<서울대학교 자연과학대학 생명과학부 학사졸업논문>

Petasearch: Fast, approximate comparison of huge sequence datasets

(페타탐색: 방대한 서열 데이터셋에 대해 빠른 유사성 검색)

이 논문을 이학사 학위논문으로 제출함

2022년 8월

구분: 실험

논문심사 대상자 소속: 자연과학대학 생명과학부

학번: 2017-17232

성명: Minghang Li (인)

논문지도교수: Martin Steinegger (인)

연구윤리 준수 서약서

본인 (Minghang Li)은 서울대학교 연구자로 연구를 진행함에 있어 다음 사항을 준수할 것을 서약합니다.

1. 서울대학교 연구윤리 관련 규정 및 지침과 국가 법령 및 정부 지침 그리고 일반적으로 학계에서 인정되는 연구윤리 기준을 준수하여 서울대학교 내에서 연구를 수행할 때 위조 변조 표절 등 학문적 진실성을 훼손하는 연구부정행위 또는 연구부적절행위를 하지 않겠습니다.

2. 인간, 동물 등 연구대상에 대한 국내외 윤리 기준을 준수하도록 하겠습니다.

3. 연구 진행 중 이해상충이 발생할 경우 이를 공개하도록 하겠습니다.

4. 정부 및 본교 지침에 따라 연구노트를 작성하며, 연구 데이터에 대한 관리를 철저히 하도록 하겠습니다.

2022년 5월 31일

서약자 소속(학과명): 자연과학대학 생명과학부

이름: Minghang Li (서명)

Abstract

The Sequence Read Archive currently holds over 60 petabases and representing a treasure trove for medicine and biotechnology. Bloom-filter and sketching based approaches were proposed to accelerate searches, however they offer only limited sensitivity. We developed Petasearch to enable fast and sensitive searching through huge protein databases. Its algorithm contains three stages: (1) We pre-process the database sequences to extract k-mers, sort and store them in a highly compressed k-mer index. (2) We extract query k-mers, add similar k-mers and find matches between query and database k-mers. To maximize throughput, we exploit the caching and prefetch infrastructure of modern CPUs, advanced Linux IO techniques, and the enormous read bandwidth of NVMe-SSDs. (3) We compute SIMD-accelerated banded Smith-Waterman alignments between sequences of high-scoring k-mer matches. With such design, Petasearch is proved to have great efficiency: it is up to 190 times faster than state-of-the-art algorithms on a 9.3TB benchmark. At much accelerated speeds, Petasearch matches state-of-the-art algorithms on sensitivity down to sequence identities of 60%. On a SCOP25 benchmark we showed that Petasearch' s profile search detects sequence homology down to 40% sequence identity. We also showed that Petasearch can be applied in finding novel Cas family proteins and discovering new RNA-dependent RNA polymerase (RdRP) homologs. In conclusion, Petasearch is a tool with huge potential. It will enable fast querying of current and upcoming databases and bring bioinformatic researches to a larger scale.

Keywords: Sequence analysis, Sequence search, Protein databases, Proteins, Protein profiles, Large-scale annotation

Student ID: 2017-17232

Contents

1. Introduction

1.1 Sequence Databases

Next generation sequencing (NGS) technologies have revolutionized the way we collect and analyze biological data. Thanks to NGS, the cost of sequencing has dropped drastically and continued to decrease with more new technologies developed. Accompanying this change is the explosive growth of the amount of sequencing data and the size of sequence databases. The Sequence Read Archive (SRA) is one of the most popular and widely used sequence databases that store both private and public sequence reads and provide access in various foramts including the commonly used `FASTQ` file format. Its size has grown exponentially since 2008 and currently reached more than 60 petabytes large. The growth in size of Sequence Read Archive is visualized in Figure 1.1.

1.2 State-of-the-art Algorithms for Sequence Searches

1.2.1 DIAMOND

1.2.2 MMseqs2

1.2.3 BIGSI

1.3 Prototype of Petasearch Algorithm

1.4 Motivation and Contribution of the Thesis

The search of homologs in large sequence databases requires a fast yet sensitive enough algorithm specially designed for petabyte-scale analysis. The state-of-the-art searching algorithms failed to satisfy this need. The prototype of `Petasearch`, despite its idea proven to be promising, has not reached its peak speed efficiency and is rather heavy in disk consumption. Limited by the current design, its searching sensitivity is also less desirable for homologs with sequence similarity less than 40%. To tackle these problems and make `Petasearch` more available to the public, we revised the design of the core data structures of `Petasearch` and added the profile-search functionality. The main contribution of this thesis is the major improvement of the `Petasearch` algorithm in speed, space consumption and sensitivity.

