Similarity based Approach for Compression of Noisy Data

Takeaki Uno (National Institute of Informatics)

Sep 2013

Compression of Noisy Data

- Recent bigdata era has given huge noisy data to us

- For these data, standard compression techniques do not work well
  ← same strings will be “different”, by noise

- Frequency and repetition are small
  ← Huffman coding, run length coding are not efficient

- However, “similarity” exists

abcdef abcDef abCdef Abcdef AbcDef …

Using Similarity

- Huffman coding represent each word by “ID”, whose length depends on its frequency

- In similarity, each “word/segment” is represented by “address” + “difference”
  - + “address” is the position of the reference string
  - + ”difference” is the difference between the word/segment and the reference word/segment

- We can use a large database, or dictionary, as a reference set

Position of Similarity

- Similarity is on the middle of Huffman/Run length type and context free grammar type

  + Huffman/run length simply use “frequency” of words
  + context free grammar use the “global structure” of the data

  + similarity is a local structure, related to the approximate frequency of the words

- Similarity makes huffman/run length/context grammar more efficient in noisy data

Performance on Genome Sequence

	human chr. 22	human chr. X	mouse chr. 19	bacteria	dicty	dros	
size	8764657	33793259	14535557	14241057	5278628	6976263	
zip deflate 9	100.4%	102.2%	102.8%	104.2%	93.3%	106.1%	
lzma 9	**88.9%**	**86.7%**	**91.1%**	**94.3%**	**82.7%**	101.2%	
bicompress2	89.5%	95.6%	91.8%	97.5%	86.7%	**97.9%**	
FuzzyLZ	90.2%	102.9%	93.4%	106.5%	90.9%	106.4%	

- Since human and mouse have redundant structures, the compression ratio is high, but is up to 90%, roughly

- Bacteria and dicty may have large word bias

Develop a compression method by similarity

Idea: Use Similarity

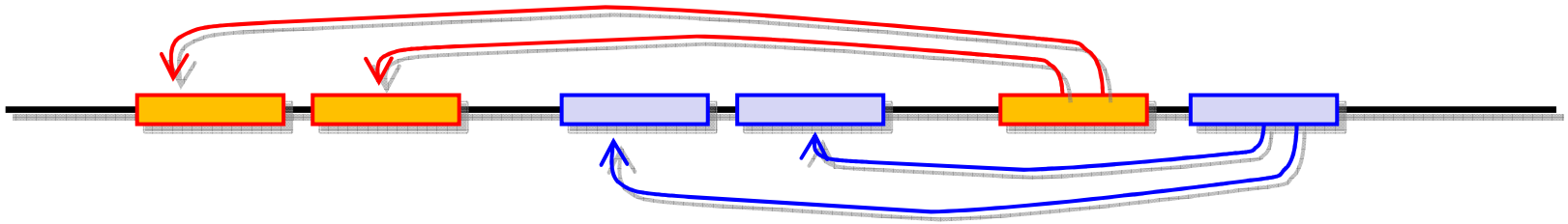
- In genome sequences, there are few same substrings, but are many similar substrings

  ➔ Refer similar string and store the difference

- Use Hamming distance as similarity

  ➔ Edit distance is too much heavy (the representation of difference would be large)

- Use Multi-sorting algorithm for finding pairs of fixed-length similar substrings



Outline of Compression

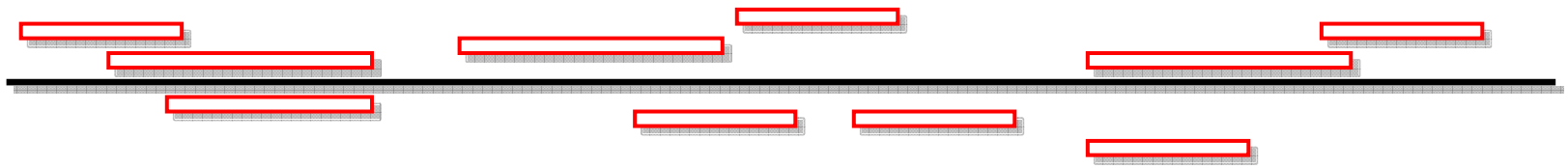
① Find similar substrings (with Hamming distance at most 3)
of length 20, from genome sequence (called **seeds**)

② Extend the similarity by blocks of length 10, in both direction,
until encountering a block with Hamming distance more than 2

③ Select the similar substring to be refered, and how many
preceding/following block we use, for each position
(by dynamic programming)

- ① can be done in short time if the string is not so much repetitive.
- ② can be done in short time
- ③ can be done in short time, by dynamic programming

