Paper discussion: MUMmer

Dr. Jared Simpson
Ontario Institute for Cancer Research
&
Department of Computer Science
University of Toronto

MUMmer: the problem

- How do we compare two similar genomes?
- We could use edit distance algorithm but real genomes are too large for this to be practical

Edit distance algorithm: O(mn) operations

E. coli genome size: 4×10^6 bp
Comparing two E. coli strains: ~10^{13} ops (maybe…)

Human genome: 3×10^9 bp
Comparing two human genomes: ~10^{19} (nope)

MUMmer: the approach

- Idea: many genomes we want to compare are similar
  - examples:
    - two strains of a bacterial species (>99.99%)
    - human and chimpanzee genome (~99%)
  - when genomes are closely related we expect long exact matches
  - use a fast method to find these exact matches, then “stitch” them together with dynamic programming

Related work

Another system developed to align long sequences is *sim2* (14). This system uses a BLAST-like hashing scheme to identify exact *k*-mer matches, which are extended to maximal-length matches. These maximal matches are then combined into local alignment chains by a dynamic programming step. In contrast, our suffix-tree approach directly finds maximal matches that are unique. These matches can then be easily ordered to form the basis of an alignment that can span even very long mismatch regions between the two input genomes.

MUMmer algorithm 1: suffix tree to find MUMs

- Use a generalized suffix tree to efficiently find *Maximal Unique Matches* between genomes
  - maximal: match cannot be extended any further
  - unique: matched substring only occurs once in each genome

Genome A: `tcgatcGACGATCGCGGCCGTAGATCGAATAACGAGAGAGCATAAcgactta`
Genome B: `gcattaGACGATCGCGGCCGTAGATCGAATAACGAGAGAGCATAAtccagag`

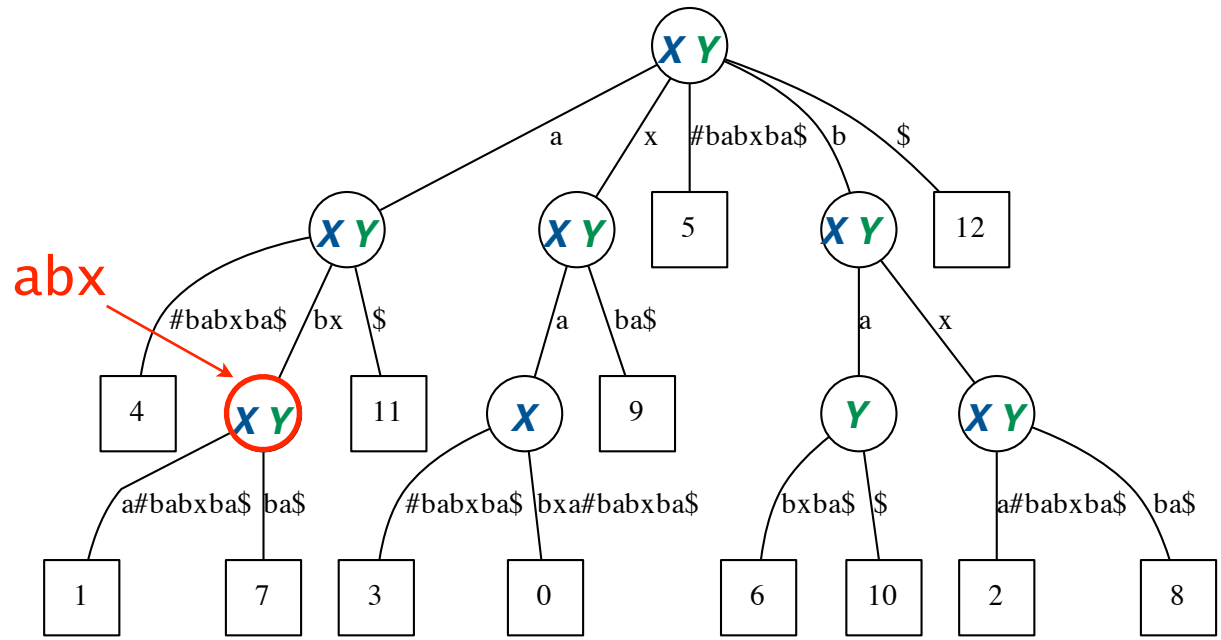
Figure 1. A maximal unique matching subsequence (MUM) of 39 nt (shown in uppercase) shared by Genome *A* and Genome *B*. Any extension of the MUM will result in a mismatch. By definition, an MUM does not occur anywhere else in either genome.

