유전체 데이터베이스를 위한 효율적인 접미어 트라이 인덱스 구조

박진만*, 원정임*, 윤지희*, 박상현**
*한림대학교 컴퓨터공학과, **포항공과대학교 컴퓨터공학과
{pjm,jiwon,jhyoon}@hallym.ac.kr, sanghyun@postech.ac.kr

An Efficient Suffix Trie Index Structure for Genomic Databases

Jin-Man Park*, Jung-Im Won*, Jee-Hee Yoon*, Sang-Hyun Park**
*Dept of Computer Engineering, Hallym University
**Dept. of Computer Science and Engineering, Pohang University of Science and Technology

요 약

DNA 시퀀스는 A, C, G, T 네 개의 문자로 구성된 매우 긴 시퀀스로 볼 수 있다. 고속으로 유사 DNA 시퀀스를 검색하기 위하여 인덱싱 기술을 이용하는 것이 일반적이다. 그러나 검색 대상의 유전체 데이터베이스는 그 크기가 매우 크며, 또한 지수 함수적으로 크기가 급속히 증가하고 있으므로, 기존의 인덱싱 기법을 그대로 적용할 경우, 실용성에 한계가 있다. 본 논문에서는 이와 같은 문제점을 해결할 수 있는 대규모 유전체 데이터베이스를 위한 효율적인 인덱싱 기법과 질의처리 기법을 제안한다. 기본 구조로서 접미어 트라이를 사용하며, 접미어 트리 인덱스 구조의 최대 단점인 인덱스 크기를 줄일 수 있는 데이터 압축 표현 방식을 제안한다. 또한 제안된 데이터 압축 표현 방식의 디스크 기반 인덱스 구성 알고리즘과 이를 활용한 부분 시퀀스 검색 알고리즘을 보이고, 그 저장 성능의 비교 평가 결과를 보인다.

1. 서 론

유전체 데이터베이스를 대상으로 하는 유사 검색 기술은 인간 유전체 프로젝트와 더불어 필수적으로 발전하여 온 분야이다. 유전체 시퀀스의 기능을 이해하는 가장 기본적인 방식으로 임의의 두 시퀀스 사이의 유사도를 측정하는 것을 들 수 있으며, 유사 시퀀스 검색과 해석 기법에 의한 수많은 괄목할만한 연구 결과가 보고되어 있다. 예를 들어 기능이나 구조 등이 판명된 DNA 시퀀스 데이터베이스로부터 임의의 새로운 DNA 시퀀스를 질의 시퀀스로 하여 유사한 시퀀스를 검색하는 문제는 새로운 DNA 시퀀스의 역할, 진화과정, 화학적 구조 등을 추론하는 중요한 방식으로 사용될 수 있다.

DNA 시퀀스 검색을 위하여 일반적으로 널리 사용되는 방식은 휴리스틱스에 기반한 필터링 기법[1][2][3]을 이용한 순차검색(sequential search)으로 볼 수 있다. 이 들 방식에 기반한 실제 응용 시스템은 일반적으로 병렬 컴퓨터를 사용한 대규모 인프라를 필요로 하며, 효율적 검색을 위해서는 사용자의 적합한 파라메터 선정 등 전문적인 지식에 의한 탐색 공간 축소 작업이 절대적으로 필요하다.

유전체 데이터베이스는 그 크기가 매우 크다. 초기의 DNA 시퀀스 데이터베이스는 느린 속도로 성장하여, 1992년에 GenBank[4]는 1억이 조금 넘는 염기에 해당하는 78,000개의 DNA 시퀀스을 갖고 있었다. 그러나, 1995년 서열 결정 기술의 진보와 함께 인간 유전체 프로젝트는 GenBank를 더 높은 단계로 성장시켰으며, 근래의 GenBank는 이미 방대한 양에 이르러, 매 6-8개월마다 그 규모가 두 배로 늘어나고 있고, 그 증가율 또한 지속적으로 증가하고 있다. 이와 같은 유전체 데이터베이스의 급속한 증가 추세와 지속적인 활용범위의 확대에 따라, 유전체 데이터베이스의 유사 시퀀스 검색 효율을 증대시키기 위한 연구가 시급히 필요하며, 최근 데이터베이스 분야에서는 인덱싱 기반 기술을 활용한 연구[5][6][7]가 활발히 진행되고 있다.

