출현하는 각 문자의 이진 표현 예를 보인다. 여기에서 사용된 '$' 문자는 각 접미어 시퀀스를 유일하게 구별하기 위하여 사용된 특수 문자를 나타낸다. 예를 들어, S1='ACGT'와 S2='ACT'의 두 개의 DNA 시퀀스에 대하여 각 시퀀스로부터 모든 접미어 시퀀스를 추출하여 <그림 1>의 방식에 의하여 각 문자당 3비트를 할당하여 이를 이진 시퀀스로 변환하면 <그림 2>와 같은 결과를 얻을 수 있다.

DNA 시퀀스를 위한 이진 접미어 트라이 인덱스 생성 알고리즘을 Algorithm 1에 보인다. 단, Algorithm 1은 DNA 시퀀스로부터 추출된 이진 접미어 시퀀스 집합을 입력으로 받는다고 가정한다.

인덱스 생성 과정을 예를 들어 설명하면 다음과 같다. 단계 1에 의하여 접미어 시퀀스를 오름차순으로 정렬하여 정렬 순서에 의하여 이를 삽입하는 과정은 이진 접미어 트라이 인덱스 구조를 단일 방향으로 증가시키는 효과를 가지며, 따라서 디스크 상의 페이지 기반 인덱스 구성을 가능하게 한다. 단계 2의 접미어 시퀀스의 삽입 과정을 예를 들어 설명하면 다음과 같다. <그림 3>은 <그림 2>의 접미어 시퀀스를 정렬하여, 정렬 순서에 따라 처음으로 S1 시퀀스의 접미어 'ACGT$'(001010011101000)를 삽입한 예를 보인다. 왼쪽 그림은 트라이를 노드 구조로 표현한 예이고, 오른쪽 그림은 실제의 이진 데이터 저장

Algorithm 1: Index construction

Input : set of binary suffix sequences S

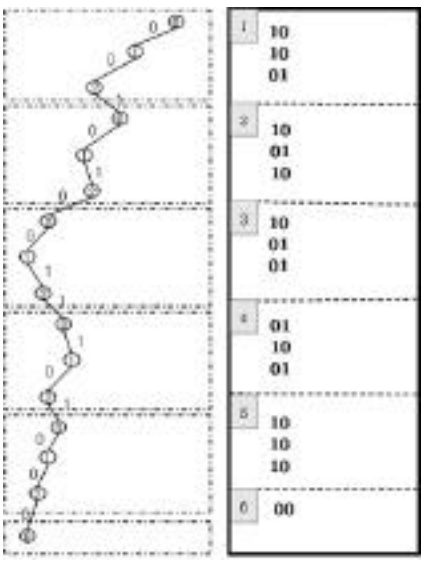
Output : binary suffix trie I

단계 1 : 이진 접미어 시퀀스 S를 오름차순으로 정렬한다.
단계 2 : 정렬된 이진 접미어 시퀀스를 다음 과정에 의하여 순서대로 삽입한다.

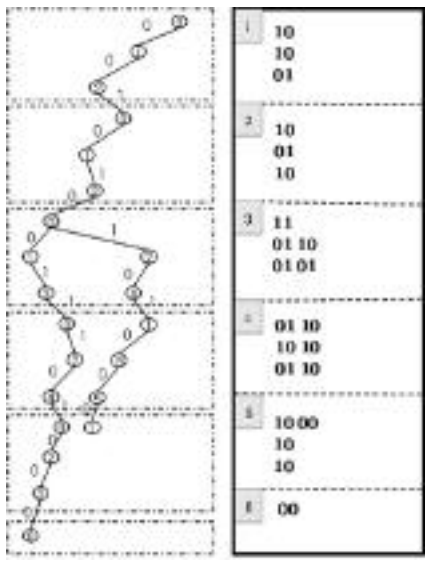
①새로이 생성 또는 변경되는 노드 구조를 2비트 노드 정보 표현하여 해당 페이지 영역에 기록한다. 이때 한 페이지에 저장할 최대 노드 레벨 수 혹은 최대 노드 수에 의하여 페이지 크기가 결정된다.

②새로이 삽입되는 노드에 의하여 해당 페이지 영역 내에 오버플로우가 발생할 가능성이 있으면, 이 노드를 제외한 나머지 해당 페이지 영역을 디스크에 기록하고, 기록되지 않은 노드 정보로 해당 페이지 영역을 재구성한다. 단, 디스크에 해당 페이지 영역을 기록할 때, 노드와 노드 사이의 페이지 연결 상태를 나타내는 페이지 정보를 함께 기록한다.

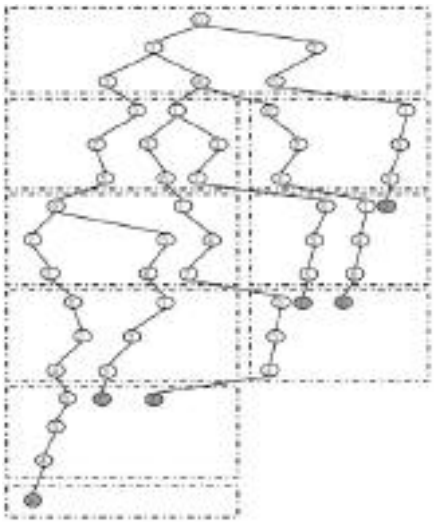
방식을 나타낸다. 다음의 <그림 4>는 S2 시퀀스의 접미어 ‘ ACT$(001010101000)를 추가로 삽입한 결과를 보인다. 이와 같이 트라이 구조에 새로운 임의의 길이의 이진 시퀀스를 삽입하면 새로이 삽입되는 시퀀스의 각 비트 정보는 트라이 노드 구조의 기존 경로를 따라 가거나, 새로운 에지를 생성하여 새로운 노드를 형성한다. 이와 같이 형성된 노드 구조는 2