Suffix tree application: find longest common substring

To find the longest common substring (LCS) of *X* and *Y*, make a new string $X\#Y\$$ where $\#$ and $\$$ are both terminal symbols. Build a suffix tree for $X\#Y\$$.

X = xabxa
Y = babxba
***X*#*Y*$** = xabxa#babxba$

Consider leaves:
offsets in [0, 4] are suffixes of ***X***, offsets in [6, 11] are suffixes of ***Y***



Traverse the tree and annotate each node according to whether leaves below it include suffixes of ***X***, ***Y*** or both

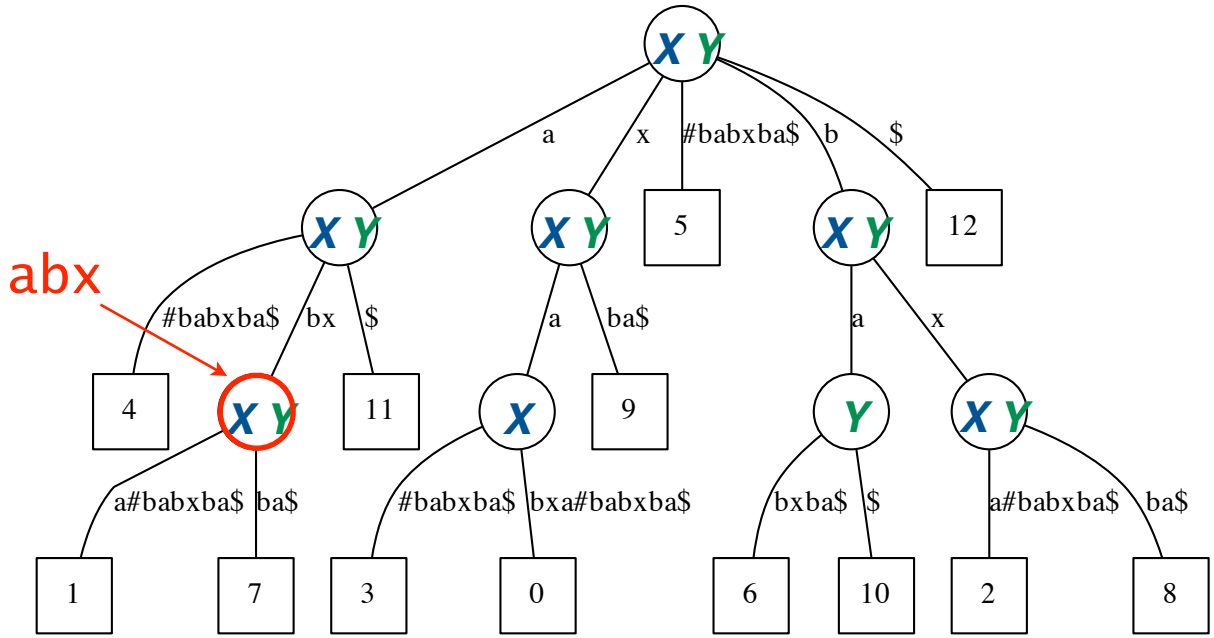
The deepest node annotated with both ***X*** and ***Y*** has LCS as its label.

O(| ***X*** | + | ***Y*** |) time and space.

Finding Maximal unique matches

Uniqueness: only a MUM if it occurs once in both X and Y (one leaf for each string in subtree rooted at node)

Maximal: check whether string can be extended when visiting node, reject if it can