기존의 인덱싱 기술을 이용하여, 대형 유전체 데이터베이스의 인덱스를 구축하기 위하여는 상당한 인덱스 구축 시간과 저장 공간이 필요하다. 일반적으로 유전체 데이터는 빈번한 업데이트 작업이 없는 안정된 상태로 유지되므로, 인덱스 구성 시간이 크게 문제되지 않는다고 가정할 수 있다. 그러나 금후 실용화를 위한 고성능 인덱싱 기술을 확보하기 위해서는 초대형 인덱스 구성을 위한 디스크 기반의 안정된 인덱스 구성 알고리즘의 개발, 저장 공간의 문제 해결, 확장성, 정확성, 검색 효율의 안정성 등을 보완한 제반 기술의 개발이 요구된다.

본 연구에서는 유전체 데이터베이스를 위한 인덱싱의 기본 구조로서 접미어 트리(suffix tree)[8]를 활용한다. 그 이유는 부분 시퀀스(subsequence) 검색 효율이 타 방식에 비하여 월등히 우수하며, 유사 검색을 위한 알고리즘들이 알려져 있고[9][10], 또한 그 성질에 의하여 이미 소규모 유전자 정보처리에 실제로 활용되고 있기 때문이다[11][12]. 그러나 앞에서 언급한 바와 같이 접미어 트리가 유전체 정보 인덱싱에 활용되어 실용화되기 위해서는 우선 인덱스 크기의 문제가 해결되어야 한다. 본 연구에서는 접미어 트리 구조의 최대 단점인 인덱스 크기를 줄일 수 있는 방안의 하나로서 포인터를 사용하지 않는 트라이 표현(pointerless trie representation)방식[13]을 사용한다.

본 논문에서는 대규모 유전체 데이터베이스를 위한 효율적인 인덱싱 기법과 질의처리 기법을 제안한다. 본 논문의 공헌은 다음과 같다. (1) 대규모 유전체 데이터베이스를 위한 디스크 기반의 안정된 접미어 트리 인덱스 구성 알고리즘을 제안한다. (2) 저장 공간 감소를 위하여 포인터를 사용하지 않는 트라이 표현 방식을 사용한다. (3) 인덱싱 대상이 되는 가변길이 접미어 시퀀스를 효율적으로 인덱싱 할 수 있는 포인터를 사용하지 않는 트라이 표현 방식을 개발한다. (4) DNA 시퀀스 데이터에 출현하는 문자의 종류가 제한적인 점 등 데이터 특성을 고려하여 각 문자를 4비트 이진코드로 표현하여 저장공간의 효율을 높인다. (5) 제안된 인덱싱 방안을 사용한 질의처리 방안을 제시한다. (6) 제안된 인덱싱 기법의 성능을 평가하기 위하여 인덱스 크기를 비교, 분석한다.

2. 관련 연구

유전체 데이터베이스를 위한 유사검색 및 인덱싱 기

법과 직접적으로 연관된 기존의 연구 결과에 관하여 간략히 요약하고, 각 방법이 갖는 문제점을 지적하면 다음과 같다. 우선, 제안된 대부분의 유사 검색 알고리즘은 인-메모리(in-memory) 방식으로서[14][15], 질의 시퀀스 검색을 위하여는 전체 데이터베이스를 순차 검색하여야 한다. 따라서 데이터베이스의 크기가 기하급수적으로 증가하고 있는 유전체 데이터베이스의 경우, 이들 방식은 실용성에 한계가 있다.