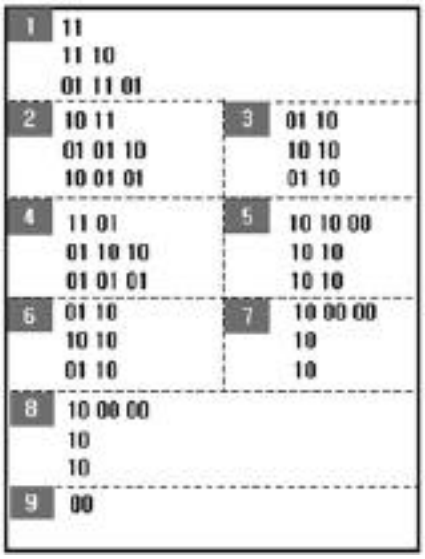
<그림 3> 접미어 트라이에 S1 ≒ ACGT$삽입



<그림 4> 접미어 트라이에 S2 ≒ ACT$삽입



<그림 5> 접미어 트라이의 구성 예



<그림 6> 접미어 트라이의 내부표현 예

비트 노드 정보로 변환되어 이진 데이터로 저장된다. <그림 5>는 <그림 2>의 예에서 보인 접미어 시퀀스를 모두 삽입한 결과의 트라이 구조를 나타내며, <그림 6>은 <그림 5>의 이진 트라이 구조를 이진 데이터로 표현한 저장 예를 나타낸다.

Algorithm 1은 디스크 기반의 인덱스 구성 알고리즘이다. 생성되는 이진 데이터 표현의 이진 트라이는 초기에는 주기억 장치에 생성되어 페이지 크기를 초과하면 바로 디스크에 쓰이게 된다. 단, 트라이 구조를 디스크에 분할 저장할 때 각 디스크 페이지에 저장할 최대 노드 수를 미리 정하여 오버플로가 발생하지 않도록 한다. <그림 3, 4, 5, 6>의 예에서 보이는 점선 영역은 디스크 상의 페이지 분할 영역을 나타낸다. 또한 <그림 3, 4, 5>에서 각 노드에 할당된 노드 번호는 디스크 페이지에 저장되는 노드의 저장 순서를 나타낸다. 예를 들어 페이지 크기를 16비트로 가정하여, 한 페이지에 저장할 수 있는 최대 노드 레벨 수를 3으로, 최대 노드 수를 8로 가정한 경우, 그림에서 보인 바와 같은 디스크 분할 결과를 얻게 된다. <그림 3>의 예에서 각 2비트의 노드 정보는 페이지의 최대 노드 레벨 수에 의하여 최대 3개씩 각 페이지에 저장됨을 볼 수 있다. 이후 각 페이지에 노드 정보가 추가되어 최대 노드 수를 초과하게 되면 각 페이지는 분리 저장되어 <그림 5>와 같은 페이지 분할 결과를 얻게 된다. 또한 여기에서 주의할 점은 트라이 구조를 페이지에 저장하는 경우, 한 페이지에 저장된 임의의 노드의 자식 노드가 다른 페이지에 나뉘어 저장되지 않도록 보장한다. 즉, 트라이를 구성하는 에지는 각 페이지의 수직 경계를 통과하여 생성될 수 있으나, 페이지의 수평 경계를 통과해서는 생성되지 못한다. 이 가정은 질의 처리 과정의 효율

# Page	Top	Bottom	Node	Addr
1	0	0	6	84
2	0	0	8	54
3	2	3	6	108
4	0	0	8	24
5	2	3	7	132
6	0	0	6	0
7	2	2	5	159
8	0	0	5	180
9	0	0	1	201

<그림 7> 페이지 테이블

을 높이기 위하여 필요하다.

이와 같이 접미어 트라이 인덱스는 디스크 페이지에 분할되어 저장되므로, 이 인덱스를 이용하여 임의의 경로를 검색하기 위해서는 노드와 노드 사이의 페이지 연결 상태를 나타내는 정보가 필요하다. 즉, 임의의 페이지에 저장되어 있는 노드로부터 직접 연결된 다음 노드가 어느 페이지에 저장되어 있는지를 알기 위해서는, 노드와 노드 사이의 페이지 연결 상태를 나타내는 정보가 부가적으로 필요하다. 이와 같은 페이지 연결 정보는 각 페이지에 유입된 에지의 수, 각 페이지로부터 나간 에지의 수 등으로부터 계산될 수 있다. 예를 들어 <그림 6>의 인덱스 구조의 페이지 교체를 위해서는 <그림 7>과 같은 페이지 정보가 필요하다. 여기에서 # page는 페이지의 번호를 나타내고, Top는 현재 페이지 레벨의 이전 페이지까지 유입된 에지의 총 수를 나타낸다. 또한, Bottom은 현재 페이지 레벨의 이전 페이지들로부터 나간 에지의 총 수를 나타내고, Node은 각 페이지에 기입된 노드 수를 나타내며, Addr은 각 페이지의 실제 기입 주소를 나타낸다.

3.2 단말 노드 저장 방안

접미어는 시퀀스 정보(시퀀스 식별자와 시퀀스 내의 오프셋)에 의해서 유일하게 구별된다. 접미어를 접미어 트라이에 저장할 때 접미어의 시퀀스 정보는 일반적으로 단말 노드에 저장된다. 그러나 제안되는 접미어 인덱스 상에서 각 노드는 2비트로 표현되므로 접미어의 시퀀스 정보는 접미어 인덱스와는 별도로 저장되어야 한다.

제 4장에서 기술하는 접미어 트라이 기반의 질의 처리 방식에서는 질의 시퀀스와 유사한 임의의 경로를 검색한 후, 해당 결과의 시퀀스 정보를 얻기 위하여 현재 노드 이하의 서브 트라이를 검색하여 모든 단말 노드와 단말 정보를 검색하는 과정을 필요로 한다. 그러나 이와 같은 단말 정보의 검색 과정은 인덱스 크기가 방대하고, 특히 질의 시퀀스의 길이가 짧은 경우, 검색 대상의 인덱스 공간이 매우 커질 수 있다는 단점이 있다.

본 연구에서는 단말 정보를 보다 효율적으로 검색하기 위하여 다차원 인덱스를 이용한 새로운 단말 정보 저장 방식을 사용한다. 다차원 인덱스 저장 방식에서는 접미어 시퀀스를 이진 비트 패턴으로 변환한 후, 이 값을 다차원 키 값으로 사용하여 단말 정보를 저장한다. 단, 이진 비트 패턴으로 변환된 접미어 시퀀스는 그 길이가 각각 다르므로, 다음과 같은 방식을 사용하여 각 시퀀스를 정해진 수 k개의 정수로 표현하여 k 차원의 다차원 인덱스 키 값으로 사용한다.

