ISSN 1598-9798

데이타베이스연구

28권 제1호 2012년 4월

차세대 시퀀싱 리드를 위한 서열 정렬 알고리즘에 관한 비교 연구

A Survey of Sequence Alignment Algorithms for Next-Generation Sequencing Read

여윤구, 박치현, 안재균, 박민서, 김판규, 박상현

Yunku Yeu, Chihyun Park, Jaegyoon Ahn, Minseo Park, Pankyu Kim, Sanghyun Park

데이터베이스 소사이어티

Database Society

사단법인 한국정보과학회

The Korean Institute of Information Scientists and Engineers

차세대 시퀀싱 리드를 위한 서열 정렬 알고리즘에 관한 비교 연구

A survey of sequence alignment algorithms for next-generation sequencing read

여윤구(Yunku Yeu)[1], 박치현(Chihyun Park)[2], 안재균(Jaegyoon Ahn)[3], 박민서(Minseo Park)[4], 김판규(Pankyu Kim)[5], 박상현(Sanghyun Park)[6]

요 약

서열 정렬 알고리즘은 유전학에서 널리 사용되는 도구 중 하나다. 차세대 시퀀싱 기술이 발달함에 따라 높은 처리량을 갖는 염기 서열 정렬 알고리즘이 다수 개발되었다. 차세대 시퀀싱 기술을 위한 서열 정렬 알고리즘은 크게 해시 기반 알고리즘과 BWT 기반 알고리즘으로 구분할 수 있다. 본 논문은 해시 기반 알고리즘과 BWT 기반 알고리즘의 대표 연구들을 자세히 설명하고, 시뮬레이션 데이터를 이용하여 두 카테고리의 성능을 비교하였다. 해시 기반 알고리즘은 단일 염기 변이가 많은 데이터 셋에서 더 많은 리드를 성공적으로 정렬했으며, BWT 기반 알고리즘은 변이가 적은 데이터 셋에서 동등한 수준의 정확도를 가지면서도 더 빠른 처리 속도를 나타냈다.

주제어 : 염기 서열 정렬, 차세대 시퀀싱

---

1) 연세대학교 컴퓨터과학과, 박사과정
2) 연세대학교 컴퓨터과학과, 박사과정
3) 연세대학교 컴퓨터과학과, 박사과정
4) 삼성 SDS CSP연구소 Bioinformatics Lab, 연구원, 공동 교신저자
5) 삼성 SDS CSP연구소 Bioinformatics Lab, 연구원
6) 연세대학교 컴퓨터과학과 교수, 공동 교신저자
† 이 논문은 2011년도 삼성SDS의 산학협력사업(2011. 8. ~ 2012. 6.) 지원을 받아 수행된 것임.
\+ 논문접수: 2011년 12월 10일, 심사완료: 2012년 4월 6일

Abstract

Sequence alignment is one of the popular tools for genomics. A number of sequence alignment tools have been developed to support Next-Generation sequencing(NGS) read. Sequence alignment tools for NGS read can be classified into two major categories, hash-based and BWT-based. In this paper, we describe representative tools of those categories, and benchmark alignment tools using simulated dataset. As a result, the hash-based tools aligned more reads than the BWT-based tools in high mutation rate. The BWT-based tools showed faster alignment speed than the hash-based tools with equivalent accuracy in low mutation rate.

Keywords : Sequence alignment, Next-Generation Sequencing

1. 서 론

서열 정렬(alignment) 알고리즘은 유전학(genomics) 연구에 있어서 가장 기본적인 도구 중 하나다. 특히 참조 조립(reference assembly)과 변이(polymorphism) 탐색에 관한 연구 등에서 서열 정렬 알고리즘이 핵심적인 역할을 수행한다.

어떤 종의 유전 정보를 나타내는 염기 서열을 얻기 위해서는 먼저 작은 염기 서열 조각인 리드(read) 서열을 얻은 뒤, 리드 서열들을 기반으로 전체 염기 서열을 복원해 내야 한다. 염기 서열을 복원하는 방법은 드 노보 조립(de novo assembly)과 참조 조립으로 구분할 수 있다. 드 노보 조립은 리드 서열 간의 유사성 정보를 이용하여 리드를 하나의 큰 염기 서열로 조립해 내는 방식이다. 참조 조립은 리드 서열과 이미 완성된 염기 서열인 참조 서열(reference sequence) 간의 유사성을 이용하여, 리드가 원래 염기 서열의 어느 위치에서 생성되었는지를 탐색하는 방식이다. 여기서 참조 서열은 리드를 생성한 개체(individual)와 동일한 종(species), 또는 계통적으로 근접한 종의 것을 사용하며, 참조 서열과 리드 서열의 유사성을 파악하기 위해서 염기 서열 정렬 알고리즘을 사용한다. 특히, 같은 종의 참조 서열을 사용한다 하더라도 리드를 생성한 개체와 참조 서열을 생성한 개체 간에는 유전적 다양성(variation)이 존재할 수 있다. 따라서 참조 조립을 위해 염기 서열 정렬 알고리즘을 사용할 때에, 일정한 불일치(mismatch)를 허용하면서 서열 정렬을 수행하여야 한다.

서열 정렬 알고리즘의 성능은 처리량(throughput)과 정확도(accuracy)로 나타낼 수 있다. 서열 정렬 알고리즘의 처리량은 같은 시간 내에 보다 많은 리드를 정렬하는 능력을 가리키며, 정확도는 시퀀싱 과정에서의 에러와 실제 염기 서열의 변이에 적절히 대응하면서 리드의 정확한 정렬 위치를 찾는 능력을 가리킨다.

초기 유전체 연구에서는 리드의 생산 비용이 상대적으로 높았고 리드 하나의 길이가 더 컸기 때문에, 처리량보다는 정확도를 높이는 것이 더 중요한 문제였다. 그러나 2007년경 NGS (Next Generation Sequencing) 기술이 개발되면서 리드의 생산 비용이 감소하고 리드 하나의 길이가 짧아진 반면, 리드의 산출량은 비약적으로 증가했다.

이와 같은 환경의 변화로 서열 정렬 알고리즘에 대한 성능 평가 기준도 변화하게 되었다. 모든 리드를 참조 서열에 정렬하려는 노력보다는, 일부 리드의 맵핑을 포기하더라도 많은 리드를 빠르게 정렬하는 방법론이 NGS 기술에 더 적합한 접근법이 되었다. 리드의 일부가 맵핑되지 않더라도 더 많은 수의 리드를 저렴하게 생성하여 정렬함으로써 보완할 수 있기 때문이다. 또한 리드의 길이가 짧아졌기 때문에 서열 정렬 작업이 시퀀싱 과정의 에러와 변이에 더 민감하게 영향을 받게 되었고, 이로 인해 모든 리드를 참조 서열에 정렬하는 것이 더 어려운 문제가 되었기 때문이다.

특히 비약적으로 증가한 리드의 산출량은 정렬 알고리즘의 패러다임을 바꾸는 중요한 원인이 되었다. 일반적으로 하나의 유전체를 완전히 참조 조립하기 위해서 전체 유전체의 수십 커버리지(coverage) 정도의 리드를 생산한다. 인간의 유전체(30억 bp)를 기준으로 했을 때 이 경우 NGS 머신으로 생산되는 리드의 개수는 수십

억 개에 달한다. 이런 리드를 감당하기 위해서는 정렬 알고리즘이 높은 처리량을 갖추는 것이 필수적이다. BLAST[1] 등 기존의 정렬 알고리즘들은 주로 스미스-워터맨 (smith-waterman) 알고리즘[2] 등 동적 프로그래밍(dynamic programming)에 기초한 정렬을 수행하는 알고리즘로서, 시퀀싱 에러나 변이에는 잘 대응할 수 있지만 계산량이 너무 커서 NGS 기술의 리드 산출량을 감당하기에 어려움이 있었다.