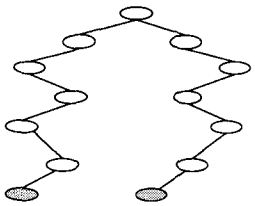
한편, 기존의 텍스트 데이터를 대상으로 하는 인덱스 기법 등 기존의 인덱싱 기법을 유전체 데이터베이스에 적용한 경우[16][17], 인덱스의 전체 크기는 데이터베이스의 크기에 비하여 지나치게 커지며, 검색 효율도 저하한다. 따라서 현시점에 있어 대부분의 유전체 데이터베이스 인덱싱 기술은 대부분 실용성의 한계에 의하여 상용화되어 활용되지 못하고 있는 실정이다[7].

최근, 참고문헌 [6]에서는 DNA 부분 시퀀스를 웨이브렛 변환(wavelet transformation) 하여 얻어진 변환 계수를 이용하여 다차원 정수 공간으로 매핑한 후, MBR(Minimum Bounding Rectangle)을 이용하는 디스크 기반 인덱싱 기법을 제안하고 있다. 한편, 참고문헌 [5]에서는 기존의 텍스트 데이터베이스를 위한 부분 시퀀스 검색에 대하여 가장 좋은 효율을 보이는 것으로 알려진 접미어 트리 인덱스 구조에 착안하여, 이를 대규모 유전체 데이터베이스 인덱싱에 활용하고 있다. 참고문헌 [5]에서는 기존의 접미어 트리 구성 알고리즘을 확장하여 디스크 기반의 인덱스 구성 방식을 제안, 접미어 트리가 실제로 대규모 유전체 데이터베이스 인덱싱 기술로 활용될 수 있음을 보이고 있다. 또한 접미어 트리의 최대 단점인 인덱스 크기의 문제를 보완하기 위한 작업을 소개하고 있으나, 인덱스의 크기는 여전히 데이터베이스의 크기보다 매우 큰 단점을 가지고 있다.

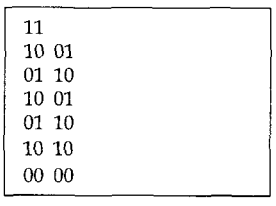
3. 인덱스 구조

3.1 트라이

트라이[18]는 노드(node)가 정보를 가지지 않고, 에지(edge)에 데이터가 저장되는 트리 구조이다. 기본 구조로 트라이의 각 노드에는 여러개의 에지가 연결되며, 에지에는 임의의 정보를 저장할 수 있어, 루트로부터 단말 노드까지의 에지 정보를 접합(concatenation)하여 임의 정보를 저장하는데 사용된다. 이진 데이터 저장 구조로서 트라이를 사용하는 경우, 트라이의 각 노드는 최대 2개의 에지를 가지는 이진 트라이(binary trie)의 형태로 제한되므로 에지 정보가 '0'이면 왼쪽 에지에, '1'이면 오른쪽 에지에 정보를 저장함으로써 에지 정보를 생략 표현할 수 있다. (그림 1)에 이진 시퀀스 데이터 S1='001010'과 S2='110100'를 이와 같은 트라이 구조로 표현한 예를 보인다.



(그림 1) 트라이 구조의 예



(그림 2) 포인터를 사용하지 않는 트라이 표현의 예

이와 같은 이진 트라이 구조는 다음과 같이 포인터를 사용하지 않는 내부 표현이 가능하다. 즉 각 노드는 왼쪽과 오른쪽 에지 중 하나를 가지거나, 모두 가지거나, 전혀 가지지 않는 형태 중 하나이므로, 각 노드 당 2비트를 할당하여 각각의 노드 형태를 표현할 수 있다. 즉 '01'은 노드에 오른쪽 에지만이 연결된 형태를 표현하고, '10'은 노드에 왼쪽 에지만이 연결된 형태를 표현하고, '11'은 노드에 왼쪽 에지와 오른쪽 에지가 모두 연결된 형태를 표현하고, '00'은 에지가 연결되지 않은 단말노드의 형태를 표현한다. (그림 2)에 (그림 1)의 트라이를 포인터를 사용하지 않는 구조로 표현한 예를 보인다.