① 시퀀스의 이진 비트 패턴이 k개의 정수 표현보다 짧을 때 : 해당 비트 패턴 뒤에 0을 k개의 정수 표현까지 계속 붙여 표현한 후, 이를 k 차원의 인덱스에 저장한다.
② 시퀀스의 이진비트 패턴이 k개의 정수 표현보다길 때: 해당 비트표현이 k개의 정수 표현까지로 표현될 수 있도록 k개의 정수 표현 이후의 뒷부분을 잘라 버린 후, 이를 k 차원의 인덱스에 저장한다.

4. 질의처리 방안

본 장에서는 제 3장에서 제안한 인덱싱 기법을 이용한 유사 검색 기법에 대해 논의한다.

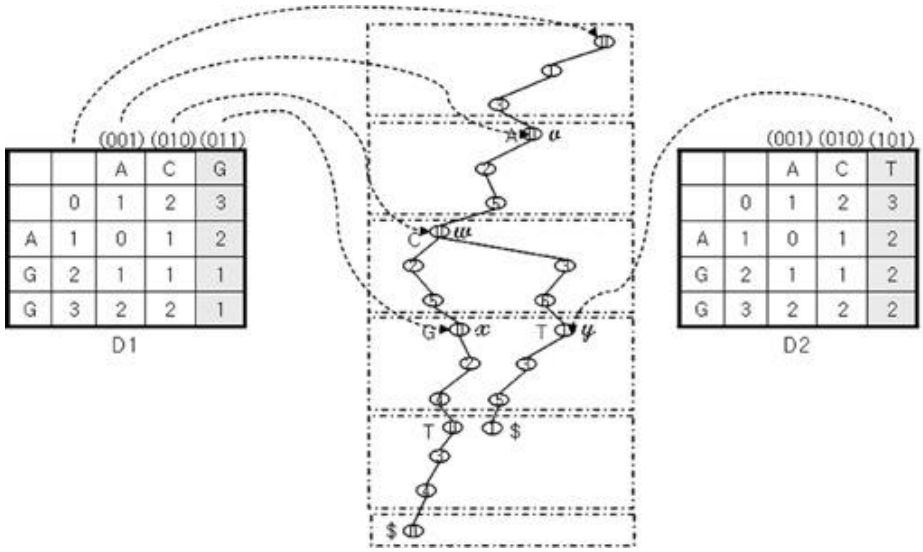
4.1 유사 검색

유사 검색을 위하여 가장 기본적으로 사용되는 방식은 다이나믹 프로그래밍(DP) 기법이다. 비교 대상의 두 시퀀스 S와 Q에 대하여 DP 기법은 |Q|+1 개의 행과 |S|+1의 열을 갖는 2차원의 DP 테이블을 생성하여 두 시퀀스 사이의 최적의 거리(optimal distance)를 얻는다. 이 때 유사도를 계산하기 위하여 응용에 적합한 유사도 함수가 정의되어야 하며, 일반적으로 에디트 거리(edit distance) 등이 유사도 함수로 사용된다[11, 23].

접미어 트리에서는 경로 운행에 의하여 모든 서브시퀀스를 발견할 수 있다. 접미어 트리 기반의 유사 검색 알고리즘에서는 접미어 트리를 루트 노드로부터 깊이 우선 방식으로 운행하며 질의 시퀀스와 루트로부터 시작된 각 경로 상의 접미어 시퀀스에 대하여 DP 테이블을 생성함으로써 모든 유사 서브시퀀스를 검색한다[11, 17, 18, 25].

이진 접미어 트라이를 이용한 유사 검색을 위하여 접미어 트리 기반의 기존 방식을 그대로 적용하여 사용할 수 있다. 그러나 제안된 접미어 트라이 인덱스는 2비트의 노드 정보만을 저장할 뿐 노드와 노드를 연결하는 포인터 정보, 노드 레벨 정보, 경로 상의 해당 서브시퀀스 정보를 별도로 저장하지 않는다. 따라서 인덱스 검색을 위하여는 해당 페이지의 노드 정보를 순차적으로 읽어 이 들의 정보를 얻어야 한다.

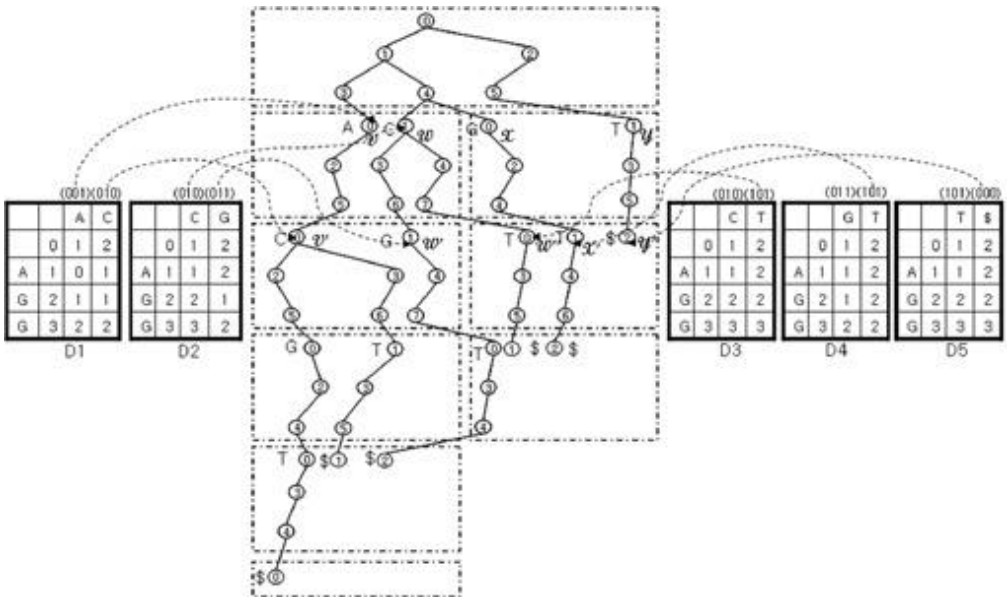
기존의 깊이 우선 방식의 트라이 운행에 의한 유사 검색 과정을 예를 들어 설명하면 다음과 같다. 제 3장에서 예제로 사용한 접미어 트라이 인덱스를 이용하여 질의 시퀀스 'AGG'와 에디트 거리 k가 1 이하인 유사 서브시퀀스를 검색하는 과정의 일부분을 그림 8에 보인다. 접미어 트라이 인덱스에서는 3비