NGS 환경의 높은 리드 산출량을 감당하기 위해, 서열 정렬 알고리즘들은 크게 두 가지 방법으로 발전하였다. 해시(hash) 기반 정렬 알고리즘과 BWT(Burrows-Wheeler Transform)[3] 기반 정렬 알고리즘이 그것이다. 두 가지 모두 빠르고 정확한 일치 정합(exact matching)을 보장하며, 각각 다른 방법으로 불일치 정합(inexact matching)을 지원한다.

해시 기반 알고리즘은 참조 서열과 리드 서열을 각각 잘게 조각내고 해시 함수를 이용해 해시 테이블에 맵핑(mapping)한다. 두 서열의 일부가 해시 테이블의 같은 버킷(bucket)에 맵핑되는 것은 두 서열간의 유사성 정보를 의미한다.

BWT 기반 알고리즘은 BWT를 이용하여 참조 서열을 접미사(suffix)의 독특한 퍼뮤테이션(permutation) 형태로 변환한다. 이 변환된 형태는 동일한 문자가 연속하여 등장하는 특징을 갖고 있으며, 이를 이용하여 부분 문자열의 등장 구간을 검색함으로써 쿼리 서열의 존재 여부를 파악할 수 있다.

NGS 기술을 이용하여 유전체 분석, CNV(copy number variation) 탐색 등의 유전체 연구를 수행하거나, NGS 리드 정렬 알고리즘을 새로이 개발하기 위해서는 기존에 제시된 다양한 정렬 알고리즘에 대한 심도 있는 이해와 성능 비교가 필요하다. 본 논문은 이와 같은 필요에 의해 선행 기술의 알고리즘 상의 특성과 정렬 성능을 비교 분석한다. 본 논문의 2장) 해시 및 BWT 기반 정렬 알고리즘의 대표적 연구에 대한 설명과, 3장) 시뮬레이션 데이터를 이용한 성능 비교 실험 및 분석, 4장) 결론 으로 구성되어 있다.

2. 대표적인 서열 정렬 알고리즘

2.1. 해시 기반 알고리즘

해시 기반 알고리즘은 참조 서열과 리드 서열을 일정한 구간으로 분절한 뒤, 해시 함수를 적용한 결과에 따라 해시 테이블에 분배하고, 같은 버킷에 분배된 서열들 사이에서 유사성을 탐색하는 알고리즘이다. 해시 기반 알고리즘에서 가장 중요한 요소는 해시 테이블의 구성과 불일치 정합을 수행하는 방법이다. 해시 테이블의 구성이 중요한 이유는, 포유류의 유전체에서 동일한 염기 서열이 반복되는 구간인 리피트(repeat)가 다수 존재하기 때문이다. 이러한 리피트 때문에 해시 테이블에는 필연적으로 충돌(collision)이 발생하게 되고, 이것은 해시 알고리즘의 이상적인 성능인 O(1)을 제한하는 요소가 된다. 다음으로, 해시는 기본적으로 일치 정합을 탐색하는 기법이기 때문에 불일치 정합을 수행하기 위한 대응책이 필요하다. 리드 서열의 단 한 부분에만 에러나 변이가 발생하면 리드의 해시 값은 원래의 해시 값과 완전히 동떨어진 값을 가질 수 있다. 해시 기반 정렬 알고리즘들은 각기 다른 방법으로 이와 같은 두 가지 문제를 해결하고 있다.

참조 서열 CCCGA**TTCAGTAA**TG

리드 서열 T<u>G</u>CAG<u>G</u>AA

템플릿	11110000 00001111	11000011 00111100	11001100 00110011
해시 값	TGCA - - - - - - - -GGAA	TG - - - -AA - -CAGG- -	TG- - GG- - - -CA - -AA

참조 서열 CCCGA**TTCAGTAA**TG

템플릿 적용	TGCA - - - -(miss) - - - -GGAA(miss)	TG - - - -AA(miss) - -CAGG - -(miss)	TG- - GG- -(miss) - -CA - -AA(hit)

[그림 1] MAQ 알고리즘의 개요. 리드 서열에서 밑줄이 쳐진 부분은 변이 또는 에러로 인하여 불일치가 발생한 위치를 의미한다. MAQ는 먼저 템플릿을 이용하여 리드 서열에서 해시 값을 추출한다. 이 그림에서는 6개의 템플릿을 사용하여 6개의 해시 값을 추출하였다. 리드에서 해시 값을 추출하고 나면, 참조 서열을 1base씩 이동하면서 리드에 적용한 템플릿을 1쌍씩 적용한다(총 3회). 6개의 템플릿을 사용했기 때문에 최대 2개의 불일치를 검색할 수 있으며, 본 예제에서는 마지막 템플릿에서 hit가 발생하였다.

MAQ[4]는 리드 서열로 해시 테이블을 구축한 뒤, 구축한 해시 테이블에서 참조 서열을 검색하는 방식의 정렬 알고리즘이다(그림 1 참조). MAQ는 일정한 개수의 템플릿(template) 쌍을 이용해 리드에서 시드(seed)를 추출, 해싱한다. 예를 들어 8bp의 리드가 있을 때, 최대 2개의 불일치를 허용하기 위한 템플릿 쌍은 (11110000, 00001111), (11000011, 00111100), (11001100, 00110011)으로 표현할 수 있으며, 이 템플릿들은 1의 위치에 해당하는 리드의 일부를 프로젝션(projection)한다. 즉, 첫 번째 템플릿 쌍의 첫 번째 템플릿은 리드의 앞부분 4bp에 해당하는 서브시퀀스를 시드로 만들어 해싱하고, 두 번째 템플릿은 뒷부분의 4bp를 시드로 만들어 해싱을 하는 식이다. 이와 같은 템플릿 구조는 전체 리드를 4등분으로 나눈 형태를 의미하며, 2개 이하의 불일치를 갖는 정렬을 찾기 위해서는 네 개의 조각 중 적어도 2개 이상의 조각은 완전히 일치해야 한다. 따라서 네 조각 중 2개 조각의 조합을 모두 탐색함으로써 2개 이하의 불일치를 갖는 모든 불일치 정합을 탐색할 수 있다.

해싱된 값과 해싱한 위치를 이용하여 해시 테이블을 구축하고 나면, 같은 템플릿 쌍을 이용하여 참조 서열을 1bp 단위로 스캔하면서 해시 테이블에서 맵핑 가능한 위치를 탐색한다. 맵핑 가능한 해시 테이블을 발견하면, 해싱 값에 해당하는 리드의 서열 전체와 참조 서열의 현재 위치를 비교하여 불일치의 위치와 개수를 파악하고, 불일치 영역의 시퀀싱 퀄리티(quality) 값을 이용하여 정렬 점수를 계산한다. 한 번의 스캔이 끝나면 다음 템플릿 쌍을 이용해서 해시 테이블을 구축하고 같은 작업을 반복한다. 6개의 서로 다른 템플릿 쌍을 사용할 경우 2개 이하의 불일치를 가진 모든 맵핑을 찾을 수 있으며, 20개를 사용할 경우 3개 이하의 불일치를 가진 모든 맵핑을 찾을 수 있다.

