ISSN 1598-9798

데이타베이스연구

제28권 제2호 2012년 8월

차세대 염기서열 결정기에서 생성된 리드의 효과적인 정렬을 위한 소수 기반의 해시 알고리즘과 클러스터링 방법

Prime Number based Hash Algorithm and Clustering Approach for Effective Alignment of Reads from Next Generation Sequencing

경규호, 박치현, 여윤구, 박상현
Kyuho Kyung, Chihyun Park, Yunku Yeu, Sanghyun Park

데이터베이스 소사이어티
Database Society

사단법인 한국정보과학회
The Korean Institute of Information Scientists and Engineers

차세대 염기서열 결정기에서 생성된 리드의 효과적인 정렬을 위한 소수 기반의 해시 알고리즘과 클러스터링 방법

Prime Number based Hash Algorithm and Clustering Approach for Effective Alignment of Reads from Next Generation Sequencing

경규호(Kyuho Kyung)[1], 박치현(Chihyun Park)[2], 여윤구(Yunku Yeu)[3], 박상현(Sanghyun Park)[4]

요 약

유전체 염기서열 정렬은 유전체 연구에서 가장 기본적이고 핵심적인 문제로 약 30억 개의 염기서열 문자로 구성된 레퍼런스 지놈 시퀀스에 염기서열 조각을 비교 정렬하여 맵핑되는 위치를 탐색하는 방법이다. 특히 최근 차세대 염기서열 분석(Next Generation Sequencing) 기술이 발전하면서 생성된 대량의 짧은 리드(read)를 빠르고 정확하게 레퍼런스 지놈 시퀀스에 정렬할 수 있는 방법에 대한 연구가 수행되고 있다. 짧은 리드 정렬 알고리즘은 대량의 데이터를 빠르고 정확하게 맵핑해야 하기 때문에 속도와 정확도에 주요한 의미를 두고 있지만 두 요소 사이의 트레이드오프(trade-off) 관계 때문에 두 가지 모두를 만족하는 알고리즘을 만들기란 매우 어렵다. 본 연구에서는 소수를 이용하여 A, C, G, T, N으로 이루어진 염기서열을 효과적으로 표현할 수 있는 새로운 해시 방법을 제안하고 미스매치(mis-match)를 고려할 수 있는 클러스터링(clustering) 방법과 비트(bit) 변환을 적용하여 정확하고 빠르게 염기서열을 정렬할 수 있는 알고리즘을 제시한다. 제안하는 방법의 우수성을 검증하기 위해 NCBI의 실제 인간 염색체를 레퍼런스 시퀀스로 사용하였고, 동일 시퀀스를 이용하여 만든 시뮬레이티드 데이터를 이용하여 BWA와 제안하는 방법에 대한 비교 실험을 수행하였다. 결과적으로 제안하는 방법이 비교 알고리즘과 비교하여 더 높은 정확도와 더 낮은 오류율이 확인되었다.

주제어: 차세대 염기서열 분석, 염색체 정렬, 해시 알고리즘, 비트 변환 및 연산, 소수, 쌍둥이 소수, 사촌 소수, 빅 데이터, 패턴

---

1 연세대학교 컴퓨터공학과, 석사과정
2 연세대학교 컴퓨터과학과, 박사과정
3 연세대학교 컴퓨터과학과, 박사과정
4 연세대학교 컴퓨터과학과, 교수, 교신저자

† 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 도약연구지원사업 지원을 받아 수행된 것임(2012-010775)
+ 논문접수: 2012년 05월 10일, 심사완료: 2012년 07월 24일

Abstract

Sequence alignment which maps DNA sequence fragment into reference genome sequence composed of 3 billion nucleotides is basic and fundamental problem in genomic. With the advent of next generation sequencing(NGS) machine and developing the technology, the researches which can fast and accurately align a large amount of short reads into reference genome have been studied. Because an alignment method has to map short reads into reference fast and accurately, both of speed and accuracy are major factor. However, they are in the trade-off relation and it is difficult to make an algorithm which satisfies both two factors. In this paper, we propose the novel hash method which can present the genomic fragment composed of A, C, G, T and N with prime number. We also propose the clustering approach which can consider the mis-match and bit transformation approach for enhancing alignment speed. **T**o verify the superiority of our method, we used the real genome sequence published by NCBI as a reference data and obtained the simulated data from that. We compared the performance with BWA algorithm using the simulated data. The results showed that our method had higher accuracy and lower error rate than the ones of comparative method.

Keywords: Next Generation Sequencing, Genome Alignment, Hash Algorithm, Bit Transformation, Twin Prime, Cousin Prime, Big Data, Pattern

1. 서 론

전통적인 DNA 염기서열 분석은 염기서열을 분석하는 DNA 부위를 중합효소연쇄반응(Polymerase Chain Reaction 이하 PCR) 기법으로 연속하여 증폭하고 증폭과정에서 ddNTP(dideoxy nucleotide tri-phosphate)가 삽입되어 중단된 조각의 크기를 전기영동 방법으로 염기서열의 크기에 따라 분리하여 형광으로 검색하는 Sanger's chain termination sequencing 방법을 사용하였으나, 차세대 염기서열 분석 방법 중 하나인 Roche 454 시스템에서는 유제(emulsion) PCR로 기름방울 속에서 분석하는 DNA를 증폭하고 증폭된 DNA 조각을 아주 작은 홈에 하나씩 넣어서 그 속에서 염기서열분석(pyro-sequencing)을 연속적으로 수행하여 DNA 염기서열을 식별한다. 또한 Applied Biosystems에서 개발된 SOLiD 시스템은 유제 PCR 과정을 통해 분석하려는 DNA 조각을 증폭하고 8-base로 만들어진 DNA 검출법(probe)을 사용하여 연속적으로 결찰(ligation) 반응을 통해 특정 염기서열의 조합을 읽어 염기서열의 순서를 확인하는 방법을 사용한다. 이러한 방법은 차세대 염기서열 분석 기술로 대량의 염기서열을 분석하는 방식이다. 차세대 염기서열 분석(Next Generation Sequencing, 이하 NGS)이라는 용어는 2007년 Solexa가 Illumina사에 합병되면서 사용하기 시작한 용어로 NGS 기술의 발전으로 전통적인 생거 기법에 의한 방법에 비해 훨씬 쉽고 저렴한 비용으로 분석할 수 있게 되었다[1].

최근 이러한 NGS 기술이 발달하면서 대용량의 짧은 유전체 조각인 리드(read)들을 빠르고 정확하게 레퍼런스 상의 올바른 위치로 맵핑하는 방법이 필요로 하게 되었다. 염기서열의 정렬은 유전체의 변이를 찾거나 전체 지놈 시퀀스를 조립하는 과정에서 반드시 필요한 핵심 기술이다. 정확하지 않은 리드 정렬은 정확한 지놈 시퀀스를 얻을 수 없을 뿐만 아니라 유전 변이도 정확하게 찾지 못하게 된다. 또한 정확한 서열조립과 유전변이 탐색을 위해서는 30X 이상의 높은 커버리지(coverage)로 시퀀싱을 해야 하는데 이에 따라서 정렬해야 하는 리드의 양이 굉장히 많아질 수 있다. 따라서 정렬 알고리즘의 정확도와 함께 실행속도는 성능을 나타내는 주요한 요소가 된다.

NGS 기술이 개발되기 전 대표적인 서열 정렬 알고리즘으로는 BLAST(Basic Local Alignment Search Tool)[2, 3, 4, 5, 7], FASTA[5, 6] 그리고 BLAT(The BLAST-Like Alignment Tool)[5, 8] 등이 있다. 이런 방법들은 대부분 대용량 데이터 환경에 맞게 디자인 되지 않았기 때문에 현재 데이터 환경에 적합하지 않다. 따라서 최근 몇 년 동안 NGS 환경에 적합한 새로운 서열 정렬 알고리즘들이 개발되었다.