<그림 8> 깊이 우선 방식의 이진 접미어 트라이 운행 예

트가 하나의 문자를 나타내므로 트라이 인덱스의 경로를 따라 3개의 노드를 연속 방문 한 후, 기존의 DP 테이블에 새로운 열이 첨가되게 된다. 즉, <그림 8>의 예에서 v, w, x의 노드에 도달하게 되면 각각‘ A(001)’,‘ C(010)’,‘ G(011) ’의 문자가 DP 테이블에 첨가되어 D1과 같은 결과 테이블이 얻어지게 된다. DP 테이블에 새로운 열이 첨가될 때마다 새로이 첨가된 열의 마지막 요소 값이 주어진 유사 허용치보다 작거나 같은지의 조건이 만족되는지를 평가한다. 만약 조건이 만족되는 경우에는 현재 방문한 노드 이하의 모든 단말 노드가 유사 검색 결과로 주어지게 된다. <그림 8>의 D1에서 현재 쌓은‘ G 열의 마지막 요소 값이 1이므로 에디트 거리가 1인 서브시퀀스가 검색되었음을 알 수 있다. 따라서 x 노드 이하의 모든 단말 노드가 유사 검색 결과로 주어지게 된다. 다음, 노드 탐색은 w 노드로 복귀되어 y 노드를 방문하게 된다. D2는 y 노드까지의 오른쪽 에지인 ‘ T(101)'의 경로에 대하여 생성된 DP 테이블을 나타낸다. 여기에서 D2 테이블의‘ A ,‘ C 열은 앞에서 작성된 D1의‘ A ,‘ C 열과 공유되어 재사용된다. D2에서는 현재 쌓은‘ T 열의 모든 요소 값이 1 보다 크므로 이 후의 경로에는 에디트 거리가 1 이하인 서브시퀀스가 존재하지 않음을 알 수 있으며, 따라서 이 이상의 경로 검색을 중지한다. 이 후 트라이 검색은 루트 노드로부터 시작되는 ‘ CGT(10011101)’ 경로 검색을 수행하며, 이와 같은 과정에 의하여 트라이 상의 대상 경로를 모두 검색하게 된다.

제 3.1절에서 보인 바와 같이 이진 접미어 트라이는 포인터를 사용하지 않는 2비트의 노드 정보만을 저장하고 있다. 또한 동일 노드 레벨의 노드 정보는 같은 페이지에 연속적으로 배열되는 특성을 갖는 디스크 기반의 인



<그림 9> 너비 우선 방식의 이진 접미어 트라이 운행 예

덱스 구조이다. 따라서 이와 같이 깊이 우선 방식에 의하여 트라이 상의 모든 경로를 검색하는 경우, 동일 페이지 내의 노드가 반복적으로 액세스 되며, 또한 디스크 상의 동일 페이지가 반복적으로 액세스 되는 단점이 있다.

이와 같은 문제점을 해결하기 위하여 본 연구에서는 너비 우선의 이진 접미어 트라이 운행 방식을 사용한 유사 검색 방식을 제안한다. 우선, 직관적인 이해를 위하여 <그림 8>과 같은 예제에 대하여 접미어 트라이를 너비 우선 방식에 의하여 운행하는 과정을 다음에 보인다. 본 방식에서는 <그림 9>에 보이는 바와 같이 루트 노드로부터의 순차적으로 v, w, x, y의 노드를 방문하여 서로 다른 4개의 DP 테이블을 각각 생성한다. 다음 v', w', w'', x'', y'' 의 노드를 방문하여 각각의 경로에서 얻어진 서브 시퀀스 'AC(001010)', 'CG(010011)', 'CT(010101)', 'GT(011101)', 'T$(101000)'에 대하여 5개의 DP 테이블 D1, D2, D3, D4, D5를 생성되게 된다. 여기에서도 D3 테이블의 'C' 열은 앞에서 작성된 D2의 'C' 열과 공유되어 재사용됨을 알 수 있다. 이와 같은 과정에 의하여 이 후 트라이 상의 대상 경로를 너비 우선 방식으로 모두 순차 검색하게 된다. 따라서 본 방식에 의하여 트라이를 너비 우선 방식으로 운행하면, 각 디스크 페이지에 존재하는 노드 정보를 순차적으로 처리하게 되어 각 디스크 페이지는 최대 한번만 액세스 되는 효율적인 검색이 가능함을 알 수 있다.

아래에 명시된 알고리즘 Search-Trie는 너비 우선 방식에 의하여 접미어 트라이 인덱스 I를 탐색하여 질의 Q를 만족하는 유사 서브시퀀스들을 검색하는 알고리즘을 나타낸다. 여기에서 질의 Q는 질의 시퀀스를 이진 비트열

Algorithm 2: Search-Trie

```
Input : index I, query sequence Q, page table P, rtree R
Output : set of answers
1. enQueue(Q_pagenumber, Root_pageNumber);
2. enQueue(Q_node[Root_pageNumber], RootNode);
3.
4. while !isEmpty(Q_pagenumber) do {
5.   pageNumber = deQueue(Q_pagenumber);
6.
7.   while !isEmpty(Q_node[pageNumber]) do {
8.       currentNode = deQueue(Q_node[pageNumber]);
9.
10.      for each child node CNi of the current_Node do {
11.          moreVisit := TRUE,
12.          AppedBitString(CNi_Path, currentNode, CNi);
13.
14.          if BitCount(CNi_Path) mod 3 == 0 {
15.              DPT_CNi = AddColumn(currentDPT, newAdded_path),
16.              Let dist be the last row value of the new added column;
17.
18.              if dist <= k then FindAnswer(CNi, R),
                     moreVisit := FALSE;
19.              else moreVisit := FurtherVisit(DPT_CNi);
20.          }
21.
22.          if moreVisit {
23.              newPageNumber := getPageNumber(CNi, P);
24.
25.              if newPageNumber != pageNumber
26.                and isEmpty(Qnode[newPageNumber]) then
27.                  enQueue(Qpagenumber, newPageNumber);
28.
29.                  enQueue(Qnode[newPageNumber], CNi);
30.          }
31.      }
32.  }
33. }
```

로 변환한 것임에 유의하여야 한다. 또한 트라이 인덱스의 페이지 분할에 관련된 정보는 페이지 테이블 P에 저장되어 있다고 가정한다.