Mummer algorithm 2: chain MUMs

- Compute the *longest increasing subsequence* of matches

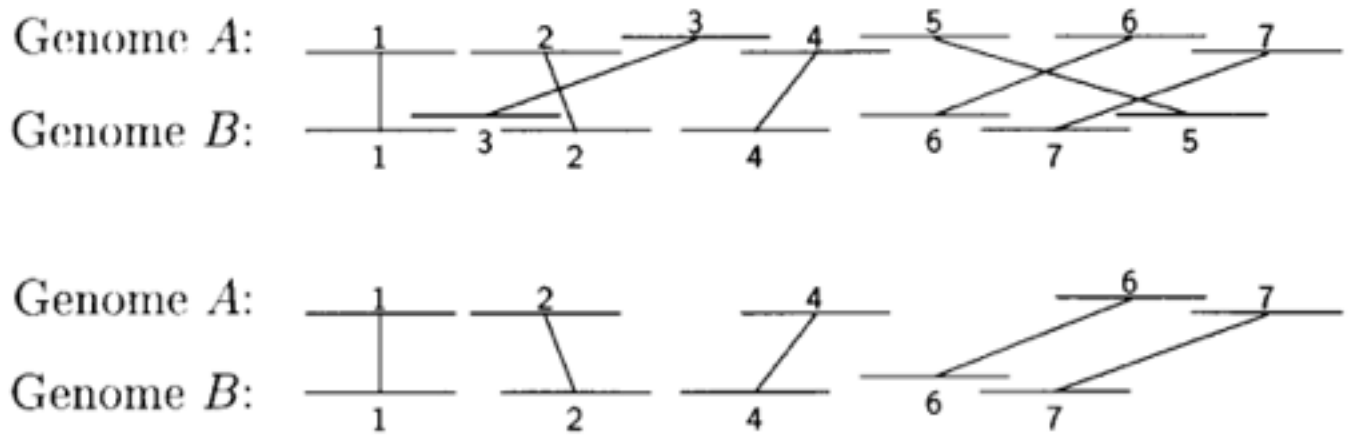


Figure 3. Aligning Genome *A* and Genome *B* after locating the MUMs. Each MUM is here indicated only by a number, regardless of its length. The top alignment shows all the MUMs. The shift of MUM 5 in Genome *B* indicates a transposition. The shift of MUM 3 could be simply a random match or part of an inexact repeat sequence. The bottom alignment shows just the LIS of MUMs in Genome *B*.

Mummer algorithm 3: close gaps

- Use dynamic programming to make local alignments

1. SNP: exactly one base (indicated by ^) differs between the two sequences. It is surrounded by exact-match sequence.

```
Genome A:  cgtcatgggcgttcgtcgttg
Genome B:  cgtcatgggcattcgtcgttg
                     ^
```

2. Insertion: a sequence that occurs in one genome but not the other.

```
Genome A:  cggggtaaccgc.................cctggtcggg
Genome B:  cggggtaaccgcgttgctcggggtaaccgccctggtcggg
                     ^^^^^^^^^^^^^^^^^^
```

3. Highly polymorphic region: many mutations in a short region.

```
Genome A:  ccgcctcgcctgg.gctggcgcccgctc
Genome B:  ccgcctcgccagttgaccgcgcccgctc
                  ^ ^^ ^^^
```

4. Repeat sequence: the repeat is shown in uppercase. Note that the first copy of the repeat in Genome B is imperfect, containing one mismatch to the other three identical copies.

```
Genome A:  cTGGGTGGGACAACGTaaaaaaaaaTGGGTGGGACAACGTc
Genome B:  aTGGGTGGGGCgACGTgggggggggTGGGTGGGACAACGTa
           ^          ^                          ^
```

MUMmer: performance

The time required to generate the alignments was 55 s on a DEC Alpha 4100, broken down as follows: 5 s for suffix tree construction, 45 s for sorting the MUMs and finding the longest increasing sequence and 5 s for generating the Smith–Waterman alignments of the gaps. (Because the sequences are so close to identical, the Smith–Waterman step needs to do very little work.)