3.2 접미어 트라이 인덱스

제안하는 인덱스 구축 방안은 다음과 같다. 본 연구에서는 부분 시퀀스 검색을 지원하기 위하여 DNA 시퀀스로부터 가능한 모든 접미어를 추출, 이를 인덱싱의 대상으로 한다. 따라서 인덱스의 기본 구조로서 접미어 트라이를 사용한다. 접미어 트라이는 인덱싱 대상이 되는 접미어 들이 많은 공통 접두어 부분 시퀀스를 가질 때 좋은 압축 효과를 갖는다. DNA 시퀀스로부터 추출된 접미어는 데이터의 특징에 의하여 많은 공통 접두어 부분 시퀀스를 가지므로 트라이 인덱스 구조로 구현되어 좋은 압축 효과를 기대할 수 있다.

DNA 시퀀스는 A, C, G, T 네 개의 문자로 구성된 매우 긴 시퀀스로 볼 수 있다. 그러나 실제의 데이터베이스에는 이 네 개의 문자 외에 11개의 와일드 카드 문자가 출현할 수 있으므로[7], 총 15개의 문자가 DNA 시퀀스에 출현할 수 있다. 또한 각 시퀀스를 유일하게 구별하기 위하여 사용된 특수 문자 '$'가 출현할 수 있다. 따라서 본 인덱스 구조에서는 인덱스의 압축 표현을 위하여 각 문자의 구별을 위한 최소 정보 단위로 4비트를 사용한다. 4비트를 사용한 문자 표현 예를 (그림 3)에 보인다. 또한 인덱스의 압축 표현을 위하여 트라이의 각 노드를 포인터를 사용하지 않는 2비트의 이진 데이터로 표현한다. DNA 시퀀스를 위한 접미어 트라이 인덱스 생성 알고리즘은 다음과 같다.

(1) DNA 시퀀스의 모든 접미어 시퀀스를 추출한다.
(2) 접미어 시퀀스의 각 문자당 4비트를 할당하여 이진화 문자열로 변환한다. 다음 이들을 오름차순으로 정렬한다.
(3) 정렬된 이진 접미어 시퀀스를 순서대로 트라이에 삽입한다. (3-1) 시퀀스가 삽입되는 과정에 의하여 새로이 생성 또는 변경되는 노드 구조를 2비트 노드 정보로 해당 페이지 영역에 기록한다. 이 때 한 페이지에 저장할 최대 트리 높이와 최대 노드 수에 의하여 페이지 크기가 결정된다. (3-2) 새로이 삽입되는 노드에 의하여 해당 페이지 영역 내에 오버 플로우가 발생할 가능성이 있으면, 이 노드 를 제외한 나머지 해당 페이지 영역을 디스크에 기록하고, 기록되지 않은 노드 정보로 해당 페이지 영역을 재구성한다. 디스크에 해당 페이지 영역을 기록할 때, 페이지 정보를 함께 기록한다.
(4) (3)의 과정에 의하여 모든 접미어 시퀀스가 삽입된 후, (2)의 모든 접미어 시퀀스를 재 입력해서 각 접미어 시퀀스의 출현 위치를 나타내는 트라이 내의 단말 노드 정보(단말 노드 번호)를 추출, 저장한다.

구체적 예를 이용하여 인덱스 구성 방식을 간략히 설명하면 다음과 같다. 예를 들어 S1='ACGT'와 S2='ACT'의 두 개의 DNA 시퀀스에 대한 인덱스를 구성하는 경우, 우선 각 시퀀스로부터 모든 접미어 시퀀스를 추출하여 각 문자당 4비트를 할당하여 (그림 4)와 같은 이진 시퀀스로 변환한다. 여기서 '$'는 각 접미어 시퀀스의 마지막 문자로 첨가되어 시퀀스를 유일하게 구분하기 위하여 사용된다. 다음, 이 들 이진 시퀀스들을 오름차순으로 정렬한 후, 차례로 트라이에 삽입하면, (그림 5)와 같은 트라이 구조와 포인터를 사용하지 않는 이진 데이터 내부