[표 1] MAQ의 성능 관련 파라미터

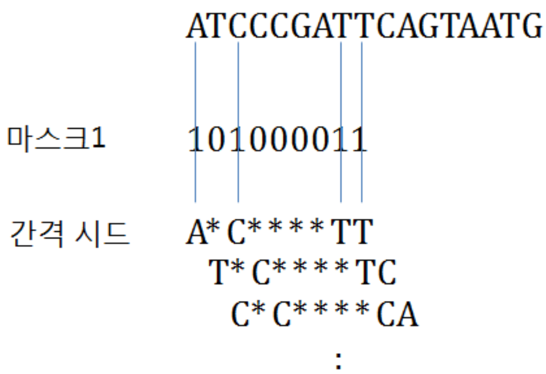
파라미터	설 명	초기값
−n	리드의 첫 24bp에서의 최대 불일치 개수	2
−m	리드 서열과 참조 서열의 불일치 비율	0.001
−e	불일치 영역에서 퀄리티 값의 최대 허용치	70

MAQ는 참조 서열을 여러 번 스캔하는 만큼, 빠른 정렬을 수행하기는 어렵다. 또한 허용하는 불일치의 개수가 커질수록 더 많은 횟수의 스캔을 수행해야 하며, 멀티스레딩을 지원하지 않는다. 또한 참조 서열이 아니라 리드 서열을 해싱하기 때문에 새로운 리드 서열이 입력될 때마다 해시 테이블을 새로 구축해야 한다는 단점을 있다. MAQ가 성능 튜닝을 위해 제공하는 대표적인 파라미터는 아래 표 1과 같다.

BFAST[5]는 접미사 배열(suffix array)와 간격 시드(spaced seed) 개념을 적용하여 불일치를 찾는 알고리즘이다. 간격 시드란 아래 그림 1과 같이 시드의 일정한 영역을 와일드카드 또는 전체 뉴클레오티드 4종류의 부분집합으로 정의하는 것을 뜻한다. 즉, 그림 1에서 *로 표시된 뉴클레오티드는 A, T, G, C 중 어느 것이어도 그 시드에 해당하는 것이 된다.

BFAST는 먼저 참조 서열을 분석하여 인덱스를 구축한다. 인덱스를 구축하기 위해서는 참조 서열의 모든 접미사를 수집한 뒤 알파벳 순서로 정렬한다. 이후 수집한 접미사들에 여러 가지 마스크(mask)를 적용해서 그에 해당하는 간격 시드를 얻어낸 뒤, 참조 서열에서 각 시드의 등장 위치를 수집하여 인덱스로 구축한다. 이후 리드가 입력되면 인덱스 구축에 사용한 것과 같은 마스크를 리드에도 적용하여 간격 시드를 얻어낸 다음, 이것을 참조 서열의 시드와 해시를 이용해 비교함으로써 CAL(Candidate Alignment Location)을 찾아낸다. 이후 찾아낸 CAL과 리드 서열 간에 스미스−워터맨 알고리즘을 수행하여 정렬을 완료한다.

BFAST는 시드를 이용해 맵핑 가능한 위치를 줄인 뒤 서열 정렬을 수행함으로써 처리 속도를 빠르게 한다. 또한 마스크 셋으로 구축한 여러 개



[그림 2] 간격 시드의 예. 이 경우 1, 3, 8, 9번째 뉴클레오티드를 선택한다. 나머지 뉴클레오티드는 어떤 것이어도 같은 간격 시드가 된다.

의 인덱스를 주 인덱스(main index)와 2차 인덱스(secondary index)로 구분하여, 먼저 주 인덱스를 이용해 CAL을 찾고, CAL을 찾지 못한 것에 대해서만 2차 인덱스를 검색하는 방법을 적용하고 있다. 이를 통해 여러 개의 인덱스를 반복 탐색하는 시간을 절약할 수 있다. 또한 BFAST는 간격 시드를 이용해 불일치 정합을 수행할 수 있다. 간격 시드에서 와일드카드로 처리되는 부분은 불일치가 있더라도 정렬이 가능하며, 다양한 마스크 셋을 이용하여 여러 가지의 간격 시드를 적용함으로써 다양한 종류의 에러에 대응할 수 있다.

BFAST의 성능 튜닝은 어떠한 마스크 셋을 쓰는가에 따라 결정된다. 많은 경우의 수를 포함할 수 있는 다수의 마스크 셋을 포함시킨다면 더 정확한 정렬을 수행할 수 있으나 실행 시간을 더 많이 소모하게 될 것이다. 반대로 적은 수의 마스크 셋을 포함시킨다면 알고리즘의 실행 시간이 짧아지는 대신 정확도는 감소할 것이다. BFAST에서 기본적으로 제공하고 있는 마스크 셋은 표 2와 같다.

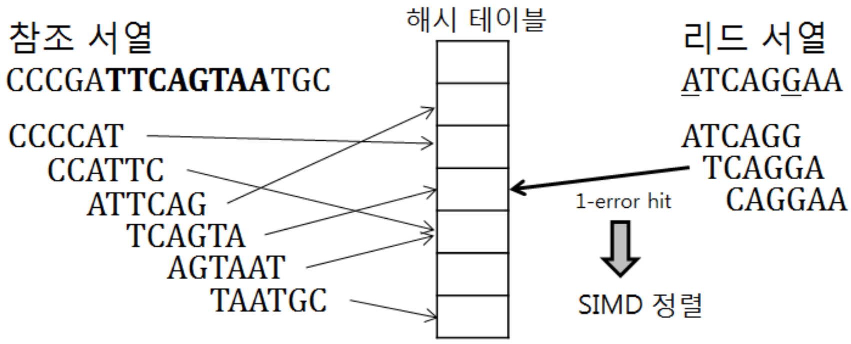
BFAST의 가장 큰 문제점은 간격 시드의 성능을 신뢰할 수 없다는 점이다. 간격 시드를 아무리 잘 정의한다 하더라도 결국 가능한 모든 변이의 부분집합만을 정의하는 것에 불과하다. 또한 각각의 간격 시드마다 별도의 인덱스를 구축해야 하기 때문에 다른 알고리즘에 비해 공간 복잡도가 높다.

[표 2] BFAST에서 제공하는 인간 유전체에 대한 마스크셋 예시 (중간 정도 정확도)

마스크	인덱스 구분
111111111111111111	주 인덱스
111110111011101010010101101111 1	2차 인덱스
10111110101101001011000011010001111111	2차 인덱스
101110011010011001001111010100 01011111	2차 인덱스

Stampy[6]는 참조 서열에서 일정 길이의 서열(k-mer)을 채취하여 해시 테이블을 구축한다(그림 3 참조). 이 때 과도한 메모리 사용을 막기 위해, k-mer를 듬성듬성하게 오버랩(overlap)되도록 하여 생성되는 서열의 개수를 줄인다. Stampy는 해시 테이블의 충돌을 방지하기 위해서 오픈 어드레싱 해싱(Open Addressing Hashing)을 적용하며, 참조 서열을 한 번 스캔하면서 특정 k-mer가 너무 많은 위치에서 나타나면 그것을 반복 시퀀스로 간주하여 해시 테이블을 구축할 때 해당 k-mer를 더 이상 고려하지 않는다. 또한 첫 번째 스캔에서 충돌로 인한 체인을 감지하면, 크기가 가장 큰 체인이 다른 체인 때문에 해싱에 방해를 받는 일이 적어지도록 체인의 길이가 가장 큰 것부터 먼저 해시 테이블에 저장한다.