대표적인 예인 MAQ(Mapping and Assembly with Quality)[7, 9]는 짧은 리드를 해싱 기법을 사용하여 인덱스 테이블로 생성하는 알고리즘으로 짧은 삽입-결실(Indel)과 잠재적인 SNP(Single Nucleotide Polymorphism), 변이를 빠르게 정렬할 수 있다. MAQ는 레퍼런스를 차례로 탐색하여 여러 위치에서 동일한 정렬이 탐색되면 리드 전체 길이의 품질 합을 베이지안 통계 모델로 계산하여 가장 좋을 것을 선택하고, 품질의 합이 동일한 경우에는 변이가 있는 것으로 간주하고 무작위로 하나의 위치를 선택하여 정렬 한다. 28 bp에서 2개의 미스매치(mis-match)를 탐색하기 위해 인접하지 않는 시드(seed) 템플릿을 6개의 해시 테이블로 하고 탐색은 시드 영역에서 미스매치 2까지(0, 1, 2) 허용하여 맵핑 품질과 위치를 메모리에 저장한다. MAQ는

MAQview와 함께 Illumina 시퀀싱의 맵핑 데이터에 효과적인 특징이 있다. 하지만 MAQ는 최근 방법들에 비하여 수행시간이 느리다는 단점이 있다.

SeqMap[10]은 해시 테이블을 이용하여 짧은 시퀀스를 레퍼런스의 모든 위치에서 탐색하는 인덱스 필터링 알고리즘으로 클러스터를 병렬 처리하여 맵핑한다. 삽입-결실(Indel)과 대체된 base를 탐색하기 위해 리드를 4부분으로 분리하여 2부분에서 완벽하게 일치하는 것을 탐색한다. 하지만 리드를 4부분으로 나누어서 미스매치를 탐색하기 때문에 모든 가능한 미스매치를 고려할 수 없고 Genome Analyzer의 리드만 이용 가능하다는 단점이 있다.

SOAP(Short Oligonucleotide Analysis Package)[11]은 레퍼런스의 시드 해시 탐색 테이블과 정렬을 위해 lookup 테이블 알고리즘을 사용하며 리드와 레퍼런스를 2 bit 숫자 데이터로 변환하고 exclusive-OR로 비교한다. 레퍼런스에서 리드를 탐색할 때 기본적으로 최대 2개의 미스매치와 하나의 연속된 갭을 허용하여 정렬한다.

MOM(Maximum Oligonucleotide Mapping)[12]은 쿼리 매칭의 개념을 적용한 방법으로 시드 기반의 해시 테이블을 이용하는 알고리즘이다. 서열 정렬을 위해 레퍼런스와 쿼리 시퀀스에서 고정길이 k-mers의 해시 테이블을 생성하고 정확하게 매칭되는 부분을 시드로 하여 명시된 미스매치 수 안에서 최대 길이의 가능한 맵핑을 찾는다. 하지만 갭(Gap)과 미스매치를 허용하지 못하는 단점이 있다. 그러나 SOAP[11]보다는 긴 길이의 정렬을 할 수 있다.

가장 널리 사용되고 있는 서열 정렬 알고리즘인 BWA(Burrows-Wheeler Alignment, 이하 BWA)[13, 14]는 BWT(Burrows-Wheeler Transform)을 적용하여 접미사 배열 값(Suffix Array value)을 구함으로서 인덱스를 구축한다. 구축된 인덱스를 기반으로 리드의 위치를 탐색하는 과정에서 상대적으로 작은 메모리를 사용하여 prefix trie의 부분 시퀀스 탐색을 효과적으로 모방할 수 있다. BWA는 실행시간 중 2GB 이내의 메모리만 사용하면서 prefix trie를 탐색하는 것과 같은 효과를 나타낼 수 있다는 특징이 있고 이 때문에 빠른 수행시간 안에 비교적 높은 정확도를 갖는 서열정렬 결과를 도출할 수 있다는 장점이 있다. 하지만 BWA는 사용자가 부여하는 인자와 사용하는 데이터의 변이 등에 따라서 정확도가 감소하고 속도는 증가하는 단점이 있다.

제안하는 방법에서는 가장 대표적인 방법인 BWA보다 유전체 데이터의 변이에 강하여 높은 정확도와 낮은 오류율을 갖는 알고리즘을 개발하였다. 이를 위해 본 연구에서는 기존 방법들과 차별되는 레퍼런스 서열에 대한 해시 알고리즘과 미스매치를 고려할 수 있는 클러스터링 방법을 제안한다. 2장에서는 새로운 해시 알고리즘을 위한 소수 이론을 제시하고 3장에서는 제안하는 해시 알고리즘과 클러스터링 및 비트 변환 방식을 설명한다. 4장에서는 제안하는 본 방법과 비교 알고리즘에 대한 실험 결과를 나타내며 마지막으로 5장에서 결론을 서술한다.

2. 해시 알고리즘과 소수

해시 함수(hash function)[15] 또는 해시 알고리즘은 임의의 데이터로부터 원래 데이터를 상징하는 짧은 값이나 키로 변환하여 해시 값을 생성하며 원래의 데이터가 바뀌면 해시 값도 달라진다는 결정적인 성질을 갖고 있다. 또한 해시 함수는 가능한 해시 충돌(collision)이 발생되지 않아야 하지만, 거의 모든 해시 함수는 잠재적인 충돌 가능성을 갖고 있다.

이러한 해시 충돌은 많이 날수록 서로 다른 데이터를 구별하기 어려워지고 데이터를 검색하는 비용이 늘어나기 때문에 해시 알고리즘에서는 충돌을 회피하기 위해 버킷(bucket), 해시테이블(hash table), 적재 밀도 등을 고려한 다양한 방법을 이용하고 있다. 일반적으로 동일한 해시 값 생성으로 발생하는 해시 충돌 최소화와 유일한(unique) 해시 값 생성을 위해 소수(prime)를 사용하기도 한다. 소수는 자연수의 세계에서 원자와도 같은데 그 이유는 모든 소수는 그 보다 작은 두 수의 곱으로 쓸 수 없기 때문이다. 소수의 중요성은 다른 모든 수들이 이들의 곱으로 만들어 진다는 것이며, 이런 소수를 이용하여 질서와 패턴을 찾고자 하는 분야에서 연구되고 있다[16].

쌍둥이 소수(Twin Prime)[17]는 소수 가운데에서 차이가 2가 나는 한 쌍의 소수이며 3-5, 5-7, 11-13, 17-19 등이 그러한 예이다. n보다 작은 쌍둥이 소수의 비율은 n이 증가함에 따라 감소하므로 시퀀스의 위치 값을 대체하여 생성되는 해시 값이 리드의 크기가 커질수록 더욱 유일하게 될 것이다. 하디-리틀우드(Hardy-Littlewood)는 n보다 작은 쌍둥이 소수의 개수는 대략 (1)과 같다고 추측했다.

$$\frac{2Cn}{(\log n)^2} \cdots (1)$$

여기서 C≒0.6601 로 쌍둥이 소수 상수라고 한다. [18]에 따르면 p+2가 소수가 되거나 아니면 두 소수의 곱이 되는 소수 p는 무수히 많다고 증명했다. n보다 작은 값 가운데 p가 소수이고 p+2는 소수이거나 준 소수가 되는 그러한 쌍의 개수는 1.05 x (1) 이상이다. 지금까지 발견한 쌍둥이 소수 가운데 최대는 $33218925 \times 2^{169690} \pm 1$로 51,090 자리 수이다.