Search-Trie의 동작 과정을 간단히 설명하면 다음과 같다. 각 페이지와 페이지에 저장된 노드를 순차적으로 탐색하기 위하여 다음 2가지의 Queue 구조를 이용한다. Q_pagenumber는 탐색하게 될 페이지 정보를

담고 있으며, Q_node는 각 페이지에서 탐색될 노드 정보를 담고 있다. 우선, 트라이 인덱스의 루트 페이지의 번호를 Q_pagenumber에, 루트 페이지에 대한 Q_node에 루트 노드를 각각 Enqueue한다(Line 1-2). 전체 알고리즘은 탐색 대상의 페이지와 탐색 대상의 노드들에 대한 이중의 while 문으로 이루어져 있다. 외부 while 문(Line 4-33) 내의 Line 4는 탐색해야할 페이지 번호를 Q_pagenumber로부터 Dequeue하는 과정을 나타낸다. 내부 while 문(Line 7-21) 내의 Line 7-8에서는 Dequeue된 페이지 번호의 Q_node로부터 탐색해야 할 노드 번호를 Dequeue하여 현재 탐색 노드 current_Node로 한다. 다음, 현재 탐색 노드인 current_Node의 모든 차일드 노드들에 대하여 검색을 수행한다(Line 11-21). Line 11에 사용된 moreVisit는 이 후의 노드 검색을 계속 수행할지의 여부를 결정하는 변수로서 우선 그 값을 TRUE로 설정한다. Line 12의 AppendBitString(은 차일드 노드 CNi의 방문에 의한 경로 정보를 추가하는 함수로서 루트로부터 current_Node까지 기존에 형성된 경로에 current_Node로부터 차일드 노드까지의 경로를 추가한다. 이렇게 만들어진 경로 정보의 길이가 3의 배수라면(Line14), 하나의 새로운 문자가 첨가된 것을 의미하므로 새로운 문자의 경로 정보를 이용하여 DP 테이블에 새로운 열을 추가하게 된다. Line 15의 함수 AddColumn(current_DPT, newAdded_path)이 이 일을 수행하며, current_Node까지 생성된 DP 테이블 current_DPT에 근거하여 DP 테이블에 새로운 열을 추가하는 과정을 담당한다. 여기에서 DP 테이블의 새로운 열의 마지막 요소 값을 dist로 한다(Line 16). 만약 dist의 값이 주어진 유사 허용치 k 보다 작거나 같으면 유사 서브시퀀스가 검색되었으므로, 함수 FindAnswer()를 호출하여, 해당 노드 CNi의 서브 트라이에 존재하는 모든 단말 노드로부터 시퀀스 정보(시퀀스 번호, 오프셋)를 얻게 된다. 그 후 그 경로에 대한 노드 탐색을 중지하기 위하여 moreVisit을 FALSE로 설정한다. 만약 dist의 값이 k 보다 큰 경우에는 함수 FurtherVisit()에 의하여 이 경로에 대한 노드 탐색을 계속할 것인지, 중지할 것인지의 여부를 결정한다. moreVisit의 값이 TRUE라면 차일드 노드를 Q_node에 Enqueue하여 그 경로에 대한 노드 탐색을 지속하도록 설정한다. Line 23-27의 과정은 차일드 노드가 현재 탐색 중인 페이지와 다른 페이지에 존재하는 경우를 위한 처리로서 새로운 페이지 번호를 Q_pagenumber에 Enqueue하고, 새로운 페이지의 Q_node에 차일드 노드를 Enqueue한다.

4.2 다차원 인덱스를 이용한 단말 노드 정보 검색

제 4.1절에서 보인 Search-Trie 알고리즘에서는 질의 처리 과정 중, 접미어 트라이의 중간 노드 N에서 유사 서브시퀀스가 검색되면, 해당 노드 N의 서브 트라이에 존재하는 모든 단말 정보를 가져오는 과정이 필요하다. 바로 Find_Answer() 함수가 단말 정보를 검색하는 역할을 담당한다. 제 3.2절에서 제안한 다차원 인덱스는 서브 트라이의 중간 노드 검색 과정 등을 필요로 하지 않고 직접적인 단말 정보의

검색을 가능하게 한다. 다차원 인덱스는 접미어 트라이를 이용하여 유사 서브시퀀스의 검색을 완료한 후, 직접 단말 정보를 검색해 내는 방식으로서, 단말 정보의 검색 방식은 다음과 같다. 트라이 인덱스의 루트로부터 중간 노드 N을 연결하는 경로 p가 유사 서브시퀀스로서 검색되면, 그 경로의 길이에 따라 다음 3가지 중 하나를 선택하여 질의를 처리함으로써, 단말 정보를 얻을 수 있다.

① 경로 p의 길이가 k개의 정수 표현 보다 짧을 때 : 경로 p 뒤에 0을 계속 붙여 만든 k-정수 값을 p0라고 하고, p뒤에 1을 계속 붙여 만든 k-정수 값을 p1이라고 하자. 다차원 인덱스를 검색하여 p0와 p1 사이에 존재하는 모든 단말 노드들을 검색한다.
② 경로 p의 길이가 정확하게 k개의 정수 표현일 경우 : 다차원 인덱스를 검색하여 경로 p를 k-정수로 표현한 값을 가지는 모든 단말 노드들을 검색한다.
③ 경로 p의 길이가 k개의 정수 표현 보다 길 때 : 경로 p를 k-정수로 표현한 값을 pt 라고 하자. 다차원 인덱스를 검색하여 pt 값을 가지는 모든 단말 노드들을 검색한다. 그 후, 후처리 과정에 의하여 잘못 검색된 단말 노드를 제거한다.