해시 테이블을 구축한 이후에는 리드에서 가능한 모든 k-mer를 오버랩을 허용하면서 생성한 뒤, 해시 테이블을 이용해 맵핑 가능한 위치를 검색한다. 이 때 k-mer 중 1/2(리드의 길이가 49bp 이하인 경우) 또는 1/3(리드의 길이가 50bp 이상인 경우)에는 1개의 불일치가 발생한 경우까지 고려하여 맵핑 가능한 후보 위치를 탐색한다. 후보 위치를 탐색한 뒤에는 x86 SIMD 명령어(Single instruction, Multiple data instruction)를 이용한



[그림 3] Stampy의 알고리즘 개요. Stampy는 먼저 참조 서열에서 일정한 간격으로 k-mer를 추출하여 해시 테이블을 생성한다(본 예제에서 k=6, 간격=2). 리드 서열에서는 1base마다 k-mer를 추출하여 해시 테이블을 탐색하며, 이 때 k-mer 중 일부에는 1개의 불일치가 발생한 경우까지 고려하여 리드의 2번째 k-mer가 hit하였다. 해시 테이블에서의 검색이 성공하면 SIMD 정렬을 통해 나머지 부분의 서열 정렬을 완성한다.

정렬 알고리즘을 수행하여 리드 전체에 대한 최종 정렬 결과를 얻는다.

Stampy의 장점은 리드를 작게 조각낸 뒤 1개의 불일치 한도 내에서 가능한 모든 서열을 생성하기 때문에 정확한 정렬 작업을 수행할 수 있다는 점이다. 반면 다른 해시 기반 알고리즘에 비해 계산량이 더 많아지게 된다는 단점이 있다. Stampy의 또 다른 단점은 리피트에 약점을 보인다는 점이다. 리피트의 경우 해시 테이블 내에서 체인을 형성하게 되고, 이 체인으로 인해 맵핑 속도가 크게 저하된다. 때문에 이 알고리즘의 저자는 반복되는 시퀀스에 대해서는 BWT 기반 정렬 알고리즘인 BWA[7]를 활용하는 하이브리드(hybrid) 모드를 권장하고 있다. Stampy에서 제공하는 성능 관련 파라미터는 표 3과 같다.

[표 3] Stampy의 성능 관련 파라미터

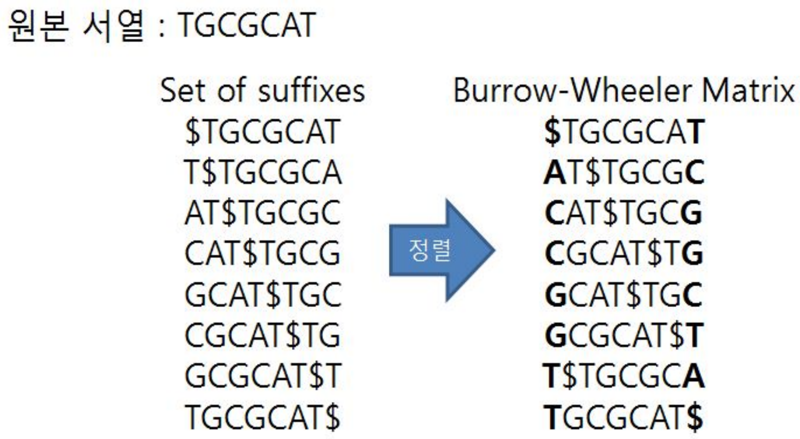
파라미터	설 명	초기값
--gapopen	갭(Gap)이 처음 생성될 때의 감점	40
-gapextent	생성된 갭이 확장될 때의 감점	3
-substitutionrate	리드와 참조 서열 사이의 최대 불일치 비율	0.001

2.2. BWT 기반 알고리즘

이 카테고리에 해당하는 알고리즘들은 BWT에 기반한 일치 정합을 수행한다. 이론적으로 BWT를 이용하면 O(n)의 시간복잡도 내에 모든 일치 정합을 수행할 수 있다 (n = 리드의 길이). BWT 기반 알고리즘의 성능 차이를 결정하는 중요한 요인은 불일치 정합을 수행하는 방식이다.

먼저 살펴볼 Bowtie[8]는 BW 행렬과 Last First mapping 법칙을 이용하여 일치 정합을 수행한다. Last First mapping 법칙은 다음과 같이 설명할 수 있다.

정의 1. Last First mapping BW 행렬의 마지막 컬럼에서 i 번째로 등장하는 문자는 BW 행렬의



[그림 4] Burrows-Wheeler 행렬의 구축. 우측 BW 행렬에서 굵게 표시된 부분만 저장한다.

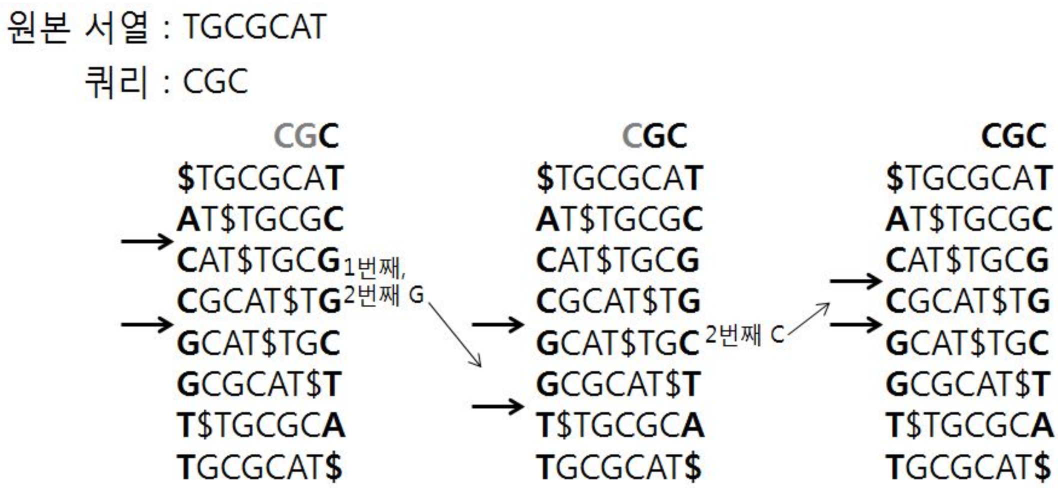
첫 번째 컬럼에서 i 번째로 등장하는 문자와 동일한 문 자이다.

Bowtie의 일치 정합 과정은 그림 4와 5를 통해 설명할 수 있다. 먼저 그림 4는 Bowtie가 BW 행렬을 구축하는 과정이다. 먼저 원본 시퀀스의 모든 접미사를 생성하고, 접미사에 해당하지 않는 나머지 서열을 문자열의 뒷부분에 연결한다. 문자 $는 서열의 끝을 의미하는 기호다. 그 이후에는 각 접미사를 알파벳 순으로 정렬한 다음 행렬에서 굵게 표시된 첫 번째 컬럼과 마지막 컬럼만을 보관한다.

그림 4와 같이 BW 행렬을 구축하면 그림 5와 같이 일치 정합이 수행된다. 일치 정합은 길이 n의 쿼리 서열 X의 마지막 문자 X[n-1] 부터 시작한다.

1) 먼저 행렬의 첫 번째 컬럼에서 쿼리 서열의 X[n]의 위치를 검색한다. (C의 위치를 검색하여 길이가 2인 구간을 얻는다.)
2) 찾아낸 구간의 마지막 컬럼에서 X[n-2]을 찾는다. (다음 문자인 G의 구간을 찾는다. 여기서는 구간에 있는 2개의 접미사 모두 G를 마지막 컬럼에 갖고 있다.)
3) 구간 내에서 발견한 X[n-2]들이 마지막 컬럼에서 i~j번째로 등장하는 지 파악한다. (다음 문자인 G는 마지막 컬럼에서 1~2번째로 등장하는 G이다.)
4) 첫 번째 컬럼에서 i~j번째 X[n-2]로 이동한 뒤, 2)부터 다시 시작한다. (LF mapping에 따라 마지막 컬럼에서 1~2번째로 등장하는 G는 첫 번째 컬럼에서도 1~2번째로 등장한다. 가운데 그림과 같이 이동한 뒤, 다음 문자인 C의 구간을 탐색한다.)