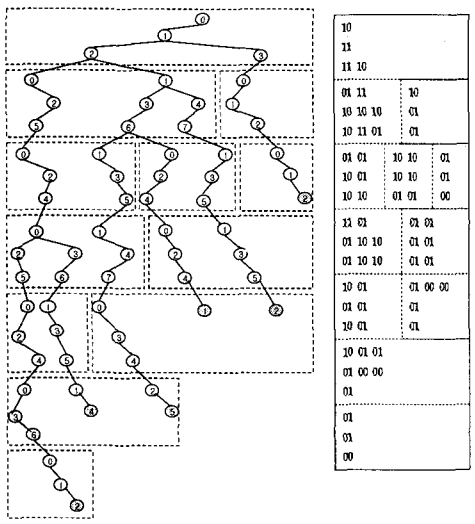
표현을 얻는다.

문 자	이진 표현
A	0001
C	0010
G	0011
T	0100
:	:
:	:
$	1111

(그림 3) 각 출현 문자의 이진 표현 예

추출된 접미어 시퀀스	이진 변환된 접미어 시퀀스
S1: ACGT$	0001001000110100 1111
CGT$	0010001101001111
GT$	001101001111
T$	01001111
S2: ACT$	0001001001001111
CT$	001001001111
T$	01001111

(그림 4) DNA시퀀스로부터 추출된 접미어 시퀀스의 예



(그림 5) 접미어 트라이 구성 및 내부표현 예

이와 같이 생성되는 트라이의 이진 데이터 노드 정보는 데이터 생성과 함께 페이지 단위로 디스크에 저장되며, 페이지 내의 노드 수가 증가함에 따라 페이지 분할이 발생한다. 따라서 각 페이지에 저장되는 노드를 식별하기 위하여 별도의 페이지 정보를 기록하여, 저장한다. (그림 5)의 점선 영역 구분은 디스크 페이지 할당을 나타내며, 여기에서는 페이지 크기를 16비트로 가정하여, 한 페이지에 저장할 수 있는 최대 트리의 높이를 3으로, 페이지 내에 저장될 수 있는 최대 노드 수를 8로 가정한 예를 나타낸다. <표 1>은 이 때 생성되는 페이지 정보를 나타낸다. 한편, (그림 5)에서 검게 표현된 1, 2, 4, 5번 노드는 각각 트라이에 삽입된 각 접미어 시퀀스의 삽입 위치를 나타내는 단말 노드를 나타낸다. 그러나 이진 데이터 표현의 트라이 구조는 별도의 노드 정보를 갖지 못하므로 단말 노드 정보는 별도로 저장, 관리되어야 한다. <표 2>에 예제에서 사용되는 단말 노드 정보의 예를 나타낸다.

<표 1> 페이지 정보

P	T	B	N	A
0 0	0	0	4	132
0 1	1	3	-1	-1
1 0	0	0	8	78
1 1	2	4	3	150
1 2	3	5	-1	-1
2 0	0	0	6	24
2 1	2	2	8	108
2 2	4	4	3	165
2 3	5	4	-1	-1
3 0	0	0	8	48
3 1	2	3	6	180
3 2	4	5	-1	-1
4 0	0	0	6	0
4 1	2	2	5	204
4 2	5	3	-1	-1
5 0	0	0	7	225
5 1	3	1	-1	-1
6 0	0	0	3	252
6 1	1	0	-1	-1

P: 페이지번호, T: 페이지로 들어오는 에지의 수,
B: 페이지에서 나가는 에지의 수, N: 페이지의 노드 수
A: 페이지가 디스크에 출력될 때의 시작 위치

<표 2> 페이지별 단말노드 정보

#Page	#Node	SeqName	Offset	Len
6 0	2	S1	0	4
5 0	5	S1	1	3
4 1	2	S1	2	2
2 2	2	S1	3	1
5 0	4	S2	0	3
4 1	1	S2	1	2
2 2	2	S2	2	1