5. 성능 평가

본 장에서는 실험에 의한 성능 분석을 통하여 제안하는 기법의 우수성을 규명한다. 본 실험에서는 GenBank[19]로부터 다운 받은 human chromosome 18, 19, 21의 시퀀스를 이용한다. 질의 시퀀스는 실험 데이터 셋으로부터 랜덤 방식에 의하여 추출한 것을 사용한다.

실험에 사용된 이 들 DNA 시퀀스에는 기본적으로 Σ = {A, C, G, T}의 네 종류 문자가 출현하며, 출현 빈도가 높지 않은 N, S, Y 등의 와일드 카드 문자가 출현한다. 실험에 사용된 각 DNA 시퀀스에는 최대 7개의 서로 다른 문자가 출현하고 있다. 따라서 본 실험에서는 서로 다른 접미어 시퀀스의 구별을 위하여 사용되는‘ $ ’문자를 포함하여 8가지 문자를 처리 대상으로 한다. 실험을 위한 하드웨어 플랫폼으로는 Windows 2003 Server를 운영체제로 사용하고, 1GB의 주기억 장치, 200GB 디스크를 갖는 Pentium IV 3.2GHz의 PC를 사용한다.

제안된 기법의 성능을 평가하기 위하여 인덱스의 크기 및 질의 처리 시간을 기존 방식과 실험을 통하여 비교 분석한다. Trie는 본 논문에서 제안한 트라이 인덱스 검색 방식으로서, 단말 노드 검색을 위하여 별도의 다차원 인덱스를 사용하는 방식이다. 다차원 인덱스로는 현재 시퀀스 데이터베이스 분야에서 널리 사용되고 있는 R*-트리[3]를 사용하여 실험한다. 사용된 R*-트리는 Maryland 대학의 Faloutsos 교수 팀에서 개발한 R*-tree Version 2.0을 사용하며, 본 실험에서는 단말 정보를 2차원 데이터로 표현하여 저장한다. 성능 비교를 위한 기존의 인덱싱 기법으로서 접미어 트리를 이용한 검색 방식인 Suffix를 사용한다. Suffix에서는 참고 문헌 [4]에서 제안된 incremental disk-based 접미어 트리 생

성 방식을 사용하며, 생성된 접미어 트리는 각 노드 당 최대 32바이트를 필요로 한다.

실험 1 (인덱스 크기 비교)

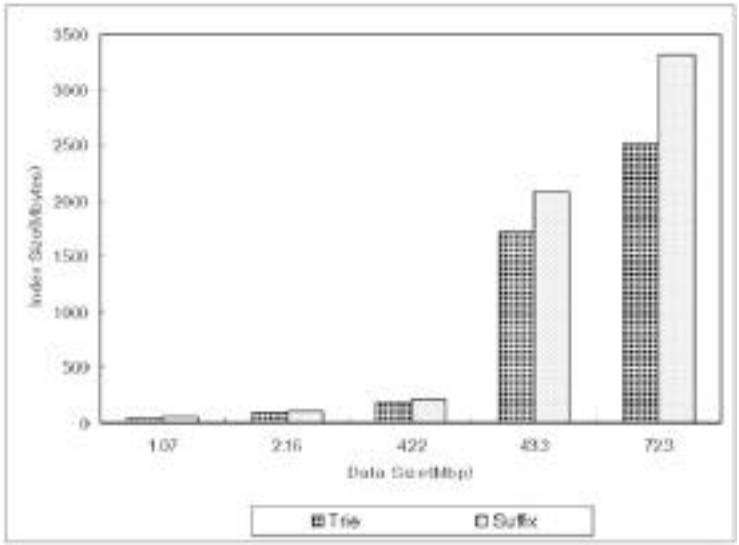
먼저 실험 1에서는 본 논문에서 제안한 Trie의 인덱스 크기를 Suffix와 비교, 평가한다. <그림 10>에 데이터 크기의 변화에 따른 인덱스의 크기 변화를 비교한 결과를 보인다. 본 실험에서는 페이지 크기로 4K를 설정하였다. Suffix의 인덱스는 단일 인덱스 구조로서 내부 노드와 단말 노드로 구성된다. Trie의 인덱스는 이진 접미어 트라이 인덱스와 페이지 테이블, 단말 정보 검색을 위하여 별도의 R-트리 인덱스로 구성된다.

실험 결과로부터 Suffix, Trie의 두 방식 모두 데이터의 크기 변화에 따라 거의 선형적으로 인덱스 크기가 비례하여 증가함을 알 수 있다. 그러나, 절대 크기를 비교 할 경우 Trie는 Suffix에 비하여 약 24%까지의 저장 공간 감소 효과를 나타내는 것으로 나타났다.

실험 2 (유사 서브시퀀스 검색을 위한 질의 처리 시간 비교)

실험 2에서는 제안된 Trie 인덱스를 사용한 유사 검색 알고리즘의 질의 처리 시간을 기존 방식과 비교한다. 유사 검색은 주어진 질의 시퀀스와 에디트 거리가 k 이하인 유사 서브시퀀스를 검색하는 방식을 사용한다. 실험 데이터로는 43.3Mbp의 human chromosome 21 시퀀스를 사용하였으며, 질의 길이 변화에 따르는 평균 질의 처리 시간을 비교한다. Trie를 사용한 유사 서브시퀀스 검색 방식으로 깊이 우선 탐색 방식 Trie-Depth와 너비 우선 탐색 방식 Trie-Breadth를 각각 구현하여 그 성능을 기존의 Suffix와 비교한다.