이와 같은 단계를 그림 5의 가장 오른쪽 그림까지 반복하면, 첫 번째 컬럼에서 2번째 C에 해당하는 구간을 찾을 수 있다. 이 시점에서 전체 쿼리에 대한 모든 탐색이 끝났으며, 검색된 구간 (길이 = 1)을 일치 정합의 결과로 출력한다. 검색된 구간 내에 속한 접미사는 원본 서열의 3번째 위치에서 시작한 접미사이며, 이것이 쿼리인 CGC가 존재하는 위치이다.



[그림 5] BW 행렬을 이용한 Bowtie의 일치 정합 과정

Bowtie의 불일치 정합은 그리디(greedy) 알고리즘 형태의 백트래킹(backtracking)을 통해 수행되는데, 이 때 허용되는 에러는 치환(substitution) 뿐이다. 어떤 시점에서 일치 정합이 실패하면 지금까지 정렬된 위치 중 시퀀싱 퀄리티(quality) 값이 가장 낮은 위치의 뉴클레오티드를 다른 3개의 뉴클레오티드로 바꾸고, 그 위치에서부터 다시 일치 정렬을 시도한다. 만약 다른 뉴클레오티드로 변경했을 때 정렬이 성공하면 1-불일치 정합을 발견한 것이다. 이와 같은 방식으로 최대 k개의 불일치를 허용하는 서열 정렬을 수행한다. 이러한 일련의 과정이 그리디 알고리즘의 형태로 이루어지기 때문에, Bowtie는 이상적인 정렬을 찾는 것을 보장하지 못한다.

Bowtie는 2가지 모드의 불일치 정합을 지원하는데, 그 중 하나는 기본 모드인 -n 모드로서 리드의 첫 L bp 내에 n개(0~3)의 불일치만을 허용하며, 이와 함께 불일치된 부분의 퀄리티 값의 합을 반영하여 정렬 작업을 수행하는 형태다. 다른 모드는 -v 모드로서 퀄리티 값을 사용하지 않고 시퀀스에서 최대 v개(0~3)의 불일치를 허용하는 모드이다. -v 모드에서는 E, L 등의 파라미터를 사용할 수 없다. Bowtie의 성능과 관련된 파라미터는 표 4와 같다.

[표 4] Bowtie의 성능 관련 파라미터

파라미터	설 명	초기값
-n	n 모드에서 리드 앞의 l 길이에서 최대 불일치 개수	2
-l	-n 모드에서 불일치를 찾는 리드 앞 부분의 길이	28
-e	-n 모드에서 불일치 부분의 퀄리티 값의 합의 최대치	70
-v	-v 모드에서 리드 전체에서 최대 불일치 개수	-

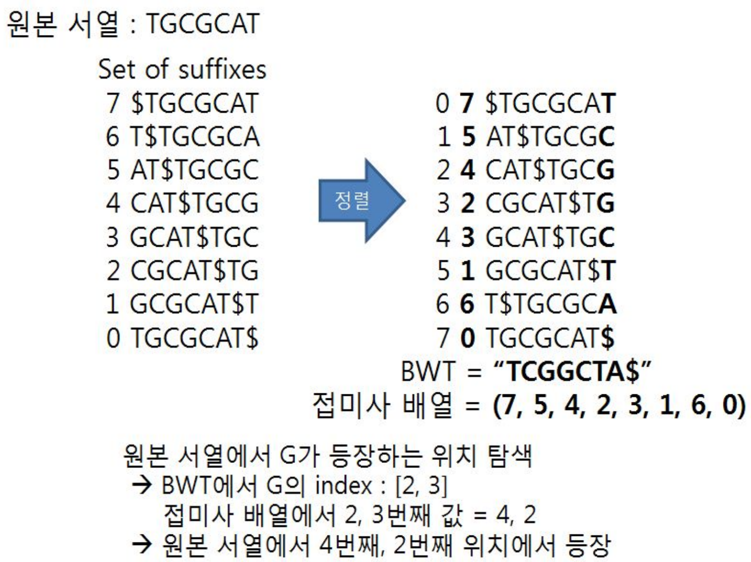
BWA는 BWT와 접미사 배열을 이용하여 정렬을 수행하는 알고리즘이다. BWA는 그림 6에서 보는 바와 같이 각 접미사의 시작 위치를 저장하는 접미사 배열(suffix array)와 BWT를 저장한다. BWA에서 어떤 쿼리가 존재하는 구간을 찾는 것은 접미사 배열의 구간(SA 구간)을 찾는 것과 같은 문제에 해당한다. 예를 들어, 그림 6에서 뉴클레오티드 G가 존재하는 구간은 BWT에서 G가 존재하는 위치인 [2, 3]이다. 이 SA 구간을 알고 있다면 접미사 배열

을 이용하여 원본 시퀀스에서 G가 존재하는 위치인 (4, 2)를 쉽게 얻을 수 있다. 하나의 뉴클레오티드로부터 시작하여 쿼리 전체의 일치 정합을 수행하는 것은 Ferragina와 Manzini의 백워드 탐색(backward search)[9] 방법을 이용한다. 이것은 접두사 트리(prefix trei)의 top-down 순회를 모사하는 방법으로서, 근본적으로 Bowtie의 일치 정합과 동일한 효과를 갖는다.

BWA 역시 Bowtie와 마찬가지로 불일치 정합을 위한 백트래킹을 수행한다. BWA의 백트래킹은 쿼리 서열의 모든 위치에서 다음과 같은 경로를 모두 탐색하는 재귀 알고리즘 형태로 수행한다. 1) 현 위치가 그대로 사용되는 경우 2) 현 위치에서 원본 서열에 갭이 있는 경우, 3) 현 위치에서 쿼리 서열에 갭이 있는 경우, 4) 쿼리 서열의 현재 위치가 다른 문자로 치환되는 경우.

모든 불일치 정합을 탐색하기 위해서는, 쿼리 서열의 모든 위치에서 이와 같은 4가지 경우의 수를 모두 시도해 보아야 하며, 따라서 BWA가 탐색해야 하는 공간의 크기가 매우 커지게 된다. 이를 해결하기 위해 BWA는 백트래킹을 수행하기에 앞서 쿼리 서열의 모든 접두사에 대하여 불일치의 최소 개수를 미리 계산한다. 이 과정은 BWT를 이용한 일치 정합 과정을 응용하여 빠른 속도로 처리된다. 차후 리드의 불일치 정합을 탐색할 때, 현재 위치까지 나타난 불일치의 개수와 앞서 계산한 앞으로 남은 불일치의 최소값을 참고하여 더 이상의 탐색을 진행할지 여부를 결정한다. 이를 통해 불일치 정합을 위한 탐색 공간을 크게 줄일 수 있다.

이론적으로 BWA는 가능한 모든 k-불일치를 찾을 수 있으며, 갭 얼라인먼트 역시 지원한다. 그러나 실질적인 성능 문제로 인하여, 쿼리 서열의 일부 영역(시드)에 대해서만 k개의 불일치를 탐색하는 형태로 구현되었다. BWA의 성능과 관련된 주요 파라미터는 표 5와 같다.