MUMmer: alignment

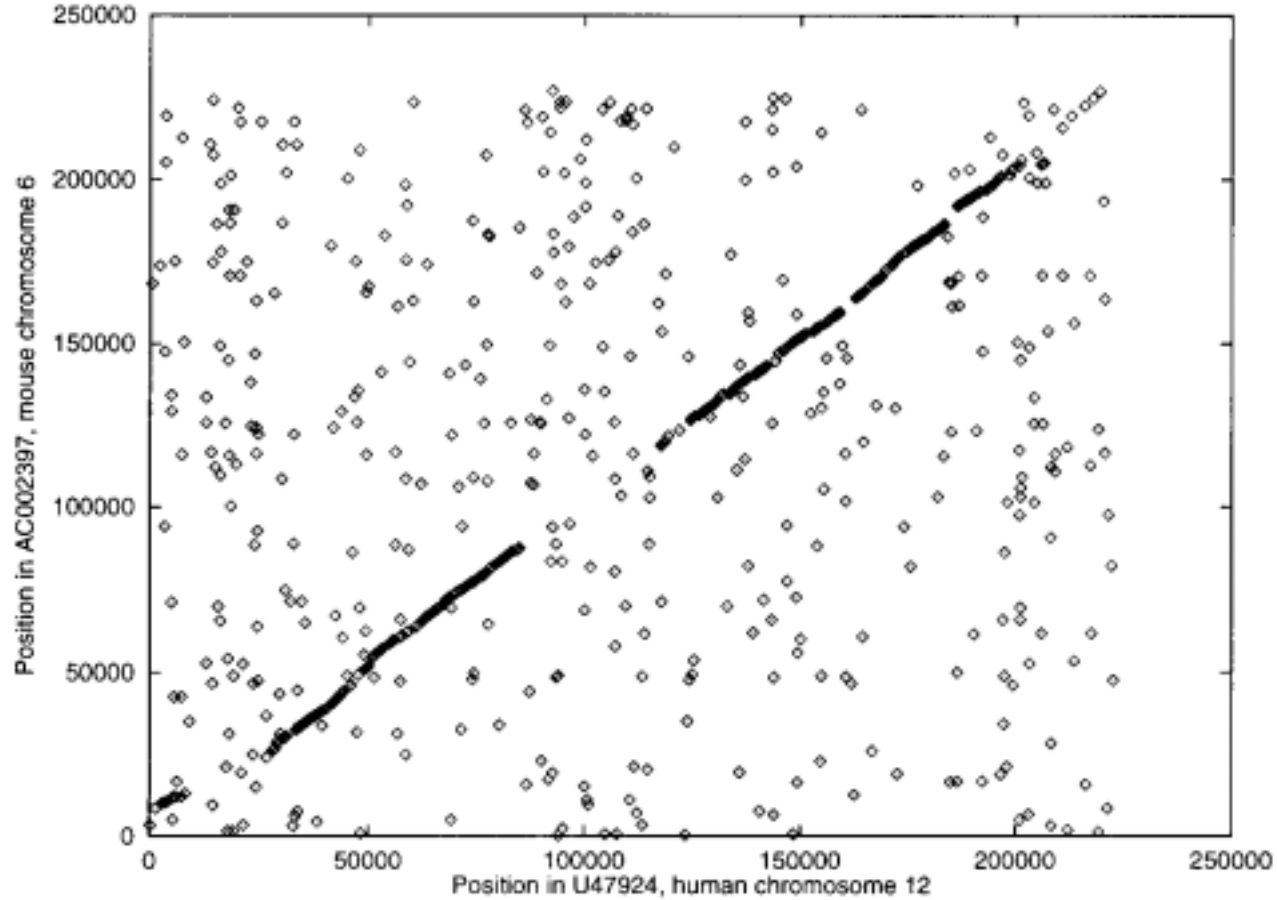
