Efficient storage of genome data

Florian Markowsky

University of Hamburg
Center for Bioinformatics
Bundesstraße 43
20146 Hamburg, Germany

http://www.zbh.uni-hamburg.de
info@zbh.uni-hamburg.de

Outline

Introduction

State of research: different compression techniques

- Reference-based read compression
- Iterative dictionary construction: COMRAD
- Robust relative compression with random access

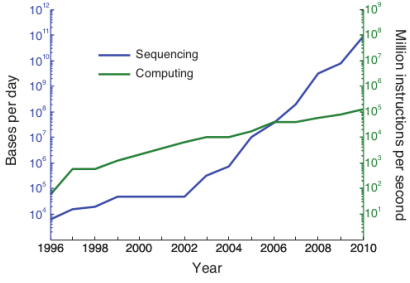
Genome ReSequencing tool

What is to come...

Genome size

Problem:

- Genomes are large: 10^5 to 10^{10} BP
- New sequencing technologies allow fast and cheap sequencing
- Genome data grows faster than processing power (Moore's Law)



Genome size

But:

- Single genome almost incompressible
- Highly similar in same species (human: 1% difference)
- $\rightarrow$ if examining genome collections, only differences have to be stored

Demands:

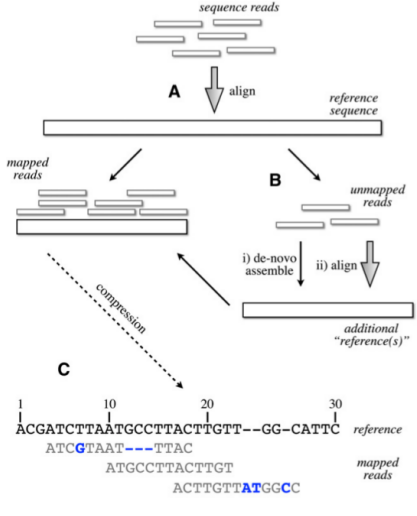
- Reduce file size as much as possible
- Optional:
  - Fast compression
  - Random access/partly decompression

Different Approaches

- Reference based compression (Fritz, Leinonen, Chochrain *et al.* 2011)
- Iterative dictionary for large DNA datasets (Kuruppu *et al.* 2011)
- Robust relative compression with random access (Deorowicz, Grabowski 2011)

Reference-based read compression - Algorithm

- Stores short sequence reads
- Alignment of reads to reference sequence
- Store position of reads, strand, Indel information
- Postitions relatively stored als Golomb-codes



Position	Strand	Substitutions	Insertions	Deletions
4	+	4-G	*none*	5-3
6	+	*none*	*none*	*none*
7	+	*none*	8-AT 4-C	*none*

Reference-based read compression - Discussion

Problem: what to do with unaligned reads?

- Discard: loss of information
- Keep: high storage cost $\rightarrow$ other compression technique?

Compression Results:

	reference based	raw FASTA	bzip2
human genome	0.41 bpb	11.45 bpb	1.84 bpb
bacterial genome	0.19 bpb	12.27 bpb	1.66 bpb

$\rightarrow$ 10- to 30-fold better compression than standard approaches

COMRAD - Algorithm

Dictionary construction by identifying repeated substrings in large DNA datasets

$\rightarrow$ works best on highly redundant datasets

In each iteration:

- Modify frequency-dictionary
  - Count frequences of substrings
- Substitiution
  - Substitute all new dictionary entries for new nonterminals
- result iteration k: Dictionary D_k, Alphabet Σ_k, set S_k of compressed sequences

final compression: Huffman

Input
aabcbcaabcabc

.

Step 1
aa:2 ab:3 bc:4 cb:1 ca:2

.

Step 2
aabcbcaabcabc (*no candidate*)
a**abc**bcaabcabc (*no candidate*)
aa**bcb**caabcabc (*cand cnt bc: 1*)
aabc**bca**abcabc (*cand cnt bc: 2*)
aabcbc**aab**cabc (*no candidate*)
aabcbca**abc**abc (*no candidate*)
aabcbcaa**bca**bc (*cand cnt bc: 3*)
aabcbcaabc**abc** (*no candidate*)
aabcbcaabca**bc** (*cand cnt bc: 4*)

.

Step 3
bc → A
aaAAaaAaA

.