[그림 6] BWA에서 사용하는 BWT과 접미사 배열의 구축

[표 5] BWA의 성능 관련 파라미터

파라미터	설 명	초기값
-l	k개의 불일치를 탐색하는 시드의 길이	32
-k	시드 내에서의 최대 불일치 개수	2

SOAP2[10]는 빠른 맵핑 속도에 중점을 둔 BWT 기반 정렬 알고리즘이다. 이것은 기본적으로 BWA와 유사한 맵핑을 수행하지만 최대 2개의 불일치만을 허용한다는 점이 다르다. 먼저 쿼리 서열의 일치 정합을 시도한 뒤, 이것이 실패하면 1개의 불일치를 허용하는 정합을 시도한다. 이것 역시 실패하면 2개의 불일치를 허용하는 정합을 수행한다. 1개의 불일치를 허용하는 탐색은 정렬 과정에서 더 이상 SA구간을 탐색할 수 없는 위치에 도달하면, 그 위치의 문자를 다른 문자로 치환하여 보는 것이다. 2개의 불일치를 허용하는 탐색은 이 치환을 2번까지 시도한다. 여기까지 실패하면 옵션에 따라 스미스-워터맨 알고리즘을 이용하여 정렬을 수행할 수 있다.

SOAP2는 구현 목적에 맞게 가장 빠른 속도를 갖지만, 허용하는 불일치의 폭이 제한적이며, 쿼리 서열에 에러나 변이가 많을 경우 다수의 스미스-워터맨 알고리즘을 수행함에 따라 처리 속도가 저하될 가능성이 있다.

지금까지 서술한 서열 정렬 알고리즘들의 특징을 간단히 요약하면 표 6과 같다.

3. 시뮬레이션 데이터를 이용한 성능 비교 실험

3.1. 실험 환경 및 방법

지금까지 서술한 정렬 알고리즘들의 간략한 성능 비교를 위해, 시뮬레이션 데이터를 이용한 성능 비교를 수행하였다. 시뮬레이션는 Samtool[11]의 wgsim을 이용하여 생성하였으며, single end 리드를 생성하여 사용하였다. 실험 데이터에 대한 상세한 설명은 표 2와 같다. 데이터셋 1은 wgsim에서 제공하는 기본 설정으로 생성한 데이터이며, 데이터셋 2는 데이터셋 1에 비해 상대적으로 indel의 비율이 낮고 시퀀싱 에러와 단일 치환 변이가 많은 데이터이다. 데이터셋 2와 같이 변이가 높은 데이터셋을 생성한 것은, 서열 정렬 알고리즘들의 불일치 정합

[표 6] 차세대 시퀀싱 리드를 대상으로 하는 정렬 알고리즘의 특징

구 분	알고리즘	특 징
해시 기반	MAQ	여러 개의 템플릿 쌍을 이용하여 리드를 해싱한 뒤 참조 서열을 정렬.
	BFAST	접미사 배열과 간격 시드를 이용하여 불일치 정합 수행
	Stampy	레퍼런스를 k-mer로 해싱하고, 리드에서 불일치의 허용 범위 내에서 생성 가능한 k-mer를 모두 생성한 뒤 맵핑
BWT 기반	Bowtie	BW 행렬와 LF mapping 법칙을 이용하는 알고리즘. 백트래킹을 통해 치환 형태의 에러에 대응한다.
	BWA	접미사 배열와 BWT를 이용하여 SA 구간을 계산하여 정렬을 수행. 이론적으로 모든 형태의 불일치에 대응 가능
	SOAP2	최대 2개의 불일치만 허용하는 대신 빠른 맵핑 수행

[표 7] 테스트 데이터의 상세 설명

참조 서열	19번 염색체 (총 길이 60M bp)	
리드의 길이 및 개수	60bp, 1000만 개 (총 길이 600M bp. X10 커버리지)	
리드 데이터셋	데이터셋 1	데이터셋 2
wgsim 옵션	에러율 = 0.02 단일 치환 비율 = 0.001 indel의 비율 = 0.1	에러율 = 0.05 단일 치환 비율 = 0.05 indel의 비율 = 0.05

성능을 비교하기 위해서이다.

실험은 4-core, 8GB 메모리를 갖는 리눅스 머신에서 수행하였다. 따라서 멀티스레딩을 지원하는 알고리즘은 4개의 스레드를 사용하도록 하였으며, 멀티스레딩을 지원하지 않는 알고리즘은 전체 데이터를 4개의 조각으로 나눈 뒤 각각을 별도의 프로세스로 실행하도록 하였다.

성능 비교 실험은 각 정렬 알고리즘의 디폴트 파라미터를 기준으로 하여, 성능에 영향을 미치는 다른 파라미터를 조절하여 가면서 실험을 수행하였다. 전체 리드의 수를 N, 참조 서열에 맵핑된 리드의 수를 M, 원래 생성된 위치에 정확하게 맵핑된 리드의 수를 C 라 할 때, 알고리즘의 맵핑률은 M/N 을 의미하며, 정확도는 C/M 을 가리킨다. 최종적으로 각 알고리즘에서 종합적으로 가장 우수한 결과를 나타낸 파라미터를 이용하여 알고리즘 간 성능 비교를 수행하였다.

3.2. 실험 결과

3.2.1. MAQ의 성능 분석

MAQ의 주요 파라미터 n과 e를 조절했을 때의 성능 변화는 아래 표 8, 9와 같다. 파라미터 m은 본 논문의 실험 데이터에서 주목할 만한 성능 변화를 이끌어내지 못했다.

[표 8] 파라미터 n에 따른 MAQ의 성능 변화

데이터셋	파라미터	맵핑률 (%)	정확도 (%)	시간 (분)
1	디폴트(n=2)	90.42	97.07	64
	n = 1	86.34	96.99	20
	n = 3	90.76	97.07	151
2	디폴트(n=2)	86.11	96.01	30
	n = 1	69.16	94.45	12
	n = 3	91.23	96.32	94

[표 9] 파라미터 e에 따른 MAQ의 성능 변화

데이터셋	파라미터	맵핑률 (%)	정확도 (%)	시간 (분)
1	디폴트 (e=70)	90.42	97.07	64
	e = 40	81.20	98.31	47
	e = 100	93.14	97.02	48
	e = 130	93.24	96.99	49
2	디폴트 (e=70)	86.11	96.01	30
	e = 40	71.09	96.61	29
	e = 100	87.23	95.45	33
	e = 130	87.63	95.15	35

3.2.2. BFAST의 성능 분석

BFAST의 디폴트 파라미터로는 인간 유전체를 위한 기본 마스크 셋으로 지정된 4개의 마스크를 사용하였으며, 그 중 주 인덱스에 해당하는 마스크 셋만을 이용하여 속도 위주의 정렬을 실험해 보았다.

[표 10] BFAST의 성능 실험 결과

데이터셋	파라미터	맵핑률 (%)	정확도 (%)	시간 (분)
1	디폴트 (마스크 4개)	90.89	95.34	92
	마스크 1개	89.43	95.58	77
2	디폴트 (마스크 4개)	96.29	87.58	89
	마스크 1개	93.11	87.44	72

3.2.3. Stampy의 성능 분석

Stampy는 본 논문의 실험 데이터 상에서 파라미터로 인한 성능 변화가 거의 없었다.