<표 1>은 주어진 질의와 에디트 거리가 1이



<그림 10> 데이터 크기 변화에 따른 인덱스 크기 비교

<표 1> 에디트 거리 1이하인 서브시퀀스 검색 시간

Query Length	# Result	Query Processing Time(msec)		
		Suffix	Trie-Depth	Trie-Breadth
6	388,321	9962.2 (6074.2)	1152.2 (488.7)	980.2 (470.2)
8	33,422	1800.5 (517.2)	704.1 (278.5)	333.5 (272.3)
10	3,966	1301.4 (61.2)	931.7 (217)	264 (212.7)
15	22	1224.4 (0)	1062.8 (3.6)	74.7 (2.3)
30	4.1	1174.2 (0.1)	1096.8 (2.6)	90.7 (1.9)

<표 2> 에디트 거리 2이하인 서브시퀀스 검색 시간

Query Length	# Result	Query Processing Time(msec)		
		Suffix	Trie-Depth	Trie-Breadth
6	3,772,995	175668.2(78113.3)	11048.8(4835.5)	9640.9 (4778)
8	524,469	22829.8 (8855.5)	10863.4 (5140)	6040.8 (5178)
10	80,740	4043.9 (1185)	16076.7 (4412)	4891.2(4313.5)
15	296	2808.5 (1.9)	19332.2 (54.5)	889.6 (49.2)
30	18	2816.3 (0)	32299.5 (5)	938.6 (5.2)

하인 모든 유사 서브시퀀스를 검색하기 위한 검색 시간을 나타내며, <표 2>는 에디트 거리가 2이하인 모든 유사 서브시퀀스를 검색하기 위한 검색 시간을 나타낸다. 질의 처리 시간은 유사 서브시퀀스의 시퀀스 번호와 오프셋 값을 반환하는 총 시간을 의미하며, 괄호 안의 값은 이 중 단말 노드 검색 시간을 측정한 값을 나타낸다. 본 연구에 서는 참고문헌 [18]의 연구 결과를 따라 비교적 짧은 질의 길이와 작은 유사 허용치 값을 갖는 유사 검색의 경우를 실험 대상으로 하였다.

실험 결과로부터 Suffix는 질의 길이가 짧은 경우, 단말 노드 검색을 위한 시간으로 인하여 질의 처리 시간이 증가하고 있음을 볼 수 있다. 한편, Trie 방식은 R-트리 인덱스를 사용한 단말 노드 검색에 의하여 비교적 안정된 단말 정보 검색 성능을 보이고 있음을 알 수 있다. 그러나 Trie-Depth의 경우, 질의 길이가 길어져 검색 공간이 증가하면 비트 연산에 의한 경로 검색 시간이 증가하여 검색 시간이 점차적으로 증가하고 있음을 알 수 있다. 반면에 제안된 Trie-Breadth는 질의 길이가 길어져도 경로 검색 시간이 크게 증가하지 않으며, 단말 정보 검색 시간의 급격한 감소

로 인하여 질의 길이 증가에 따라 오히려 검색 성능이 좋아지고 있다. 실험 결과에 의하여 제안된 Trie-Breadth는 질의 길이, 유사 허용치 변화에 따른 성능 비교에 있어 Suffix, Trie-Depth에 비하여 우수한 성능을 보임을 알 수 있다.

6. 결 론

본 논문에서는 효율적인 DNA 시퀀스 검색을 위한 인덱스 구조와 유사 검색 알고리즘을 제안하였다. 제안된 인덱스는 포인터 없이 트라이를 비트 스트링으로 표현하는 구조를 기본 구조로서 갖는다. 이러한 구조의 장점은 저장 공간의 오버헤드를 크게 줄일 수 있다는 점과 전체 구조를 페이지 기반으로 형성시킬 수 있다는 점이다. 또한, DNA 시퀀스 검색 시 검색 대상이 되는 트라이의 단말 노드를 신속하게 찾을 수 있도록 다차원 인덱스 구조를 보조 자료 구조로서 사용한다. 따라서 제안된 인덱스 구조를 사용함으로써 기존의 접미어 트리가 가지던 저장 공간, 검색 성능, DBMS와의 통합 등의 문제점들을 모두 해결할 수 있다.

또한, 제안된 인덱스를 이용하여 유사 검색을 효과적으로 처리하는 질의 처리 알고리즘을 제시하였다. 유사 검색 알고리즘은 다이나믹 프로그래밍 기법에 근거하여 이진 접미어 트라이를 루트로부터 너비 우선(breadth-first) 방식으로 운행하며 경로 상에 존재하는 유사 서브시퀀스를 효율적으로 검색해낸다.

제안된 기법의 우수성을 검증하기 위하여, 실험을 통한 성능 평가를 수행하였다. 실험 결과에 의하면, 제안된 인덱스는 기존의 접미어 트리와 비교하여 더 작은 저장 공간을 가지고도 기존의 접미어 트리에 비하여 3배에서 18배까지의 검색 성능의 개선 효과를 나타내는 것으로 나타났다.

참 고 문 헌

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," Journal of Molecular Biology, Vol. 215, No. 3, pp. 403-410, 1990.

[2] S. Altschul, T. Madden, A. Schaffer, J. Zhang, W. Miller, and D. Lipman, "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs," Nucleic Acids Research, Vol 25, No. 17, pp. 3389-3402, 1997.

[3] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," In Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 322-331, 1990.

[4] P. Bieganski, J. Riedl, J. V. Carlis, "Generalized suffix trees for biological sequence data: applications and implementation," In Proceedings of Hawaii International Conference on System Sciences, Vol. 5, pp. 35-44, 1994.

[5] A. Califano and I. Rigoutso, "FLASH:

A Fast Look-up Algorithm for String Homology,"In Proceedings of Intelligent System Conference for Morecular Biology, pp. 56-64, 1993.

[6] A. L. Delcher, S. Kasif, R. D. Fleischmann, and J. Peterson, O. White, and S. L. Salzberg," Alignment of whole genomes," Nucleic Acids Research, 27, pp. 2369-2376, 1999.

[7] C. Fondrat and P. Dessen," A Rapid Access Motif database(RAMdb) with a search algorithm for the retrieval patterns in nucleic acids or proteun databanks," Computer Applications in the Biosciences. Vol. 11, No.3, pp. 273-279, 1995.

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

[9] R. Giegerich, S. Kurtz, and J. Stoye, " Efficient Implementation of Lazy Suffix Trees," Softw. Pract. Exp., Vol 33, pp. 1035-1049, 2003.