In chapter 2, we will continue with describing the further development and optimization of `Petasearch`. We will also describe the design of the benchmarks in chapter 2. In chapter 3, we will first show the improvements in efficiency and effectiveness of the forementioned optimizations. Afterwards, we will show a thorough comparison of the performance of the `Petasearch` algorithm with the state-of-the-art algorithms. In chapter 4, we will discuss the potential application of `Petasearch` and show two examples of its usage.

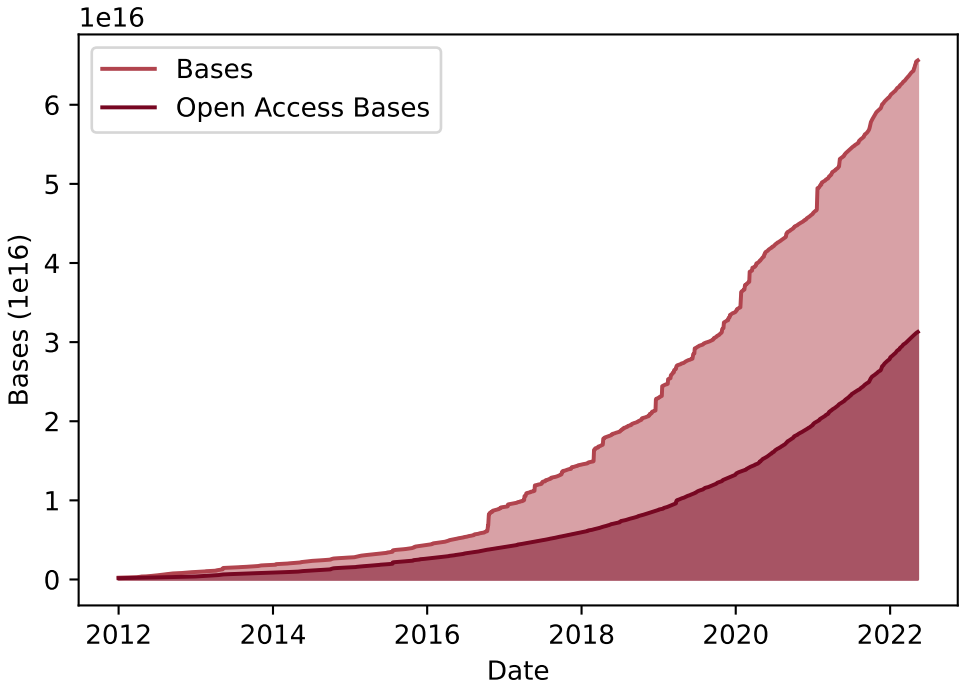

Figure 1.1: The exponential growth of the Sequence Read Archive from 2008 to 2022. The total amount of sequence data (unit in bases) and publicly available data are visualized in pink and dark red respectively.

2. Materials & Methods

2.1 Space Optimization

2.1.1 Diff-index Compression

As is described in chapter 2, the diff-index created in the k-mer extraction step will store multiple `USHRT_MAX` as long as the difference is larger than `USHRT_MAX`. This will make any k-mer difference larger than $4 \times$ `USHRT_MAX` $= 262140$ require a larger space to store than the original `unsigned long` representation. This situation is not uncommon especially when k is large. Also, in the prototypical implementation, the ID of sthe source sequence will also be stored multiple times in the ID table. This redundancy is both unnecessary and troublesome. It will increase the size of `Petasearch` data structures even more than the repeated `USHRT_MAX` since the IDs are stored as `unsigned long` (64-bit integers). Figure 2.1 showed the space consumption of the diff-index created in the k-mer extraction step when $k = 11$. Without optimization, the diff-index (k-mer table) and its corresponding ID table will take up 17 GB of space for a merely 1GB-sized database.

To optimize the size of the diff-index, we devised the bit-squeezing technique to compress the difference between two adjacent k-mers: For any 64-bit k-mer difference, we continuously fetch 15 bits into a write buffer starting from the least significant bits. We stop the retrieval until we encounter a zero chunk (15 bits of zeros). To enable the correct decoding of the diff-index during the next phase, the sign bit of the last element in th write is set to `1` to indicate the end of encoding. Afterwards, we write all the elements in the write buffer to the diff-index. An example encoding process for differecne of value 2039432531946 is shown in Figure 2.2. For ID table, the optimization is simple: we store the ID of the source sequence only once instead of repeatedly. Using the bit-squeezing method, it is possible to obtain a maximum of five chunks, making the final space consumption larger than the size of a `unsigned long` integer. However, such situation only happens when the difference is larger than `1UL << 59` $= 576460752303423488$, which is extremely rare.

While decoding the compressed diff-index in the process of double-index search, we will reverse the bit-squeezing process through repeatedly retrieving 15 bits from the diff-index table until we encounter the chunk with the sign bit set to `1`. The decoded difference value will be add to the current k-mer. Moreover, since we do not store redundant IDs, the ID pointer will not be incremented until the end of k-mer decoding. Algorithm 1 showed the simplified pseudocode for k-mer decoding.