[표 11] Stampy의 성능 실험 결과

데이터셋	파라미터	맵핑률 (%)	정확도 (%)	시간 (분)
1	디폴트	94.02	95.79	49
2	디폴트	99.64	93.56	58

3.2.4. Bowtie의 성능 분석

[표 12] 파라미터 n에 따른 Bowtie의 성능 변화

데이터셋	파라미터	맵핑률 (%)	정확도 (%)	시간 (분)
1	디폴트 (n=2)	89.87	95.20	5
	n = 1	82.54	95.31	3
	n = 3	88.09	95.14	7
2	디폴트 (n=2)	78.44	88.81	10
	n = 1	53.35	88.94	6
	n = 3	77.18	87.65	9

[표 13] 파라미터 e에 따른 Bowtie의 성능 변화

데이터셋	파라미터	맵핑률 (%)	정확도 (%)	시간 (분)
1	디폴트(e=70)	89.87	95.20	5
	e = 40	82.16	96.41	6
	e = 100	92.18	92.49	7
2	디폴트(e=70)	78.44	88.81	10
	e = 40	67.38	93.96	9
	e = 100	79.31	85.40	7

[표 14] 파라미터 l에 따른 Bowtie의 성능 변화

데이터셋	파라미터	맵핑률 (%)	정확도 (%)	시간 (분)
1	디폴트(l=28)	89.87	95.20	5
	l = 24	89.91	95.15	7
	l = 32	89.48	95.25	5
2	디폴트(l=28)	78.44	88.81	10
	l = 24	82.52	88.23	9
	l = 32	72.79	89.46	7

[표 15] 파라미터 v에 따른 Bowtie의 성능 변화

데이터셋	파라미터	맵핑률 (%)	정확도 (%)	시간 (분)
1	v = 1	61.11	95.14	5
	v = 2	82.20	96.60	4
	v = 3	90.76	97.07	11
2	v = 1	14.19	97.04	2
	v = 2	33.54	96.51	7
	v = 3	55.70	96.77	25

3.2.5. BWA의 성능 분석

[표 16] 파라미터 k에 따른 BWA의 성능 변화

데이터셋	파라미터	맵핑률 (%)	정확도 (%)	시간 (분)
1	디폴트 (k=2)	89.83	97.07	16
	k = 1	80.75	96.93	5
	k = 3	91.00	97.08	15
2	디폴트 (k=2)	53.18	96.74	17
	k = 1	38.95	96.46	5
	k = 3	56.20	96.74	15

[표 17] 파라미터 l에 따른 BWA의 성능 변화

데이터셋	파라미터	맵핑률 (%)	정확도 (%)	시간 (분)
1	디폴트 (l = 32)	89.83	97.07	16
	l = 24	90.55	97.06	14
	l = 40	88.64	97.06	13
2	디폴트 (l = 32)	53.18	96.74	17
	l = 24	55.02	96.75	13
	l = 40	50.05	96.70	10

3.2.6. SOAP2의 성능 분석

SOAP2는 파라미터에 따른 성능 변화가 거의 없었기 때문에, 디폴트 파라미터의 실험 결과만을 기록하였다.

[표 18] SOAP2의 성능 실험 결과

데이터셋	파라미터	맵핑률 (%)	정확도 (%)	시간 (분)
1	디폴트	82.19	97.10	7
2	디폴트	33.54	96.87	8

3.2.7. 6개 알고리즘의 성능 비교 결과

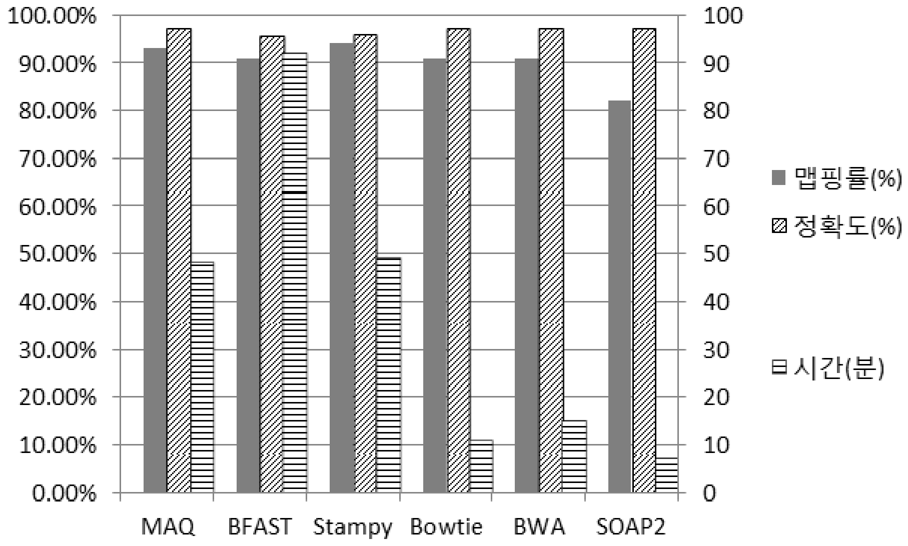
알고리즘 전체의 성능 비교 실험 결과는 아래 그림 7, 8과 같다. 각 알고리즘이 보인 최고의 성능을 이용하여 성능을 비교하였다. 그림 7은 데이터셋 1에서의 실험 결과를, 그림 8은 데이터셋 2에서의 실험 결과를 나타낸다.

3.3. 실험 결과 분석

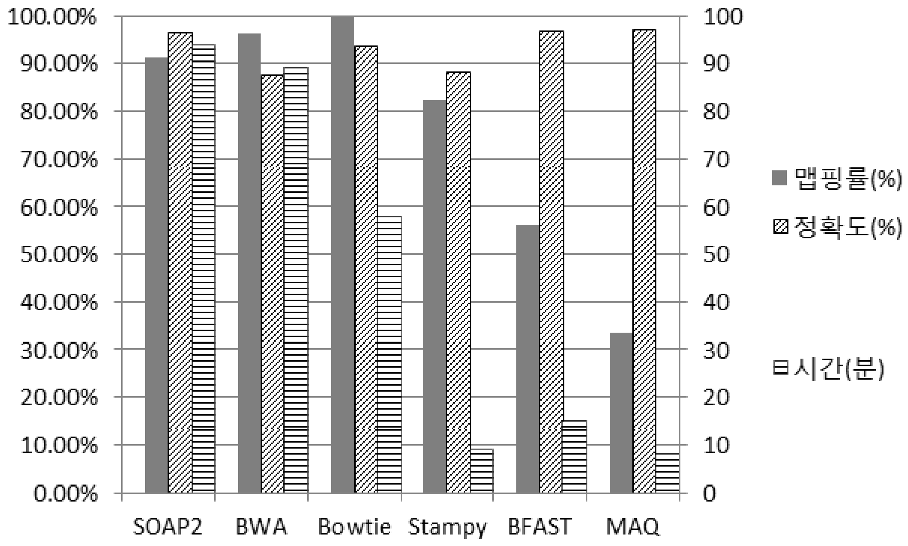
그림 7과 8을 살펴보면, 두 가지의 데이터셋에서 모두 Stampy가 가장 높은 맵핑률을 나타냈으며 MAQ, Bowtie와 BWA가 그 다음 수준의 높은 맵핑률을 보였다. SOAP2의 맵핑률은 상대적으로 낮은 편이었다. 정확도 측면에서는 데이터셋 2에서 Bowtie와 BFAST의 정확도가 다소 낮게 나타난 것을 제외하면 대부분의 알고리즘이 비슷한 수준의 높은(약 95%) 성능을 나타냈다.