[10] E. Horowitz, S. Sahni, and S. Anderson-Freed, Fundamentals of Data Structures in C, Computer Science Press, 1993.

[11] 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.

[12] T. Kahveci and A. K. Singh," An Efficient Index Structure for String Databases," In Proceedings of the 27th VLDB Conference, pp. 351-360, 2001.

[13] K. Kelly and P. Labute," The A* Search and Applications to Sequence Alignment," http://www.chemcomp.com/article/astar.htm, 1996.

[14] S. Kurtz, Reducing the Space Requirement of Suffix Trees. Softw. Pract. Exp., Vol 29, pp. 1149-1171, 1999.

[15] S. Kurtz, J. Choudhuri, E. Ohlebusch, C. Schleiermacher, J. Stoye, and R. Giegerich, " REPuter: the manifold applications of repeat analysis on a genome scale," Nucleic Acids Research, Vol. 29, No. 22, pp. 4633-4642, 2001.

[16] U. Manber and G. Myers," Suffix Arrays: A New Method for On-Line String Searches," SIAM J. Comput., Vol. 22, No. 5, pp. 935-948, 1993.

[17] C. Meek, J. M. Patel, and S. Kasetty, " OASIS: An Online and Accurate Technique for Local-Alignment Searches on Biological sequences,"In Proceedings of the 29th VLDB Conference, pp. 920-921, 2003.

[18] G. Navarro and R. Baeza-Yates," A Hybrid Indexing Method for Approximate String Matching," J. of Discrete ALgorithms, Vol. 1, No. 1, pp.205-239, 2000.

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

[20] K. Sadakane and T. Shibuya," Indexing huge genome sequences for solving various problems," In Proceedings of the 12th

Genome Informatics, pp. 175-183, 2001.
[21] H. Shang and T. H. Merrett,“ Tries for approximate string matching,” IEEE Trans. on Knowlege and Data Engineering, Vol. 8, No. 4, pp. 540-547, 1996.
[22] T. Smith and M. Waterman, “ Identification of Common Molecular Subsequences,” Journal of Molecular Biology, 147, pp. 195-197, 1981.
[23] G. A. Stephen, String Searching Algorithms, World Scientific Publishing, 1994.
[24] Z. Tan, X. Cao, B. Ooi, and A. Tung, “ The ed-tree: An Index for Large DNA Sequence Databases,”In Proceedings of SSDBM Conference, pp. 1-10, 2003.
[25] E. Ukkonen, “ Approximate string matching over suffix trees,”In Proceedings of Combinatorial Pattern Matching (CPM93), pp. 228-242, 1993.
[26] H. Wang et al.,“ BLAST++: A Tool for BLASTing Queries in Batches,”In Proceedings First Asia-Pacific Bioinformatics Conference, pp. 71-79, 2003.
[27] H. E. Williams and J. Zobel,“ Indexing and Retrieval for Genomic Databases,” IEEE TKDE Vol. 14, No. 1. pp. 63-78, 2002.

홍 상 균
2005. 2 : 한림대학교 정보통신공학부(학사)
2005. 3 ~ 현재 : 한림대학교 정보통신공학부 석사과정
관심분야 : 데이터베이스 시스템, XML, 저장 시스템
e-mail: kyoons@hallym.ac.kr

원 정 임
1992. 2 : 한림대학교 전자계산학과(학사)
1997. 8 : 한림대학교 컴퓨터공학과(석사)
2003. 2 : 한림대학교 컴퓨터공학과(박사)
2000. 3 ~ 2004. 2 : 한림대학교 교양교육부 강의전담
2004. 3 ~ 현재 : 연세대학교 컴퓨터과학과 연구교수
관심분야 : 데이터베이스 시스템, 데이터 마이닝, XML 응용, 바이오 정보공학, 데이터베이스 보안
e-mail: jiwon@cs.yonsei.ac.kr

윤 지 희
1982. 2 : 한양대학교 전자공학과 졸업(학사)
1985. 3 : 일본 구주대학교 정보공학과 졸업(석사)
1988 3 : 일본 구주대학교 정보공학과 졸업(박사)
1998. 3 ~ 1999. 2 : 미국 UCLA대학교 전산학

과 방문교수
1988. 4 ~ 현재 : 한림대학교 정보통신공학부 교수
관심분야 : 시계열 데이터베이스, 데이터 마이닝, XML, 공간 데이터베이스/GIS
e-mail : jhyoon@hallym.ac.kr

박 상 현
1989. 2 : 서울대학교 컴퓨터공학과(학사)
1991. 2 : 서울대학교 컴퓨터공학과(석사)
2001. 2 : UCLA대학교 전산학과(박사)
1991. 3 ~ 1996. 8 : 대우통신 연구원
2001. 2 ~ 2002. 6 : IBM T. J. Watson Research Center Post-Doctoral Fellow
2002. 8 ~ 2003. 8 : 포항공과대학교 컴퓨터공학과 조교수
2003. 9 ~ 현재 : 연세대학교 컴퓨터과학과 조교수
관심분야 : 데이터베이스, 데이터 마이닝, 바이오인포매틱스, XML
e-mail : sanghyun@cs.yonsei.ac.kr

김 상 욱
1989. 2 : 서울대학교 컴퓨터공학과 졸업(학사)
1991. 2 : 한국과학기술원 전산학과 졸업(석사)
1994. 2 : 한국과학기술원 전산학과 졸업(박사)
1991. 7 ~ 8 : 미국Stanford University, Computer Science Department 방문 연구원
1994. 2 ~ 1995. 2 : KAIST 정보전자연구소 전문 연구원
1999. 8 ~ 2000. 8 : 미국 IBM T.J. Watson Research Center Post-Doc.
1995. 3 ~ 2000. 8 : 강원대학교 컴퓨터정보통신공학부 부교수
2003. 3 ~ 현재 : 한양대학교 정보통신대학 정보통신학부 부교수
관심분야 : 데이터베이스 시스템, 저장 시스템, 데이터 마이닝, 바이오 정보공학, 멀티미디어 정보 검색, 공간 데이터베이스/GIS, 주기억장치 데이터베이스, 트랜잭션 관리
e-mail : wook@hanyang.ac.kr