사촌 소수(Cousin Prime)[17]는 소수 가운데에서 차이가 4가 나는 한 쌍의 소수이며, 둘 사이의 관계는 쌍둥이 소수보다는 멀다. 200 이내의 쌍둥이 소수와 사촌소수의 쌍은 14쌍이며, 사촌 소수는 3-7, 7-11, 13-17, 19-23 등이 그러한 예이다.

만약 하디-리틀우드 추측 가운데 첫 번째 추측이 옳다면 무한대로 갈 때 쌍둥이 소수와 사촌 소수는 유사한 밀도를 갖는다. [표 1]에서 보는 바와 같이 쌍둥이 소수와 사촌 소수는 각 구간 10^7까지의 평균 간격이 거의 유사하다.

본 연구에서는 다양한 소수 종류 중 대표적인 소수(Prime)와 쌍둥이 소수(두 수의 차가 2인 소수의 쌍(P, P+2)인 소수), 사촌 소수(두 수의 차가 4인 소수의 쌍(P, P+4)인 소수)에서 P 소수를 사용하여 레퍼런스 지놈 시퀀스를 슬라이딩 윈도우 하면서 한 윈도우(Window)의 각 base 위치 값을 P 값으로 대체하여 유일한 해시 값을 키로 생성한다.

[표 1] 각 소수의 10^7까지의 구간별 개수와 소수간의 평균 간격

구간	Prime	평균 간격	Twin Prime	평균 간격	Cousin Prime	평균 간격
10^2	25	4	8	12.50	8	12.50
10^3	168	5.95	35	28.57	40	25.00
10^4	1,229	8.14	205	48.78	202	49.50
10^5	9,592	10.43	1,224	81.70	1,215	82.30
10^6	78,498	12.74	8,169	122.41	8,143	122.80
10^7	664,579	15.05	58,980	169.55	58,621	170.59

3. 제안하는 알고리즘

제안하는 알고리즘은 전처리 과정 1, 2 단계를 통해 윈도우를 해시 데이터 셋으로 전환한다. 전처리 1단계에서는 레퍼런스 시퀀스에서 base-by-base 단위의 윈도우(32/28/24-bases 길이)를 생성하고 각 윈도우마다 해시 알고리즘을 사용하여 고유한 해시키(hash key) 값을 갖는 리드 테이블을 만든다. 고유한 해시키 값을 생성하기 위해 염기 5문자를 숫자로 전환하고 윈도우(32 bases)에 있는 각 염기문자의 합과 각 base의 위치 값에 대응되는 소수 값을 각 base 대체 숫자 값과 곱하여 각각의 결과 값을 조합하여 해시키 값으로 사용한다. 또한 서로 다른 윈도우에서 동일한 해시키 값이 생성되어 충돌이 일어나는 경우에는 순번 증가로 고유한 해시키 값이 유지되도록 한다. 또한 레퍼런스 시퀀스의 서로 다른 위치에서 생성된 윈도우의 길이가 모두 동일한 염기문자인 경우에는 체이닝(chaining) 방법으로 포지션 테이블을 생성하여 리드 테이블에는 동일한 길이의 염기문자 원도우가 없도록 한다. 전처리 2단계에서는 1단계에서 생성된 리드 테이블과 포지션 테이블을 이용하여 패턴에 따른 클러스터 테이블을 생성한다.

3.1 해시키: 염기문자의 대체 숫자 선택

5개 염기문자{A, C, G, T, N}를 대체할 수 있는 숫자 집합을 선별한다. 대체 숫자는 10미만의 숫자를 사용하여 해시키의 공간 사용을 줄이고자 하였다. 1부터 9 사이의 숫자에서 5개 염기문자를 대체할 숫자의 조합을 선택한다. 조합은 5개의 숫자 출현 개수의 합이 실험에서 사용하는 리드의 길이 32 bases를 넘지 않는 조건으로 시뮬레이션 하였으며, 중복되지 않는 5개의 숫자 집합은 모두 126개가 추출 되었다. 이중에서 충돌(collision)로 인한 중복이 최소가 되고 버킷(bucket) 수가 가장 많은 집합은 {1, 2, 3, 8, 9}과 {1, 2, 7, 8, 9}으로 분석되었고, 마지막으로 이 두 개의 숫자 집합에서 해시키 값의 크기를 고려하여 작은 수가 많은 {1, 2, 3, 8, 9}를 대체 숫자 조합으로 선택한다. 해시키 생성을 위한 염기문자 대체 숫자 조합에 대한 실험은 4장에 제시하였다.

3.2 해시키: 염기문자와 숫자의 매치

시뮬레이션에 의해 선택된 대체 숫자 집합 {1, 2, 3, 8, 9}를 염기문자와 매치하는 방법은 선택된 숫자에 2의 배수와 3의 배수가 존재하므로 각 리드에 의해 생성되는 해시키 값의 충돌을 최소화 하고자 레퍼런스 시퀀스에서 각 염기문자 {A, C, G, T, N}의 출현 빈도(A=24.3%, C=22.7%, G=22.8%, T=24.4%, N= 5.6%)를 계산하고 가장 빈도가 많은 염기문자 T에 가장 큰 숫자를 9를 매치하고 다음으로 빈도가 높은 염기문자 A를 가장 먼 숫자 2로 매치하는 방법으로 각각의 염기문자에 숫자를 매치한다. 그러나 1의 숫자는 곱셈 연산으로 생성되는 해시키 값에 영향을 주지 않기 때문에 가장 출현빈도가 작은 N을 매치한다. 결과적으로 염기문자와 숫자의 매치 값은 A=2, C=3, G=8, T=9, N=1이 된다. 해시키를 생성하기 위한 윈도우의 특성을 분석하는 것은 해시키의 장점인 O(1) 탐색 기능을 유지하면서 다섯 문자의 반복으로 구성된 윈도우의 충돌을 최소화 하는 특성을 찾기 위함이다. 또한 소수, 쌍둥이 소수의 (P, P+2)의 P 값, 사촌 소수의 (P, P+4)의 P값을 염기문자의 각 위치 값으로 대체하여 곱한 수를 더하여 해시키 값으로 한다. 여기서

1~10 사이의 숫자는 각 염기문자를 대체하는 숫자로 사용하였기 때문에 11이상의 각 소수를 사용한다. 만약 [그림 1]과 같이 “AGAGA...AGCC”로 이루어진 32 bases 길이의 리드가 있다고 한다면 윈도우의 각 base 대체 숫자의 합 [그림 1의 ①](156: 2 + 8 + 2+ 8 + … + 3 + 3), 윈도우의 각 base 대체 숫자 곱하기 쌍둥이 소수 의 합 [그림 1의 ②](57588: 2x11 + 8x17 + … + 3x827 + 3x857), 윈도우의 각 base 대체 숫자 곱하기 쌍둥이 소수의 역방향으로 위치하여 곱한 합 [그림 1의 ③](50556: 2x857 + 2x827 + … + 3x17 + 3x11)과 해시 충돌(서로 다른 윈도우에서 상기 3개의 값이 동일하게 생성되는 경우)을 해결하기 위한 순번을 조합하여 해시 값을 생성한다.

이 경우 해시 값은 156, 57588, 50556과 1을 조합한 1565758850556 1이 된다.