Step 4
aa:2 aA:3 AA:1 Aa:2 bc:1

...

aA → B
aBAaBB

...

aB → C
CACB

COMRAD - Discussion

- Provides random access and single sequence decompression
- Efficient on large dataset with many/long repeats
- Nearly no effect on small datasets with short repeats: disproportionately large codebook

- Influenza genomes 113 MB (1.97 bpb) to 6 MB (0.43 bpb)
  - time for compression: about 0.03h $\rightarrow$ 1.04 MB/s
- Yeast genomes 485.87 MB (2.19 bpb) to 15.29 MB (0.25 bpb)
  - time for compression: about 0.19h $\rightarrow$ 0.71 MB/s
- 4 human genomes 12 066,06 MB (2.18 bpb) to 2 176,06MB (1.44 bpb)
  - time for compression: about 8h $\rightarrow$ 0.42 MB/s

Robust relative compression with random access - Algorithm

- Input sequence is parsed in sequence of matches and literals
- Hash array used to find matches
- Lookahead buffer
- Stores matches as pair of reference offset and match length
  $\rightarrow$ Huffman
- Literals and reference sequence $\rightarrow$ Huffman

Robust relative compression with random access - Discussion

- Allows random access to parts of compressed sequences due to blockwise coding
- 39 yeast genomes: 493.98 MB compressed to 6.88 MB, 33.15 MB/s
- 70 human genomes: 218 961,98 MB compressed to 1 201,15MB, 146.16 MB/s

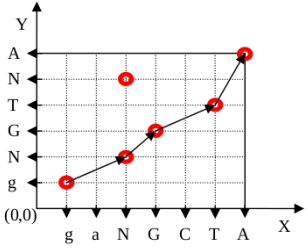
Genome ReSequencing tool

GRS Algorithm computes compressed difference files

- calculate varied sequence percentage δ for each chromosome based on a reference sequence
- if $\delta < 0.03$
  - compute different sequence file
- if $0.03 < \delta < 0.1$
  - cut chromosome into *n* pieces
  - calculate each δ_i
  - find position with minimal $\Sigma\delta_i$
  - compress each piece
- if $\delta > 0.1$: Data not suitable

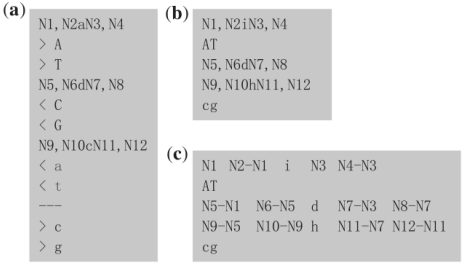
Genome ReSequencing tool

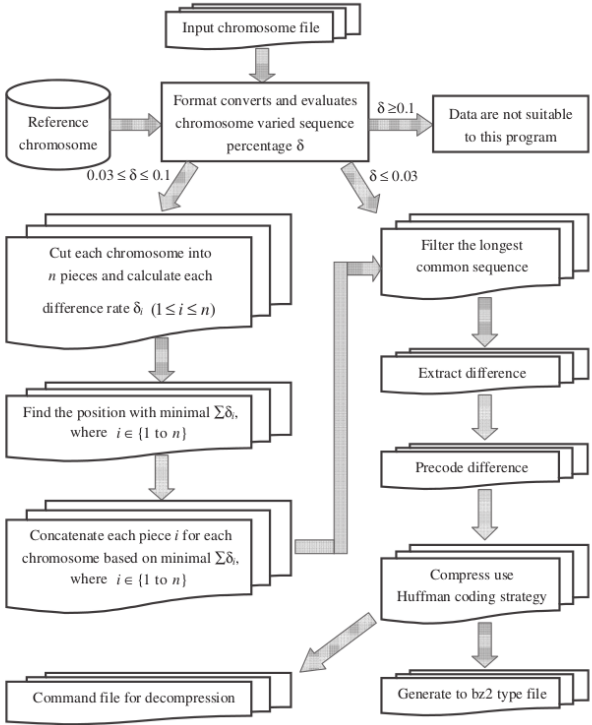
- find longest common local nucleotide sequences
- example: gaNGCTA and gNGTNA



Genome ReSequencing tool

- method to compute difference file similar to UNIX diff program
- process diff file to reduce size
- encode using Huffman




Input chromosome file
Reference chromosome
Format converts and evaluates chromosome varied sequence percentage δ
δ ≥0.1
Data are not suitable to this program
0.03 ≤ δ ≤ 0.1
δ ≤ 0.03
Cut each chromosome into n pieces and calculate each difference rate δi (1 ≤ i ≤ n)
Find the position with minimal ∑δi, where i ∈ {1 to n}
Concatenate each piece i for each chromosome based on minimal ∑δi, where i ∈ {1 to n}
Filter the longest common sequence
Extract difference
Precode difference
Compress use Huffman coding strategy
Generate to bz2 type file
Command file for decompression

Genome ReSequencing tool - Discussion

- Allows encoding without knowledge of reference SNPs map
- Rice genomes: 361 MB compressed to 4.4 MB, 0.26 MB/s
- Human genome: 2 986,8 MB compressed to 18.8 MB, 1.81 MB/s