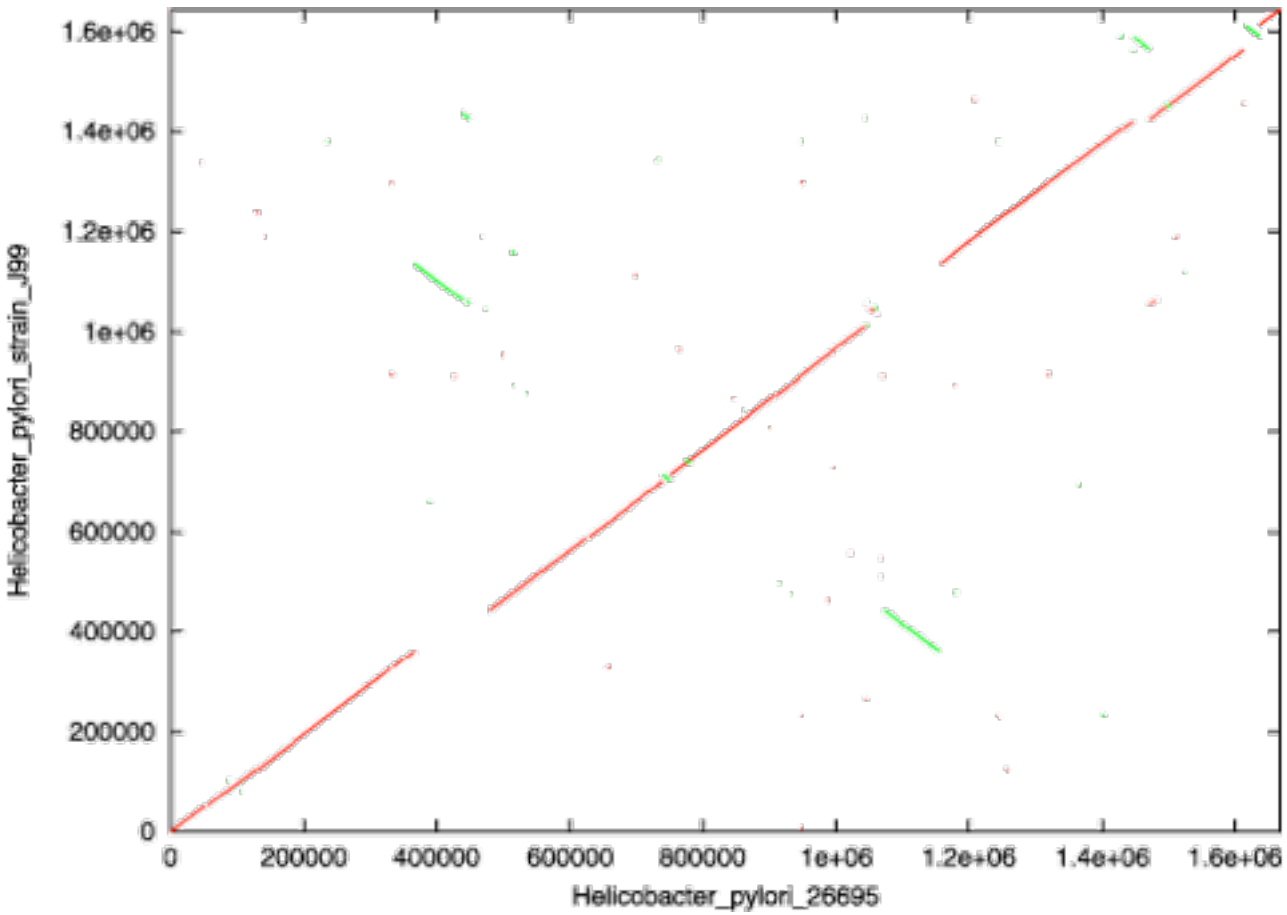