1	2	3	4	5	…	29	30	31	32	
A	G	A	G	A	…	A	G	C	C	
										①
2	8	2	8	2	…	2	8	3	3	156
(Twin Prime)										
1	2	3	4	5	…	29	30	31	32	
11	17	29	41	59	…	809	821	827	857	
857	827	821	809	659	…	41	29	17	11	
(2*11)	(8*17)								(3*857)	②
22	136	58	328	118	…	1618	6568	2481	2571	57588
(2*857)	(8*827)								(3*11)	③
1714	6616	1642	6472	1318	…	82	232	51	33	50556

[그림 1] 쌍둥이 소수를 사용한 해시키 값 예

3.3 해싱 데이터 셋: 염기문자의 비트 변환

해싱 데이터 셋은 리드 테이블, 포지션 테이블, 클러스터 테이블로 구성되며 레퍼런스 시퀀스에서 32/28/24 bases 길이의 윈도우를 분리하여 리드 테이블을 생성하고, 해시키 값과 각 베이스를 비트로 변환한다. 5개의 염기문자를 5 bit로 변환하는 것은 컴퓨터의 문자가 1byte(8 bit)이므로 메모리 저장 공간을 줄이고, 각 염기문자의 고유한 bit 위치를 사용하여 리드 크기 n에 대해 O(n) 시간에 비교하기 위함이다. 변환 값은 A=10000, C=01000, G=00100, T=00010, N=00001로 각 base의 염기문자열을 bit array로 변환하여 리드 테이블에 함께 저장한다. 포지션 테이블은 서로 다른 윈도우가 동일한 32 bases 값을 갖는 경우가 발생하므로 각 윈도우의 생성 위치를 리드 테이블의 윈도우 해시키 값과 동일한 값을 키로 하는 포지션 테이블을 생성하고 32 bases가 동일한 윈도우의 위치 값만을 저장하도록 하여, 위치가 다른 곳에서 생성된 동일한 32 bases를 여러 번 비교하는 것을 방지하도록 구성하였다.

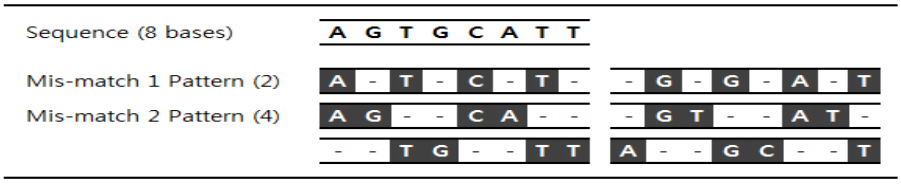
3.4 해싱 데이터 셋: 클러스터 테이블

리드 테이블의 데이터인 각 base를 변환한 bit array가 테스트 리드와 비교 대상이 되므로 미스매치를 탐색하기 위해 각 base의 패턴을 분석하여 미스매치 1인 경우에는 2가지 패턴으로 미스매치 1의 차이를 탐색할 수 있고 미스매치 2인 경우에는 4가지 패턴을 추가하여 미스매치 2까지 탐색할 수 있다. 미스매치 1인 경우는 각 base의 문자열을 처음부터 1 base씩 하나는 값을 알고, 다음 하나는 값을 무시하는 패턴으로 모든 미스매치 경우에 대한 해시 값을 구하고 미스매치 2인 경우에는 32 bases의 문자열을 처음부터 2 bases씩 처음 2개는 값을 알고, 다음 2개는 값을 무시하는 패턴으로 모든 미스매치 경우에 대한 해시 값을 구한다.

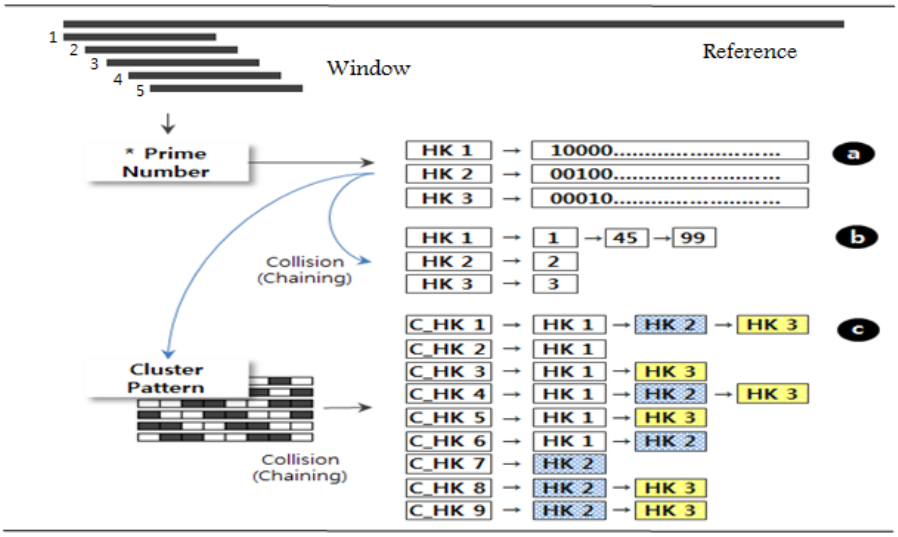
[그림 2]는 제안하는 클러스터링 패턴의 구성 방법 예를 보여준다. [그림 2]와 같이 미스매치 2까지

허용하기 위해 6개의 패턴을 만들고 패턴에 맞는 해시키 값과 리드 테이블의 키 값을 array로 갖는 클러스터 테이블을 생성한다. [그림 2]에서 8 bases가 미스매치 1개를 허용하는 경우는 2가지 패턴, 미스매치 2개까지 허용하는 경우에는 6가지 패턴을 사용하여 클러스터링 할 수 있으며 (-) 부분은 클러스터링을 위한 해싱키 값 생성 계산에서 제외 된다.

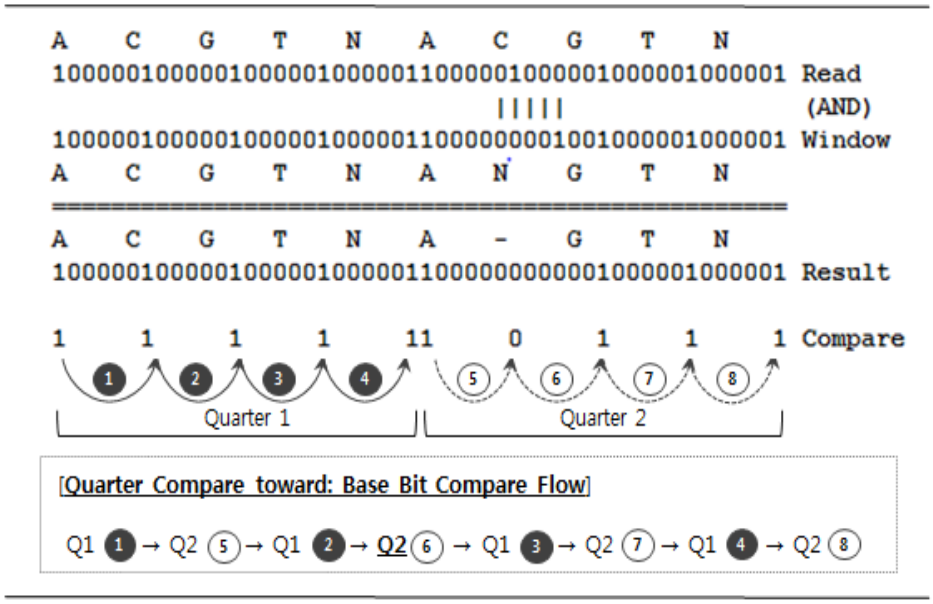
클러스터 테이블의 해시키 구조는 군집화를 위해 기본 해시 알고리즘에서 격자 모양의 소수 위치 값을 각 base 대체 숫자 값과 곱한 합을 정 방향과 역 방향으로 계산하여 구성된다. 앞서 언급한 [그림 1]에서와 같이 "AGAGA…AGCC"로 이루어진 32 bases 길이의 리드가 있다고 한다면 윈도우의 각 base 대체 숫자 곱하기 쌍둥이 소수의 합은 [그림 1의 ②]와 [그림 2]의 패턴에 따라 (26383: 2*11 + … + 3*827), 윈도우의 각 base 대체 숫자 곱하기 쌍둥이 소수의 역 위치 값의 합은 [그림 1의 ③]과 [그림 2]의 패턴에 따라 (17839: 2*827 + … + 3*11)을 조합하여 해시 값 (2638317839)으로 사용한다. 클러스터링 해시 알고리즘은 유사한 미스매치 형태를 가지는 해시키 값을 군집화 한다. [그림 3]은 제안하는 알고리즘을 수행하기 위해 레퍼런스 시퀀스로부터 구성되는 해시 데이터 셋([그림 3의 ⓐ] Read table, [그림 3의 ⓑ] Position table, [그림 3의 ⓒ] Cluster table) 구조를 보여준다.