4. 부분 시퀀스 검색 알고리즘

생성된 접미어 트라이 인덱스를 사용하여 주어진 질의와 일치하는 (부분)시퀀스를 검색하기 위하여는 현재의 트라이 레벨을 구성하는 각 노드들의 비트정보를 기반으로 다음의 트리 레벨의 구조를 파악하는 연산과정이 필요하다. 페이지를 구성하는 각 레벨의 구조를 파악하기 위해 다음과 같은 4개의 카운터를 이용한다. 다음 레벨의 노드 수를 파악하기 위하여 size 변수를 사용하며, 각 레벨의 마지막 노드 위치를 파악하기 위하여 last 변수를, 다음 레벨의 어느 노드로 이동할 지를 결정하기 위한 srch와 next 변수를 사용한다. 접미어 트라이 인덱스를 이용한 부분 시퀀스 검색 알고리즘을 (그림6)에 보인다.

5. 성능 평가

제안된 인덱스 구조의 성능을 평가하기 위하여 실험을 통하여 인덱스의 크기를 비교 분석한다. 비교 대상

```
Page Change
Input : Page Info P, last, next
Output : Null

    find_page(P, last, next);
    next = B+srch-T'-1;
    size = T"-T';
    last = size-1;
    return Null;
```

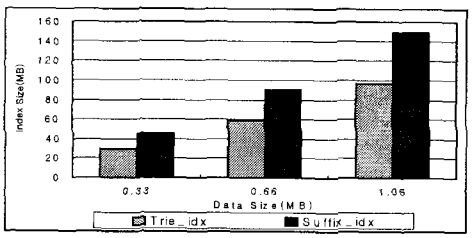
```
Search_Trie
Input : Trie index T, Page Info P, Query Q, Terminal node
Info N
Output : ResultSet R

 next=0; size=1, last=0;
 R ← Search(next, size, last);
 return R

Search
Input : Trie index T, Page Info P, Query Q, Terminal node
Info N, next, size, last
Output : ResultSet R

 pageLevel=0, position=0;
 data_read(P);
 while(pageLevel < page_height) {
    srch = 0;
    for(from current level's start position before next){
        read trie_node
        if(trie_node == 11) srch=srch+2; size++;
        else if(trie_node != 00) srch++;
        else if(trie_node == 00) size--;
        position++;
        }
 read next position's trie_node, read query bit
 if(trie_node == 11 && query bit == 1) srch=srch+2;
 else if( (trie_node==10 && query bit==0)
     || (trie_node==01 && query bit==1) ) srch++;
 else if( (trie_node==10 && query bit==1)
          || (trie_node==01 && query bit==0) )
     No match; break;
 if(paeLevel < page_height-1){
    next=last+srch;
    if(query bit == last query bit)
          R ← find_answer(next, N); break; }
 else {
    next = current page B + last + srch;
    Page_change(current page last, next);
      if(query bit == last query bit){
          data_read(P);
          R ← find_answer(next, N); break; }
      else
              Search(T, P, Q, N, next, size, last);
    }
 for(from current node position until last){
     read trie_node
     if(trie_node == 11) size++;
     else if(trie_node == 00) size--;
     position++;
   }
 last = last + srch;
 pageLevel++;
}
```

(그림 6) 부분시퀀스 검색 알고리즘



(그림 7) 인덱스 크기 비교

의 인덱스 구조로는 디스크 기반의 접미어 트리를 사용한다. 실험 데이터로는 GenBank[19]로부터 다운 받은 HumanChromosome 18 DNA 시퀀스를 사용한다. 실험을 위한 하드웨어 플랫폼으로는 Windows 2000 Server를 운영체제로 사용하고, 1GB의 주기억장치, 40GB 디스크를 갖는 Pentium IV 2GHz의 PC를 사용한다.

실험을 통한 인덱스 크기 비교 결과를 (그림 7)에 보인다. 제안한 인덱스 구조는 접미어 트리 인덱스에 비하여 약 35%의 저장 공간 감소를 나타냄을 알 수 있다. 본 연구에서 제안한 인덱스 구조는 기존의 접미어 트리 인덱스 구성 방식에 비하여 공간적으로 효율적일 뿐 아니라 주기억장치 크기에 거의 영향을 받지 않는 안정된 디스크 기반 인덱스 구성 방식이다.