실험 결과를 살펴보면 데이터셋 2에서 BWT 기반 알고리즘의 맵핑률이 매우 낮게 나타난 것을 알 수 있다. 데이터셋 2는 데이터셋 1에 비해 단일 치환의 비율을 크게 늘린 데이터다. MAQ나 Stampy와 같은 알고리즘은 많은 수의 치환으로 인해 리드의 일부 영역이 잘 해싱되지 않더라도 다른 영역의 해싱 값에 근거해 정렬을 수행할 수 있다. 반면 BWT 알고리즘은 파라미터로 정해진 횟수 이상의 치환이 발생하면 정렬 자체가 성립하기 어렵다. 그렇지만 데이터셋 1과 같이 보다 일반적인 경우라면 BWT 기반 알고리즘 역시 해시 기반 알고리즘에 뒤떨어지지 않는 성능을 보일 수 있다. 특히 정확도 부분에서 모두 높은 성능을 나타냈다는 것이 주목할 만한 결과이다. 다음으로 알고리즘의 처리량을 살펴보면, BWT에 기반한 세 알고리즘이 해시에 기반한 다른 세 알고리즘에 비해 약 3~7배 빠른 실행 속도를 보이고 있음을 알 수 있다.

위 실험 결과와 NGS 기술의 특징을 종합해 보았을 때, 데이터셋 1과 같은 환경에서는 BWT 기



[그림 7] 데이터셋 1에서 알고리즘의 성능 비교



[그림 8] 데이터셋 2에서 알고리즘의 성능 비교

반 알고리즘이 더 우수한 것으로 판단할 수 있다. 비록 BWT 기반 알고리즘들이 Stampy에 비해 다소 낮은 맵핑률을 보였지만, 수 배 빠른 속도로 정렬 작업을 수행한다면, 결국 같은 시간에 훨씬 많은 수의 리드를 정렬할 수 있게 된다. 맵핑한 리드의 정확도는 해시와 BWT가 비슷한 수준이었기 때문에 결국 BWT 기반 알고리즘이 훨씬 더 많은 리드를 정확하게 참조 서열에 정렬하게 될 것이

다. 반대로 데이터셋 2와 같이 단일 치환이 많은 유전체를 사용하는 프로젝트라면 BWT 기반 알고리즘보다는 해시 기반 알고리즘을 선택하는 것이 더 유리할 것이다.

4. 결론

NGS 기술로 생산된 짧은 리드를 성공적으로 정렬하기 위해서는 가능한 한 높은 정확도를 유지하면서 높은 처리량을 갖추는 것이 중요하다. 이전의 시퀀싱 기술과는 달리, 생산한 모든 리드를 맵핑할 수 없더라도 높은 커버리지로 그것을 보충할 수 있기 때문이다.

NGS 기술의 짧은 리드를 대상으로 하는 정렬 알고리즘은 크게 해시를 이용한 알고리즘과 BWT를 이용한 알고리즘으로 구분할 수 있다. 해시를 기반으로 하는 알고리즘은 이론적으로 O(1)의 복잡도를 기대할 수 있으나, 염기 서열의 반복성과 해시 테이블의 높은 메모리 소요량으로 인하여 이론상의 성능을 발휘하도록 구현하기가 어렵다. BWT를 기반으로 하는 알고리즘은 O(n)의 복잡도로 일치 정합을 수행할 수 있으나, 불일치 정합을 수행하기 위해서는 탐색의 복잡도가 급증하며, 이로 인해 여러 가지 휴리스틱을 적용해야 한다.

시뮬레이션 데이터를 이용한 실험 결과에서, 일반적인 환경에서는 BWT에 기반한 알고리즘이 해시에 기반한 알고리즘에 비해 전반적으로 높은 처리 속도를 나타내면서도 동등한 수준의 정확도를 나타냈다. 반면 단일 치환의 비율이 매우 높은 환경에서는 BWT 기반 알고리즘의 성능이 크게 저하되는 경향을 나타냈다.

서열 정렬 알고리즘을 이용하는 유전체 분석 연구를 수행할 때에는 위와 같은 정렬 알고리즘의 특징과 유전체의 특징을 이해하고, 그에 맞는 알고리즘을 선택하는 것이 중요하다. 분석하고자 하는 유전체에 변이가 매우 많을 것으로 추정된다면 해시 기반 알고리즘을 선택하는 것이 유리할 것이다. 반면 변이 수준이 높지 않고 처리해야 할 리드가 많다면 BWT 기반 알고리즘을 선택하는 것이 유리할 것이다. 추후 개선된 성능의 염기 서열 정렬 알고리즘을 개발하려고 한다면, 본 논문에서 제시한 각 알고리즘의 단점을 개선할 필요가 있다. 해시 기반 알고리즘의 경우 실행 속도를 개선하여 처리량을 개선할 필요가 있으며, BWT 기반 알고리즘의 경우 변이가 많은 상황에 대한 개선이 필요하다.

5. 참고 문헌

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman, “Basic local alignment search tool,” Journal of Molecular Biology, vol. 215, pp. 403−410, Oct. 1990.

[2] T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequences," Journal of Molecular Biology vol. 147, pp. 195-197, 1981.

[3] M. Burrows and D.J. Wheeler, “A Block Sorting Lossless Data Compression Algorithm,” Technical Report 124 Palo Alto, CA: Digital Equipment Corporation, May 1994.

[4] H. Li, J. Ruan, and R. Durbin, “Mapping short DNA sequencing reads and calling

variants using mapping quality scores," Genome Research, vol. 18, pp. 1851-1858, Aug. 2008
[5] N. Homer, B. Merriman, and S.F. Nelson, "BFAST: An Alignment Tool for Large Scale Genome Resequencing," PLoS one, vol. 4, issue 11, e7767, Nov. 2009
[6] G. Lunter and M. Goodson, "Stampy: A statistical algorithm for sensitive and fast mapping of Illumina sequence reads," Genome Research, vol. 21, pp. 936-939, Oct. 2011
[7] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows-Wheeler transform," Bioinformatics, vol. 25, no. 14, pp. 1754-1760, May 2009
[8] B. Langmead, C. Trapnell, M. Pop, and S.L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," Genome Biology, vol. 10, issue 3, Article R25, Mar. 2009
[9] R. Li et al, "SOAP2: an improved ultrafast tool for short read alignment," Bioinformatics Application note, vol. 25, no. 15, pp. 1966-1967, June 2009
[10] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," Proc. 41st Symposium on Foundations of Computer Science (FOCS' 00), pp. 390-398, Nov. 2000
[11] H. Li et al, "The Sequence alignment/map (SAM) format and SAMtools," Bioinformatics, vol. 25, no. 16, pp. 2078-9, June 2009

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 치 현

2007년 홍익대학교 컴퓨터공학과 졸업(학사)
2009년 연세대학교 컴퓨터과학과 졸업(석사)
2009년 - 현재 연세대학교 컴퓨터과학과 박사과정
관심분야 : 바이오인포매틱스, 데이터 마이닝, 데이터베이스 시스템

안 재 균

2006년 연세대학교 컴퓨터과학과 졸업(학사)
200년 연세대학교 컴퓨터과학과 졸업(석사)
2009년 - 현재 연세대학교 컴퓨터과학과 박사과정
관심분야 : 바이오인포매틱스, 데이터 마이닝, 데이터베이스 시스템

박 민 서

2009년 University of Massachusetts, Computer Science 졸업(박사)

2009 – 현재 Samsung SDS CSP 연구소 Bioinformatics Lab 연구원

관심분야 : 바이오인포매틱스, 알고리즘

김 판 규

2006년 부산대학교 전자계산학과 졸업 (박사)

2006년 – 현재 Samsung SDS CSP 연구소 Bioinformatics Lab 연구원

관심분야 : 바이오인포매틱스, 알고리즘

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, 데이터마이닝, 바이오인포매틱스, 적응적 저장장치 시스템, 플래쉬메모리 인덱스, SSD