[그림 2] 클러스터링 패턴 (8 bases - mis-match 2)



[그림 3] 레퍼런스 시퀀스로부터 k-size의 윈도우 분리와 해시 데이터 셋 생성



[그림 4] 비트의 AND 연산과 미스매치 Bit 비교 방법

3.5 비트 비교 연산

레퍼런스 시퀀스와 테스트 리드의 각 base 비교는 bit array를 이용하여 비교하여 speed를 높였다. {A, C, G, T, N}의 문자를 고유하게 식별하기 위해 각 문자는 5 bit로 구성이 되도록 하였다. 테스트 리드는 비교를 위해 해시키 값을 생성하여 리드 테이블과 동일한 해시키 값이 있는 경우에는 32 bases를 비트로 전환하여 비교한다. 이때 동일 base 위치 값이 다른 경우 해당 bit 위치 값이 0이 되고 같은 경우에는 1을 유지하는 AND 연산을 사용하고, 비교 연산 결과는 리드의 각 32 bases 의 bit 위치를 알고 있으므로 32 bit 만을 계산하여 미스매치가 2 이하인 경우에는 레퍼런스 윈도우를 32 bases의 bit array를 {A, C, G, T, N}의 염기문자로 복원하여 결과를 생성한다. [그림 4]는 미스매치 1인 경우의 매치 결과로 AND 연산 후 리드의 base 위치 bit가 0으로 변경되었으며, 동일한 레퍼런스 윈도우 base는 1로 유지되는 예를 설명한다.

[그림 4]의 레퍼런스 윈도우와 테스트 리드의 AND 연산 결과는 32 bases의 해당 비트 위치(32 bit)만을 Quarter로 나누고 각 Quarter의 base Bit만을 Quarter 순으로 비교하여 완전매치와 미스매치를 확인한다. 미스매치 2 이상인 리드는 bit 값 0이 2개 이상이 되므로, 미스매치는 Read size bit count > Read size – 3 경우 비교를 중단하고 다음 리드를 처리하도록 한다.

4. 실험 및 결과 분석

4.1 실험 데이터 및 환경

본 알고리즘의 성능 테스트 실험에 앞서 해시키 생성을 위해 염기문자의 대체 숫자 선택을 위한 실험과 소수 선택을 위한 실험을 수행하였다. 알고리

즘의 성능 테스트를 위해서 본 연구에서는 NCBI[19]에서 제공되는 hs_alt_HuRef_Chr21 염색체(31.4M byte)를 레퍼런스 시퀀스로 사용하였다. 실험 서버의 제원은 2.0GHz Intel CPU, 16GB memory 이었고 OS로는 Windows 7을 사용하였다. 실험을 위한 테스트 리드는 SAMtools[20]의 *wgsim*에서 노이즈가 없는 리드 1백만 건과 노이즈가 있는 리드 1백만 건을 각각 32, 28, 24 bases 길이로 생성하였다. 비교 알고리즘인 BWA도 동일한 레퍼런스 시퀀스와 리드를 사용하였고 제안하는 방법과 동일하게 미스매치 2까지 허용하여 실험하였으며 실행속도의 경우 제안하는 알고리즘과의 정확한 비교를 위해 3회 실험하여 평균 속도를 사용하였다. 4.2와 4.3은 본문 3.1과 3.2에서 언급한 해시키 값 생성을 위한 실험 결과를 제시한다.

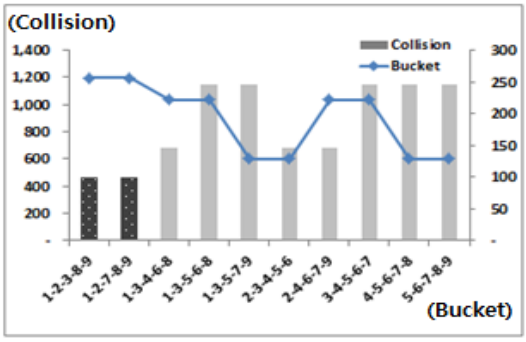
4.2 해시키 생성을 위한 염기문자의 대체 숫자 선택 실험 결과

[그림 5]는 5개의 문자{A, C, G, T, N}를 대체하기 위한 숫자 조합 시뮬레이션 결과이며, 최선, 최악, 중간 조건에 해당하는 숫자 집합의 결과를 보여준다. 실험결과 일반적으로 해시에서 사용하는 소수(2,3,7..) 집합이 선택될 것으로 예상하였으나, 가장 큰 수와 가장 작은 수의 집합인 {1,2,3,8,9}와 {1,2,7,8,9}이 버킷의 개수가 가장 많았고, 충돌이 가장 작게 나타났다.

4.3 해시키 생성을 위한 염기문자와 숫자의 매치 실험 결과

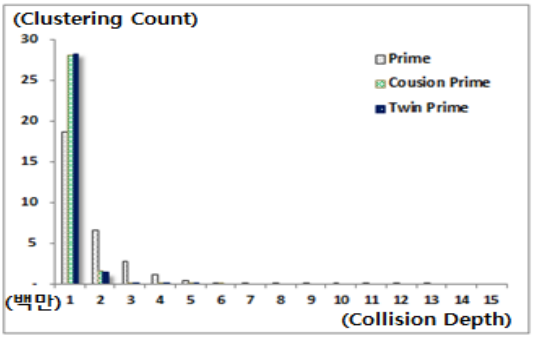
해시 알고리즘에서는 키 값의 유일성이 알고리즘의 우수성을 나타내는 척도가 될 수 있으므로, 소수,

Number Set	Bucket	Collision
1-2-3-8-9	257	465
1-2-7-8-9	257	465
1-3-4-6-8	222	678
1-3-5-6-8	222	1,143
1-3-5-7-9	129	1,143
2-3-4-5-6	129	678
2-4-6-7-9	222	678
3-4-5-6-7	222	1,143
4-5-6-7-8	129	1,143
5-6-7-8-9	129	1,143



[그림 5] 염기문자 대체 숫자 선택을 위한 시뮬레이션 결과

Collision	Prime	Cousion Prime	Twin Prime
1	18,561,516	28,025,411	28,199,753
2	6,637,032	1,532,886	1,378,323
3	2,717,234	86,660	68,448
4	1,095,053	4,477	3,024
5	418,835	228	121
6	149,028	8	2
7	49,834	1	
8	15,233		
9	4,317		
10	1,203		
11	299		
12	65		
13	15		
14	6		
15	1		



[그림 6] 소수, 사촌소수, 쌍둥이소수의 충돌 깊이와 유일성 실험 결과

사촌 소수, 쌍둥이 소수를 각각 사용하여 유일성과 충돌의 빈도를 실험하였다. 실험을 위해 레퍼런스 시퀀스에 대하여 슬라이딩 윈도우 기법으로 32 bases 길이의 윈도우를 생성하고 3장에서 기술한 방법대로 해시 값을 생성하였고 이에 대한 충돌 회수를 분석하였다.