The Tasks



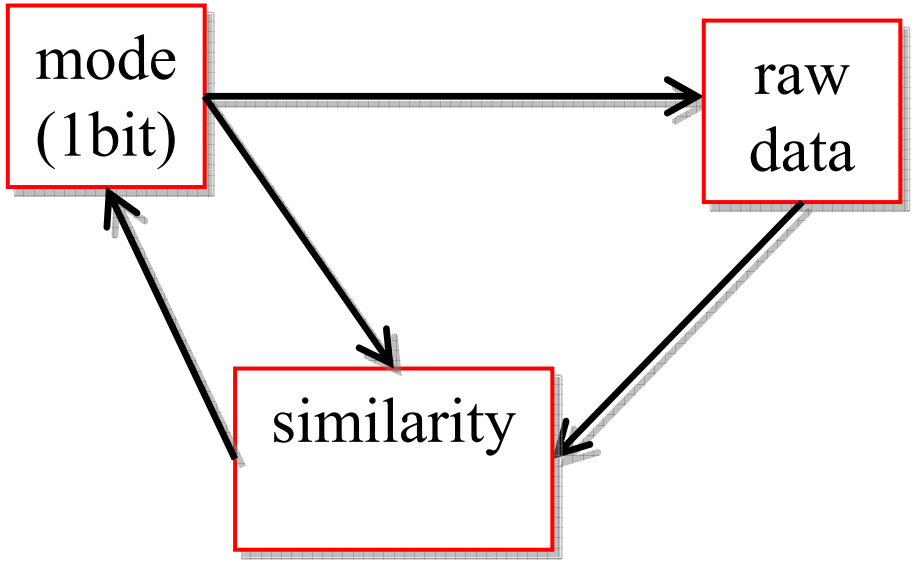
- We have to determine the similarities we use to compress
  (each similarity has different efficiency)
- We also have to determine the positions (blocks) from which the similarities start

- By dynamic programming, for each position, we compute the best possible compression ratio, from left to right

- The compression ratio is computed according to #bits used for similarities (#bits needed to refer a position and represent diff.)

Coding Rule: Outline

- In the code, raw data and similarity alternatively appear
- Raw data is stored by “length of raw data” + raw string (2bits / letter)
  (length is usually short, thus stored by a variable length integer)
- Each similarity is represented by “ref” + “diff” of blocks …



Coding Rule: Block

- A block is represented by a **mark** of 2bits, and **difference**

  mark: **00**,**01**,**10** Hamming distance is 0,1,2, **11** is end-mark

- The number of possible differences is **1**, ${}_{10}C_1 \times 3$ and ${}_{10}C_2 \times 3 \times 3$ which can be coded in **0**, **5**, **9** bits

- A block is represented by 2bits + ??bits, and repeat until endmark

Coding Rule: Seed

- The seed is always used

  (thus, first end-mark of similarity means, seed appear)

- The mark of seed is **00**,**01**,**10**,**11** → Hamming distance is 0,1,2,3
- #differences is **1**, $_{20}C_1 \times 3$, $_{20}C_2 \times 3^2$ and $_{20}C_2 \times 3^3$

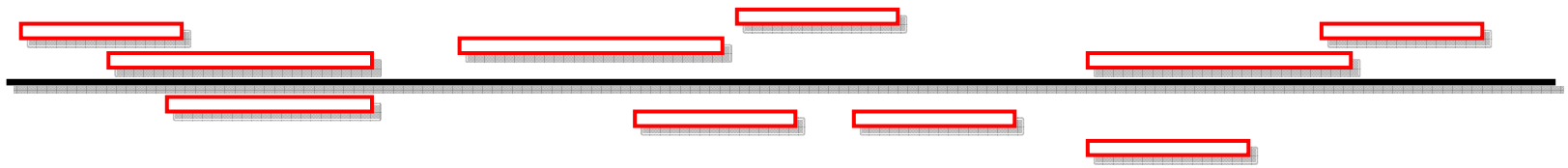
  which can be coded in **0**, **6**, **11**, **17** bits

- After the seed, usual blocks, again, follow

- The next end-mark means the (true) end of the similarity
- The next mode can be raw data or similarity, thus end-mark is

110: end + raw string follows, **111**: end + similarity follows

Dynamic Programming

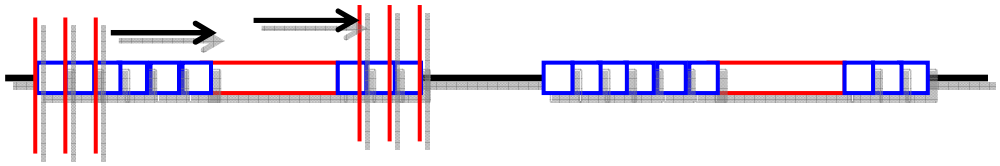


• When the coding rule is determined, we can compute best compression for each position, for both cases that at the position (a) a similarity ends, and (b) raw mode ends