Figure 8. Alignment of a 222 930 bp subsequence of human chromosome 12p13, accession no. U47924, to a 227 538 bp subsequence of mouse chromosome 6, accession no. AC002397. Each point in the plot corresponds to an MUM of ≥15 bp.

MUMmer: alignment

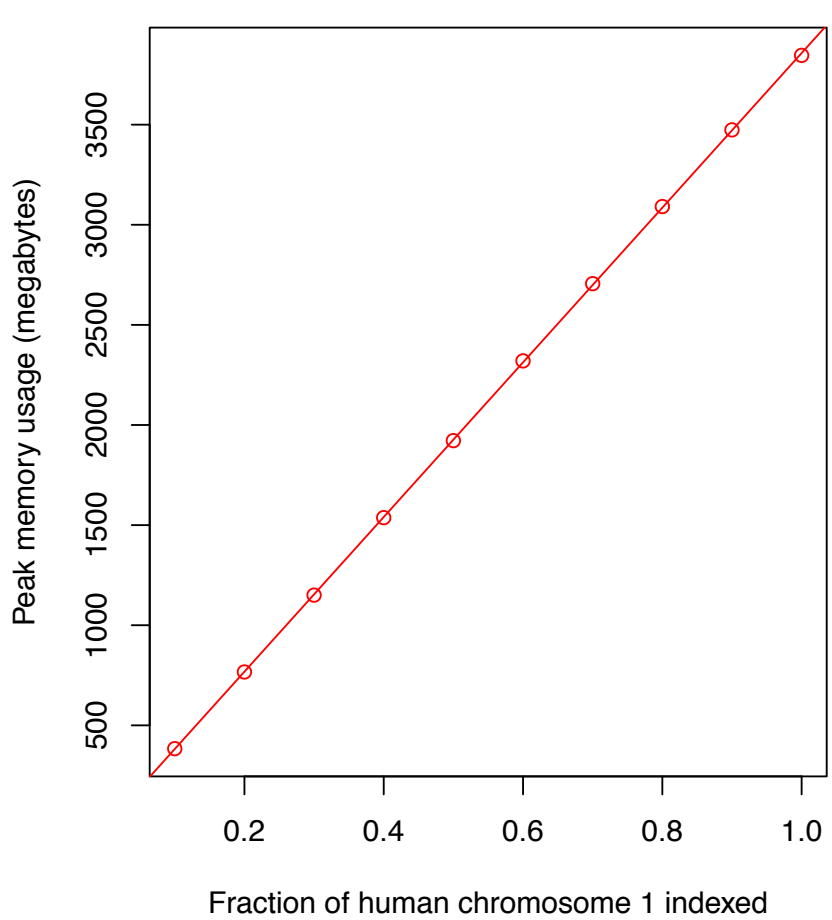


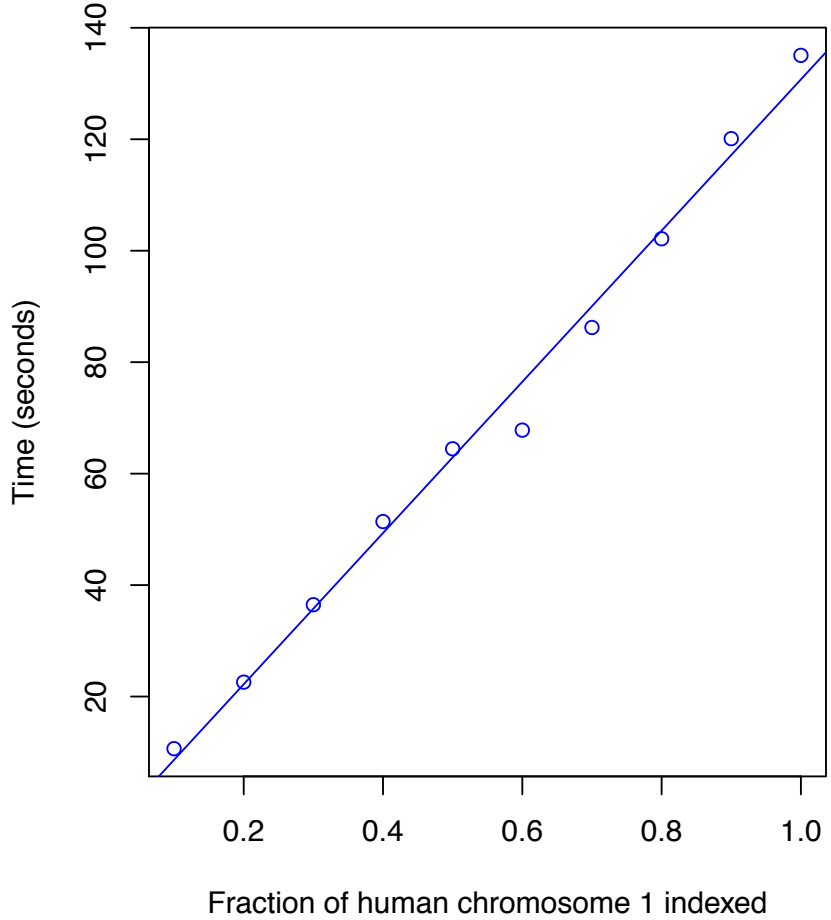
Red = match was between like strands, green = different strands

Axes show different strains of Helicobacter pylori, a bacterium found in the stomach and associated with gastric ulcers

Suffix trees in the real world: MUMmer

MUMmer v3.32 time and memory scaling when indexing increasingly larger fractions of human chromosome 1





For whole chromosome 1, took 2m:14s and used 3.94 GB memory

Suffix trees in the real world: MUMmer

Attempt to build index for whole human genome reference:

```
mummer: suffix tree construction failed: textlen=3101804822 larger
than maximal textlen=536870908
```

We can predict it would have taken about 47 GB of memory