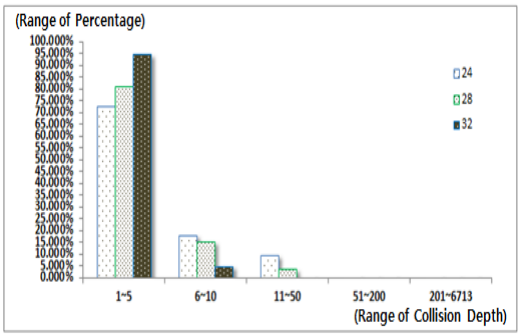
[그림 6]은 각 소수를 위치 값으로 사용하였을 경우 해시키 값의 충돌이 얼마나 발생하는지 보여주는 실험 결과이다. 결과는 소수와 사촌소수에 비해 쌍둥이 소수를 사용한 경우 전체 윈도우 중 95.1%가 유일한 해시 값을 갖는 등 가장 좋은 유일성을 갖는 것을 확인하였다.

4.4 클러스터 테이블의 군집화 실험 결과

클러스터 테이블은 미스매치 2까지 검색하기 위해 3.4에서 설명한 [그림 2]와 같이 패턴으로 32/28/24 bases의 윈도우를 분리하여 클러스터링 해시키 값을 생성하였으며, 군집화된 클러스터 테이블의 해시키 값이 유일(unique)할수록 리드와 비교되는 횟수가 작아지고, 이는 속도 개선에 영향을 주게 된다. 실험에서는 쌍둥이 소수를 사용하여 클러스터 테이블의 유일성과 충돌 빈도를 분석하였다.

[그림 7]은 패턴을 이용한 클러스터링 해시 알고리즘이 군집화의 유일성을 얼마나 유지하는지 보여주는 실험 결과이다. 결과는 윈도우의 길이가 길어질수록 클러스터 테이블의 유일성 밀도가 증가하는 것을 확인하였다. 특히 32 bases의 1~5구간에서 94.6%이상의 좋은 군집화 밀도를 갖는 등 검색 속도 향상을 위해 패턴을 이용한 클러스터링 해시 알고리즘이 비교적 좋은 것으로 확인되었다. 또한 6가지 패턴이 동일한 리드 테이블의 해시 값을 갖게 되므로 하나의 리드가 일치하는 동일한 리드 테이블의 해시키 값은 한 번만 비교하고 나머지는 무시하여 여러번 비교하는 것을 방지하도록 구성하였다.

	24 bases	%	28 bases	%	32 bases	%
1~5	26,052,172	72.445%	40,032,486	81.042%	73,096,628	94.679%
6~10	6,478,110	18.014%	7,543,396	15.271%	3,751,921	4.860%
11~50	3,373,227	9.380%	1,797,044	3.638%	336,960	0.436%
51~200	54,832	0.152%	22,123	0.045%	17,503	0.023%
201~6713	2,771	0.008%	2,047	0.004%	1,710	0.002%
SUM	35,961,112		49,397,096		77,204,722	



[그림 7] 클러스터 테이블의 군집화 실험 결과

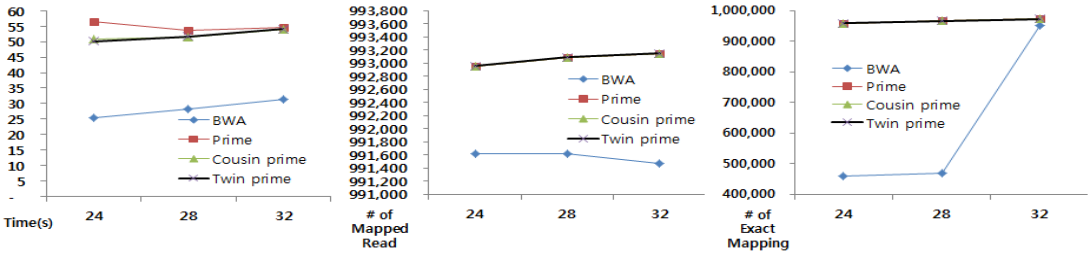
4.5 노이즈 없는 리드의 완전매치 후 클러스터 테이블 탐색 결과

첫 번째 실험으로 제안하는 방법에 노이즈가 없는 리드를 이용하여 정 방향과 역방향에서 리드 테이블과 포지션 테이블 만을 사용하여 완전매치를 우선 탐색하고, 완전매치 탐색이 실패한 경우에만 클러스터 테이블을 이용하여 불일치 2까지 탐색하여 BWA와 비교하였다. 본 알고리즘은 24, 28, 32 bases에서 모두 맵핑율과 오류율에서 BWA 보다 좋은 결과를 보였다.

특히 [표 2]와 [그림 8]의 실험결과 24와 28 bases는 BWA 대비 exact mapping이 50% 이상 높았으며, 32 bases에서는 2배 이상 더 낮은 오류율을 확인하였다. 또한 BWA에서는 리드의 크기에 따라 맵핑과 오류율의 편차가 심하게 나타나고 있으나,

[표 2] 노이즈 없는 리드의 완전매치 후 클러스터 테이블 탐색과 BWA 비교

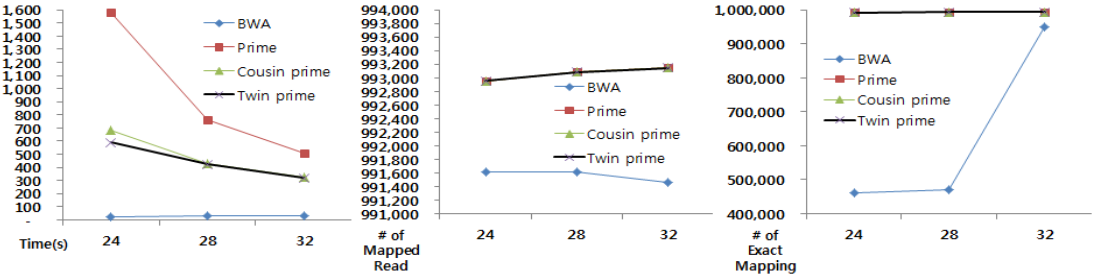
Single	24 bp read				28 bp read				32 bp read			
	Time (s)	# of mapped read	# of exact mapping	Err (%)	Time (s)	# of mapped read	# of exact mapping	Err (%)	Time (s)	# of mapped read	# of exact mapping	Err (%)
BWA	25.4	991,617	459,842	53.63	28.22	991,611	468,863	52.72	31.39	991,462	951,193	4.06
Prime	56.41	992,956	957,690	3.55	53.61	993,084	966,401	2.69	54.49	993,148	973,109	2.02
Cousin	50.89	992,956	957,690	3.55	51.46	993,084	966,401	2.69	54.14	993,148	973,109	2.02
Twin	50.19	992,956	957,690	3.55	51.72	993,084	966,401	2.69	54.04	993,148	973,109	2.02



[그림 8] 노이즈 없는 리드의 완전매치 후 클러스터 테이블 탐색과 BWA 비교

[표 3] 노이즈 없는 리드의 클러스터 테이블 탐색과 BWA 비교

Single	24 bp read				28 bp read				32 bp read			
	Time (s)	# of mapped reads	# of exact mapping	Err (%)	Time (s)	# of mapped reads	# of exact mapping	Err (%)	Time (s)	# of mapped reads	# of exact mapping	Err (%)
BWA	25.4	991,617	459,842	53.63	28.22	991,611	468,863	52.72	31.39	991,462	951,193	4.06
Prime	1581.92	992,956	992,956	0	763.79	993,084	993,084	0	510.37	993,148	993,148	0
Cousin	685.97	992,956	992,956	0	430.3	993,084	993,084	0	324.43	993,148	993,148	0
Twin	589.25	992,956	992,956	0	420.22	993,084	993,084	0	320.39	993,148	993,148	0



[그림 9] 노이즈 없는 리드의 클러스터 테이블 탐색과 BWA 비교

본 알고리즘에서는 리드의 크기에 상관없이 맵핑율이 안정적이고 오류율 또한 작게 나타났다.