- No random access

Future development

Presented Algorithms:

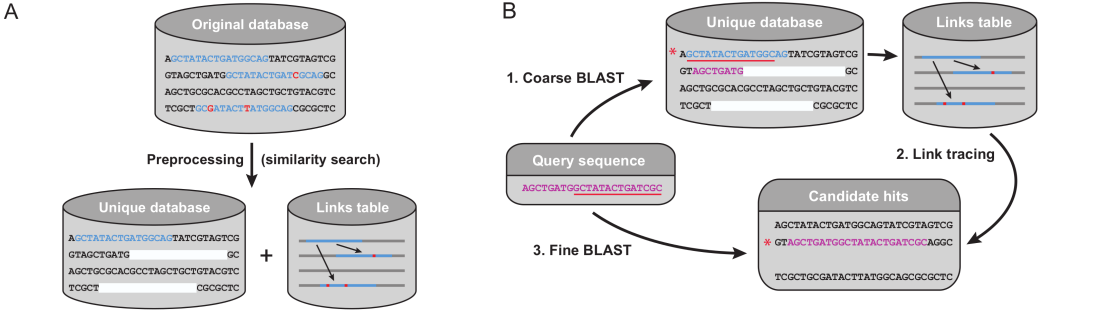
- can compress effectively
- are efficient enough to store many genomes on available disk space

but...

- Decompression necesarry for computation
- Aim: algorithms computing on compressed data
- Next slides: a compressive BLAST algorithm is presented

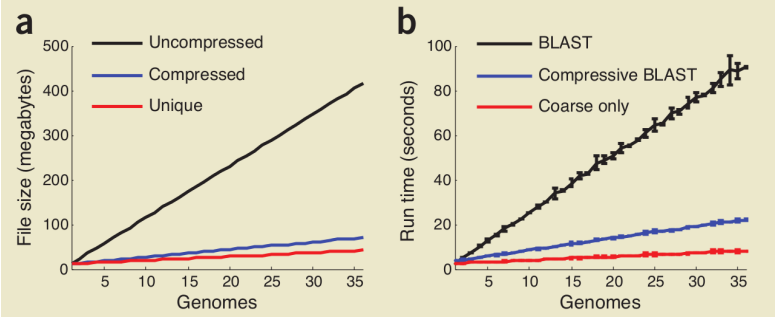
CaBlast - Algorithm

- Preprocession: compression of input data
  - fill unique database with first occurances of substrings
  - store repeats as pointers to subsequence from unique occurrence
- CaBLAST:
  - perform coarse BLAST Search on unique database
    $\rightarrow$follow pointers to repeats of hits
  - perform fine BLAST Search on coarse-hits and decompressed repeats


A
Original database
AGCTATACTGATGGCAGTATCGTAGTCG
GTAGCTGATGGCTATACTGATCGCAGGC
AGCTGCGCACGCCTAGCTGCTGTACGTC
TCGCTGCGATACTTATGGCAGCGCGCTC
Preprocessing
(similarity search)
Unique database
AGCTATACTGATGGCAGTATCGTAGTCG
GTAGCTGATG GC
AGCTGCGCACGCCTAGCTGCTGTACGTC
TCGCT CGCGCTC
+
Links table
B
Unique database
AGCTATACTGATGGCAGTATCGTAGTCG
GTAGCTGATG GC
AGCTGCGCACGCCTAGCTGCTGTACGTC
TCGCT CGCGCTC
Links table
1. Coarse BLAST
Query sequence
AGCTGATGGCTATACTGATCGC
2. Link tracing
Candidate hits
AGCTATACTGATGGCAGTATCGTAGTCG
GTAGCTGATGGCTATACTGATCGCAGGC
TCGCTGCGATACTTATGGCAGCGCGCTC
3. Fine BLAST

CaBlast - Discussion

- improved runtime and space requirements



Bibliography

- M. Fritz, R. Leinonen, G. Cochrane, E. Birney: Efficient storage of high throughput DNA sequencing data using reference-based compression, Genome Research, 2011
- S. Kuruppu, B. Beresford-Smith, T. Conway, J. Zobel: Iterative Dictionary Construction for Compression of Large DNA Datasets, IEEE Transactions on computational Biology and Bioinformatics, 2011
- S. Deorowicz, S. Grabowski: Robust Relative Compression of Genomes with Random access, Bioinformatics, 2011, 27(21):2979-86
- C. Wang, D. Zhang: A novel compression tool for efficient storage of genome resequencing data, Nucleic Acids Research, 2011
- P. Loh, M.Baym, B. Berger: Compressive Genomics, Nat Biotechnol. 2012, 30(7):627-30