Algorithm 1 Pseudocode for the k-mer decoding process

```
procedure DecodeKmer(currentKmer, currentTargetKmerPtr, currentTargetIDPtr)
    currentDiffIndex ← 0
    while *currentTargetKmerPtr > 0 do          ▷ This means the sign bit is not 1.
        currentDiffIndex ← Get15Bits(*currTargetKmerPtr)
        currentDiffIndex ← currentDiffIndex << 15
        Next(currTargetKmerPtr)
    end while
    currentDiffIndex ← Get15Bits(*currTargetKmerPtr)
    currentKmer ← currentKmer + currentDiffIndex
    Next(currTargetKmerPtr)
    currentTargetIDPtr ← Next(currentTargetIDPtr) return currentKmer
end procedure
```

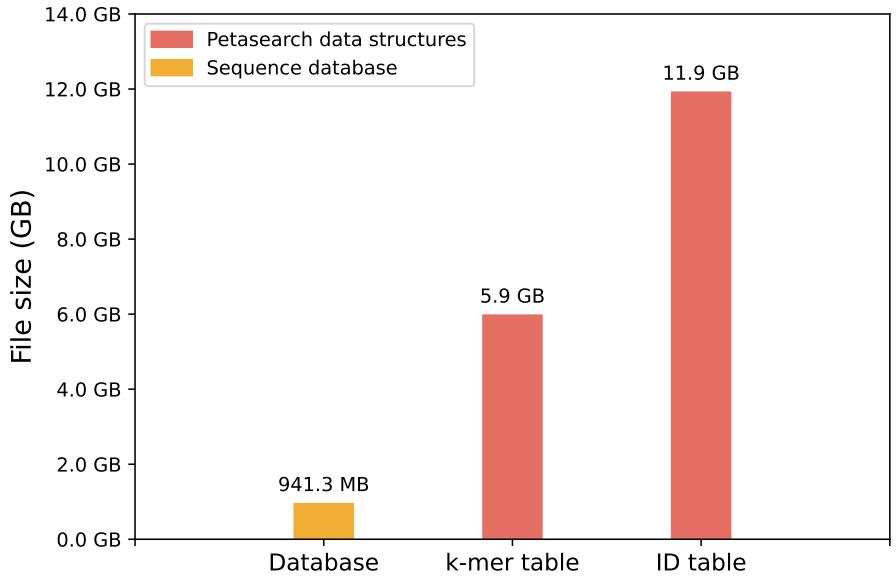

Figure 2.1: Visualizaiton of k-mer table and ID table sizes when $k = 11$. The database is the `UniProtKB/Swiss-Prot` database obtained through `mmseqs databases UniProtKB/Swiss-Prot swissprot tmp` command. Without optimization, the sizes of `petasearch` data structures are 6.46 times and 12.92 times larger than the sequence database.

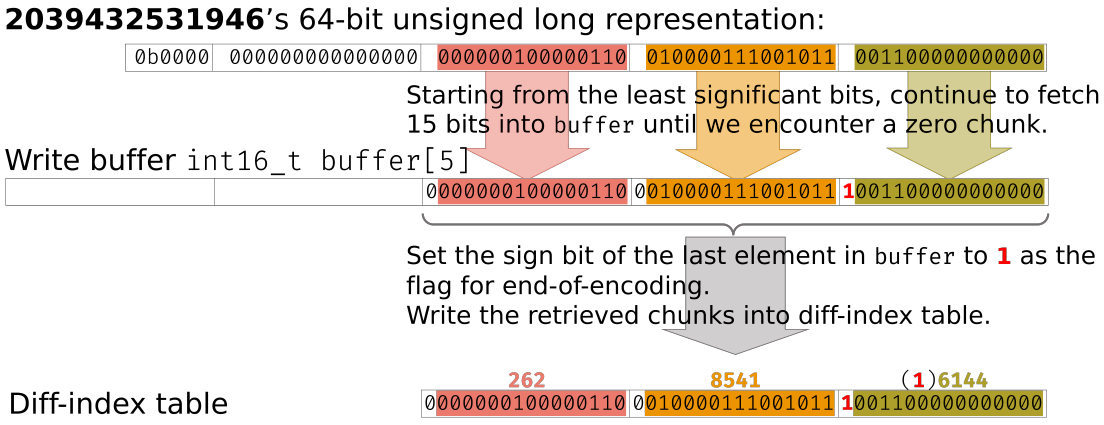

Figure 2.2: The example decoding process of difference index 2039432531946. We first retrieve 15-bit chunks starting from the least significant bits and store them into a write buffer in the reverse order until we encounter a zero chunk. For 2039432531946, its highest non-zero bit is 39, which means that we need three 16-bit `short` to store it.

2.2 Speed Optimization

2.2.1

2.3 Sensitivity Improvement

3. Results

4. Discussion

5. References