4.6 노이즈 없는 리드의 클러스터 테이블에 의한 탐색 결과

두 번째 실험으로 제안하는 알고리즘에 대하여 노이즈가 없는 리드를 이용하여 정 방향과 역방향에 대하여 클러스터 테이블에서 매치 후보를 찾아 리드 테이블과 포지션 테이블에서 완전매치와 불일치 2까지를 동시에 탐색하도록 하여 실험 결과를 BWA의 경우와 비교하였다.

결과적으로 제안하는 방법은 24/28/32 bases에서 모두 매핑과 오류율에서 정확하였으며 특히 [표 3]에서 보는 바와 같이 본 방법을 적용했을 경우 24/28/32 bases에서 오류율이 모두 0% 였음을 확인하였다. 다만 32 bases에서 패턴으로 클러스터링된 미스매치 매핑 후보를 비교해야 했으므로 BWA보다 수행시간에 있어서 289초 증가 하였으나, 매우 많은 리드가 정확하게 맵핑되었다. BWA는 제안하는 방법에 비해 짧은 시간에 탐색을 완료하였으나, 매핑과 오류율이 제안하는 방법보다 좋지 못하였다. [표 3]과 [그림 9]은 실험 결과를 나타낸다.

4.7 노이즈가 있는 리드의 완전매치 후 클러스터 테이블 탐색 결과

세 번째 실험으로는 노이즈가 있는 리드를 이용하여 정 방향과 역방향에서 리드 테이블과 포지션 테이블 만을 사용하여 우선 완전매치를 탐색하고, 탐색이 실패한 경우에만 클러스터 테이블을 이용하여 미스매치 2까지 탐색하도록 하였다. BWA도 동일하게 노이즈가 있는 리드를 사용하여 실험을 수행하였고 결과를 비교하였다.

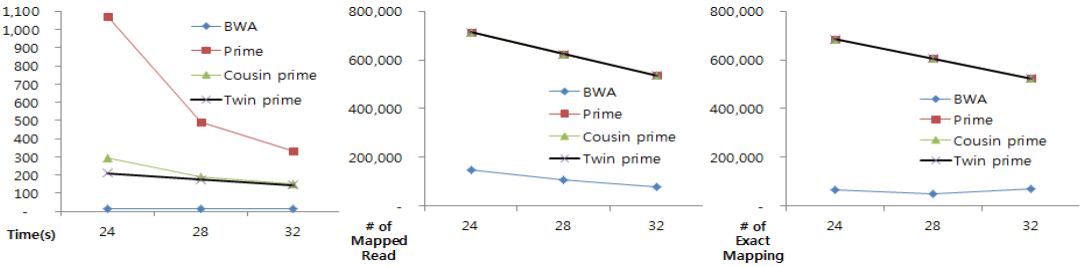
[표 4]에서와 같이 제안하는 방법의 수행 시간은 노이즈 없는 리드를 사용할 때 보다 증가하였으나, 24 bases에서 제안하는 방법이 BWA 보다 맵핑율의 경우 약 4.7배 높았으며 정확하게 맵핑된 것의 비율인 exact mapping의 경우도 약 10배 정도 높았고, 오류율 또한 14배 이상 좋게 나왔다. 32 bases에서는 본 알고리즘의 수행시간이 BWA에 비하여 약 129초 더 걸렸으나, 매핑율은 약 45.94% 높았고, 오류율은 2.2%로 BWA 보다 약 5%정도 낮은 수치를 보였다. 노이즈가 없는 리드의 실험 4.4와 4.5 보다 노이즈가 있는 리드의 실험에서 BWA 대비 제안하는 알고리즘이 맵핑율과 정확성이 보다 안정적이고 우수하게 차이를 나타나고 있다. 다만 BWA 보다 수행 시간이 상대적으로 증가하는 것은 맵핑량의 증가와 대상을 비교하는데 기인한 것이라 보인다. [표 4]와 [그림 10]는 실험 결과를 나타낸다.

4.8 노이즈가 있는 리드의 클러스터 테이블에 의한 탐색 결과

마지막으로 제안하는 알고리즘으로 노이즈가 있는 리드에 대하여 정 방향과 역방향에서 클러스터 테이블의 매치 후보를 찾아 리드 테이블과 포지션 테이블에서 완전매치와 미스매치 2까지를 동시에 탐색하도록 하였고, BWA도 동일하게 노이즈가 있는 리드에 대하여 실험을 수행하였다. 제안하는 방법은 24, 28, 32 bases에서 모두 매핑율과 오류율이 BWA보다 우수하였다. 또한 사용하는 리드의 크기가 커질수록 해시키와 클러스터 테이블의 해시키가 더욱 유일해지기 때문에 리드의 base 크기가 커질수록 탐색시간이 더욱 단축되었다. 본 실험에서 24

[표 4] 노이즈 있는 리드의 완전매치 후 클러스터 테이블 탐색과 BWA 비교

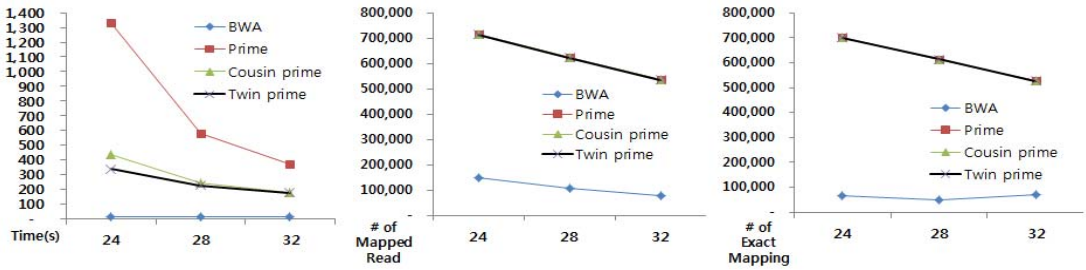
Single	24 bp read				28 bp read				32 bp read			
	Time (s)	# of mapped reads	# of exact mapping	Err (%)	Time (s)	# of mapped reads	# of exact mapping	Err (%)	Time (s)	# of mapped reads	# of exact mapping	Err (%)
BWA	13.9	149,051	64,812	56.52	15.05	106,434	48,284	54.63	15.34	75,750	70,125	7.43
Prime	1069.52	714,667	685,941	4.02	492.71	624,062	606,155	2.87	329.54	535,165	523,368	2.2
Cousin	296.38	714,667	685,941	4.02	186.96	624,062	606,155	2.87	148.9	535,165	523,368	2.2
Twin	210.2	714,667	685,941	4.02	173.61	624,062	606,155	2.87	145.1	535,165	523,368	2.2



[그림 10] 노이즈 있는 리드의 완전매치 후 클러스터 테이블 탐색과 BWA 비교

[표 5] 노이즈 있는 리드의 클러스터 테이블 탐색과 BWA 비교

Single	24 bp read				28 bp read				32 bp read			
	Time (s)	# of mapped reads	# of exact mapping	Err (%)	Time (s)	# of mapped reads	# of exact mapping	Err (%)	Time (s)	# of mapped reads	# of exact mapping	Err (%)
BWA	13.9	149,051	64,812	56.52	15.05	106,434	48,284	54.63	15.34	75,750	70,125	7.43
Prime	1332.44	714,667	700,249	2.02	579.79	624,062	613,886	1.63	372.69	535,165	527,519	1.43
Cousin	435.79	714,667	700,249	2.02	242.35	624,062	613,886	1.63	180.77	535,165	527,519	1.43
Twin	336.28	714,667	700,249	2.02	227.53	624,062	613,886	1.63	176.06	535,165	527,519	1.43



[그림 11] 노이즈 있는 리드의 클러스터 테이블 탐색과 BWA 비교

base 크기의 탐색보다 32 bases 탐색 시간이 약 2배 정도 수행 시간이 빨랐다. 매핑율의 경우 모든 리드 크기에 대하여 BWA보다 높았으며 완전매치 후 클러스터 테이블을 탐색한 경우와 큰 차이는 없었다. 오류율의 경우도 BWA 보다 모든 경우에 대해서 낮았으며 실험 4.7에서 나온 약 2.2% 에 비하여 실험 4.8에서는 1.43%로 더 낮아짐을 확인하였다.