Examine the cases

1. raw mode continues (increase 2bits)
2. similarity ends just before, and raw mode begins here
   (increase 2bits + XX bits for a number)
3. Similarity end here

Cost Computation for Similarity



- For a seed, if the optimal both costs of all preceding position is determined, we can compute from where we should start (that yields optimal compression)

- Once optimal start position is determined, we can compute the compression cost at the end of each following block

- Compute as this for all seeds, starting at the current frontier position

Some Additional Rule

- Genome sequence is chopped into chunks to be compressed
  (each chunk is compressed, with referring a preceding block)
- Algorithm refers the reverse direction of the sequence
  distance to the reference is limited by chunk size
- The cost for representing a number is fixed, for conciseness
  (reference → the size of chunk, length → variable length integer)
- When the sequence is partially very similar, code 00 is frequently used
  In such case, we give code 0 for 00, and (100, 101, 11) for (01, 10, 11)
- Similar pairs are stored with unifying continuous pairs
  (so, maximal continuously similar substrings are stored)

The Performances

	human chr. 22	human chr. X	mouse chr. 19	Bacteria	dicty	dros	
size	8764657	33793259	14535557	14241057	5278628	6976263	
zip deflate 9	100.4%	102.2%	102.8%	104.2%	93.3%	106.1%	
lzma 9	**88.9%**	**86.7%**	**91.1%**	**94.3%**	**82.7%**	101.2%	
biocompress2	89.5%	95.6%	91.8%	97.5%	86.7%	**97.9%**	
FuzzyLZ	90.2%	102.9%	93.4%	106.5%	90.9%	106.4%	
genz 20	86.9(27.2)	86.2(31.4)	90.3(20.9)	98.5(4.4)	91.3(22.7)	98.8(3.1)	
+ lzma 9	**84.8%**	**84.1%**	**88.1%**	**94.3%**	84.3%	**97.4%**	
genz 20 l	87.2[6.3]	86.6[4.9]	90.5[6.5]	98.6[3.7]	91.7[7.2]	98.8[3.7]	
genz 16	89.7[4.4]	91.5[2.2]	93.9[3.5]	99.3[1.2]	93.4[15.3]	99.2[1.1]	
genz Ham2	87.2[3.1]	86.7[3.0]	90.6[4.0]	98.6[4.3]	91.6[2.9]	98.8[3.0]	
genz 24,lk	86.1[0.87]	83.1[0.85]	89.0[1.5]	94.2[1.34]	88.3[6.0]	97.3[1.79]	
24lk+lzma 9	**83.7%**	**81.0%**	**86.4%**	**89.8%**	**80.8%**	**96.2%**	

Advantages / Disadvantages

- For human and mouse, genz is strong (may have much similarity)
- For dros, all are bad, but genz is the best (may have few similarities)
- For bacteria and dicty, lzma is the best
  (may have perfect similarities, and large word bias)
- Computation time for 1MB is
  3 sec. ← zip -9
  1 sec. ← lzma
  40～ sec. ← genz

For Speeding Up

- The bottleneck is “finding similar substring pairs”
  ← there are several approximations

  (1) decrease the size of chunks
  (2) decrease the threshold for Hamming distance
  (3) use interleave positions

- Each accelerates up to (1), decrease to 16, 2x (2), 3x, (3) 6x,
  and loses accuracy 0.5% - 1.0%

- By increasing the chunk size, genz outperforms all the others, but takes long time… (with interleave and limitation of similar pairs, the time becomes short)

Compression on Short Read

- Short read is a fragment of a genome sequence, taken by genome sequencers
  - ← Usually very short, such as 36 letters (Solexa)

- The position of each short read is randomly chosen, thus there are many overlaps
  - ← There are much redundancy

- We approach efficient compression from this redundancy
  - ← represent the short reads by their differences from similar ones

Reference to Genome Sequence

- In principle, each short read can be mapped to some positions of the genome with small errors

  ← can be represented by difference!

- However, the addressing takes much space, such as 32 bits for Homo-sapience

  (36 letters takes 72 bits, so compression ratio must be > 40%)

- We consider another method to avoid heavy addressing

Self-Reference

- Since referring the genome is heavy, we refer the short read itself

- To save the space for addressing, we use the permutation of the ordering, of the short reads

- The idea is, find a minimum spanning tree on the similarity graph

  vertices: short reads

  edges: between similar short reads

  edge weights: distance between the short reads

- We encode the tree in 2n bits, and for each short read, store the difference from its parent

Implementation and Experiments

- **… to be done**

Conclusion

- We proposed the use of similarity for compression

- An example is a genome sequence compression, based on representing a segment by referring similar substring

→ The compression ratio is considerable better

- Next example is a short read compression, based on MST in a similarity graph

Future work:

- Implementation and experiments
- Sequence data with real numbers (ex., trajectory)
- Images and Movies

e