6. 결론

본 논문에서는 대규모의 유전체 시퀀스 데이터베이스를 대상으로 하는 포인터를 사용하지 않는 접미어 트라이 인덱스 구조를 제안하였다. 저장 효율, 검색 속도, 확장성, 정확성을 고려한 디스크 기반의 인덱스 구조와 부분 시퀀스 검색 알고리즘을 제안하고, 인덱스 크기 비교 실험에 의하여 제안된 인덱스 구조의 성능을 평가하였다. 현재, 다양한 방식을 도입한 인덱스 압축 방안에 관한 연구 및 실험을 지속적으로 수행하고 있으며, 향후 제안된 인덱스 구조를 보다 효율적으로 활용할 수 있는 유사 부분 시퀀스 검색 알고리즘을 개발하고 그에 따르는 성능 평가를 수행할 예정이다.

참고 문헌

[1] S. F. Altschul et al, "Basic local alignment search tool," J. Mol. Biol., 215, pp. 403-410, 1990.

[2] S. F. Altschul, T. L. Madden, A. A. Schaeer, J. Zhang, Z. Zhang, W. Miller, D. J. Lipman. "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," Nucleic Acids Research, 25, pp. 3389-3402, 1997.

[3] W. R. Pearson and D. J. Lipman, "Improved tools for biological sequence comparison," In Proc. Natl Acad Sci USA, 85, pp 2444-2448, 1988.

[4] D. A. Benson, M. S. Boguski, D. J. Lipman, J. Ostell, and B. F. Quellette, "Genban," Nucleic Acids Research, 26(1), pp. 1-7, 1998.

[5] E. Hunt, R. W. Irving, and M. P. Atkinson, "Persistent Suffix Trees and Suffix Binary Search Trees as DNA Sequence Indexes," Technical report, Univ. of Glasgow, Dept. of Computing Science. TR-2000-63, 2000.

[6] T. Kahveci, A. K. Singh, "An Efficient Index Structure for String Databases," In Proc. Int'l. Conference on Very Large Data Bases, VLDB'01, pp. 351-360, 2001.

[7] H. E. Williams and J. Zobel, "Indexing and Retrieval for Genomic Databases," TKDE, 2002.

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

[9] G. Navarro, R. Baeza-Yates, "A new indexing method for approximate string matching," In Proc. CPM99, Lecture Notes in Computer Science, 1645. Springer, pp.163-185, 1999.

[10] E. Ukkonen, "Approximate string matching over suffix trees," In Proc. CPM93, Lecture Notes in Computer Science, 684. Springer, pp. 228-242,1999.

[11] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg, "Alignment of Whole Genomes," Nucleic Acids Res, 27, pp.2369-2376, 1999.

[12] S. Kurtz, C. Schleiermacher, "REPuter: fast computation of maximal repeats in complete genomes," Bioinformatics, 15(5), pp.426-427, 1999.

[13] H. Shang, T. H. Merrett, "Tries for Approximate String Matching,"IEEE TKDE,8(4),pp.540-547, 1996.

[14] R. A. Baeza-Yates and G. Navarro, "Fast Approximate String Matching," Algorithmica, 23(2), pp.127-158, 1999.

[15] S. Uliei, A. Fliess, A. Amir, and R. Unger, "A Simple Algorithm for Detecting Circular Permutations in Proteins," Bioinformatics, 15(11), pp. 930-936, 1999.

[16] R. A. Baeza-Yates and G. Navarro, "A Practical Index for text retrieval allowing errors," CLEI, 1, pp. 273-282, 1997.

[17] U. Manber and E. Myers, " Suffix Arrays : A New Method for on-line string searches," SIAM J. on Computing, 22(5), pp. 935-948, 1993.

[18] E. H. Fredrick, "Trie Memory," CACM, 3(9), pp. 490-499, 1960.

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