결과적으로 노이즈가 있는 리드에서 제안하는 방법이 BWA 보다 매우 좋은 결과를 보임을 확인하였으며, 노이즈가 없는 리드와 있는 리드 두 경우에 대해서 모두 제안하는 방법이 BWA에 비하여 정확하게 탐색을 하고 있다는 것을 확인하였다. [표 5]와 [그림 11]은 실험 결과를 보여준다.

5. 결론

본 논문에서는 NGS에서 생성된 짧은 리드를 레퍼런스 시퀀싱에서 정확하게 탐색하기 위해 윈도우에 대한 해시키 값을 생성하고, 비트 비교 연산, 미스매치를 탐색하기 위한 클러스터링 방법을 제안하였다. 또한 속도와 정확도가 우수한 BWA 알고리즘과 비교하여 모든 실험에서 본 알고리즘이 BWA보다 좋은 맵핑율과 오류율을 보임을 확인하였다. 정확성과 탐색시간 사이에는 트레이드오프(trade-off) 관계가 존재하지만, 본 알고리즘은 절대적인 수치를 기준으로 실제 인간의 염색체를 레퍼런스로 사용하여 비교적 느리지 않은 탐색시간 안에서 BWA에 비하여 매우 안정적이고 정확하게 레퍼런스 시퀀스 상에서의 리드 위치를 탐색할 수 있다는 것이 특징이 있음을 확인하였다. 또한 본 해시 알고리즘과 클러스터링 방법은 리드의 크기가 커질수록 더 유일한 해시키 값을 생성하고 있으며, 최근 NGS에서 생성되는 리드의 크기가 커지는 추세에서도 비교적 안정적인 탐색시간을 유지할 수 있다.

일반적으로 해시 알고리즘을 사용하는 기존 방법과 같이 I/O를 줄이기 위해 사용되는 메모리 사용량이 크다는 단점이 있지만, 이는 향후 분산처리 등으로 해결할 수 있는 여지가 충분히 존재한다. 향후 분산처리와 함께 전체 유전체를 대상으로 해싱 데이터셋을 저장하기 위해 필요한 메모리 공간을 줄이는 연구를 진행할 계획이다.

6. 참고문헌

[1] 이종극, 2010. 질병 유전체 분석법 2, 서울: 월드사이언스.

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

[3] Altschul, S.F., et. al., "Gapped BLAST and PSI-BLAST: a new generation of protein database search program," *Nucleic Acids Res.* Vol.25, No.17, pp.3389-3402, 1997.

[4] R. S.C. Goble, P. Baker, and Brass, "A Classification of tasks in bioinformatics," *Bioinformatics,* Vol.17, No.2, pp.180-188, 2001.

[5] 김유선, 김종현, 여윤구, 김우철, 박상현. "다염기변이 유전체에 대한 서열 정렬 툴 분석". 한국컴퓨터종합학술대회 논문집 Vol.35, No.1(C), pp.217-221, 2008.

[6] Pearson, W.R., and D.J. Lipman,

"Improved tools for biological sequence comparison Proc," *Natl. Acad. Sci.* USA Vol.85, pp.217-221, April, 1988.

[7] Heng Li and Nils Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Bioinformatics,* Vol.II, No.5, pp.473-483, 2010.

[8] W. James Kent, "BLAT-The BLAST-Like Alignment Tool," *Genome Research,* Vol.12, No.4, pp.656-664, 2002.

[9] H. Li, J. Ruan, R. Durbin, "Mapping short DNA sequencing reads and calling variants using mapping quality scores," *Genome Research,* Vol.18, pp.1851-1858, 2008.

[10] H. Jiang, W. Wong, "SeqMap: mapping massive amount of oligonucleotides to the genome," *Bioinformatics,* Vol.24, No.20, pp.2395-2396, 2008.

[11] R. Li, Y. Li, K. Kristiansen, J. Wang, "SOAP: short oligonucleotide alignment program," *Bioinformatics,* Vol.24, No.5, pp.713-714, 2008.

[12] H. Eaves, Y. Gao. "MOM: maximum oligonucleotide mapping," *Bioinformatics,* Vol.25, No.7, pp.969-970, 2009.

[13] Heng Li, R Durbin. "Fast and accurate short read alignment with Burrows-Wheeler," *Bioinformatics,* Vol.25, No.14, pp.1754-1760, 2009.

[14] http://bio-bwa.sourceforge.net/

[15] http://en.wikipedia.org/wiki/Hash_function

[16] Marcus du Stautoy, 2004. The Music of the Primes: Searching to Solve the Greatest Mystery in Mathematics, April 27, Harper Perennial.

[17] David Wells, 2003. Prime Numbers: The Most Mysterious Figures in Math, John Wiley & Sons Inc., New Jersey

[18] Jing-run Chen, "On the representation of a large even integer as the sum of a prime and the product of at most two primes", *Scientia Sinica,* Vol.XXI, No.4, pp.421-430, 1978.

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

[20] Heng Li, et al., "The Sequence Alignment/Map format and SAMtools", *Bioinformatics,* Vol.25, No.16, pp.2078-2079, 2009.

경 규 호

1992년 서울과학기술대학교 전자계산학과 졸업(학사)

2012년 연세대학교 컴퓨터공학과 졸업(석사)

1989년 - 2009년 데이콤 부장

2010년 - 현재 엘지유플러스 부장

관심분야 : 데이터베이스, 데이터 마이닝, 바이오인포매틱스, 빅 데이터, 데이터웨어하우스, ERP, CRM

박 치 현

2007년 홍익대학교 컴퓨터공학과 졸업(학사)

2009년 연세대학교 컴퓨터과학과 졸업(석사)

2009년 – 현재 연세대학교 컴퓨터과학과 박사과정

관심분야 : 시스템 생물학, 바이오인포매틱스, 데이터 마이닝, 데이터베이스 시스템

여 윤 구

2009년 연세대학교 컴퓨터과학과 졸업(학사)

2011년 연세대학교 컴퓨터과학과 졸업(석사)

2011년 – 현재 연세대학교 컴퓨터과학과 박사과정

관심분야 : 바이오인포매틱스, 데이터 마이닝, 데이터베이스 시스템

박 상 현

1989년 서울대학교 컴퓨터공학과 졸업(학사)

1991년 서울대학교 대학원 컴퓨터공학과(공학석사)

2001년 UCLA 대학원 컴퓨터과학과(공학박사)

1991년 – 1996년 대우통신 연구원

2001년 – 2002년 IBM T. J. Watson Research Center Post-Doctoral Fellow

2002년 – 2003년 포항공과대학교 컴퓨터공학과 조교수

2003년 – 2006년 연세대학교 컴퓨터과학과 조교수

2006년 – 2011년 연세대학교 컴퓨터과학과 부교수

2011년 – 현재 연세대학교 컴퓨터과학과 교수

관심분야 : 데이터베이스, 데이터 마이닝, 바이오인포매틱스, 적응적 저장장치 시스템, 플래쉬메모리 인덱스, SSD