CMSC 858W Lecture Notes

Mihai Pop

3/25/2010

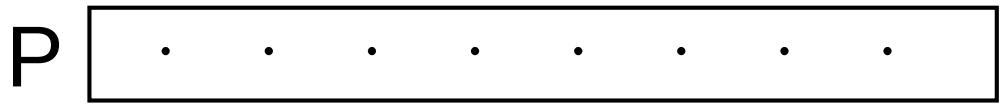
Prepared by Ferhan Ture

Administrivia

- Project proposal
  - Due tomorrow (3/26)
  - Doesn’t need to be formal
- Project deliverables (details later)
  - Report (most points from this one)
    - Academic paper style (~3-4 pages or more)
    - Cover basics of the ideas
  - Code
  - Documentation

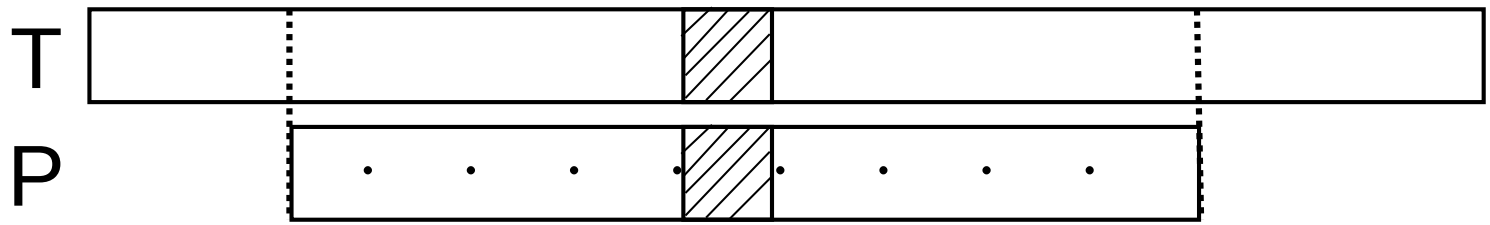
Baeza-Yates Method

- For doing alignment in cases where it's known there will be at most *k* mismatches.



- Based on the pigeonhole principle: If |P| is divided into *k*+1 equally-sized blocks, then at least one block does not contain any mismatches (i.e., there is an exact alignment for that block).

- We can run an exact alignment algorithm (e.g. Aho-Corasick) on each of the *k*+1 blocks. Once we hit a block that aligns exactly to the corresponding part of the text, we run the expensive dynamic programming algorithm:



Baeza-Yates Method

- The aligned shaded region is called a “seed”
- Pattern is more likely to align to a region close to the seed: “first find short exact match, then extend to longer alignment”
- Instead of |T|x|P| running time, the dynamic programming takes |P|x|P| running time.
- It is guaranteed to not miss an alignment

BLAST

- Heavily used alignment program
- Tries all possible mutations of seeds
- Assumes exact alignment of seeds, which are defined as a sequence of consecutive *k* characters.
- Depending on *k*, there are two problems with this:
  - There may be too many matches for a short seed
  - Some regions may be missed if a long seed is used
- IDEA: Use longer but *nonconsecutive* seeds by using the Locality Sensitive Hashing (LSH) approach.

Spaced seeds and LSH

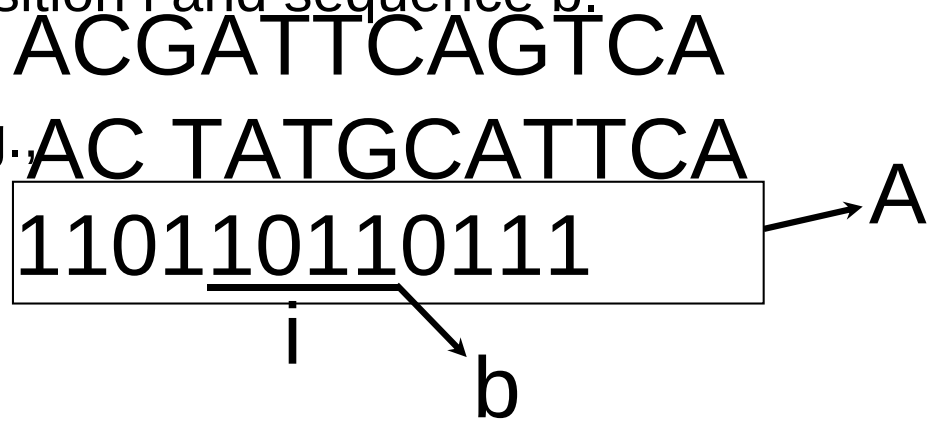
- Idea is to hash similar k-mers to the same hash value.

- If we use a hash function that keeps all characters but the third one (i.e., h(AGGAC)=h(AGCAC)=AGAC))
- The hash function is called a “spaced seed”. It can be denoted by 1s and 0s (e.g. h = 11011)
- How many spaces should we have and where should we put them?
  - If we want to find alignments with at least p% identity, then the proportion of 1s should be p%.
  - In order to determine the spaced seeds in a formal way, we’ll use dynamic programming.

Spaced seeds and LSH

- M = length of seed
- L = length of alignment
- p = proportion of 1s in seed
- i = 1...L-M
- A = alignment
- f_s(i,b)=probability of finding a compatible seed s in first i positions of alignment when last M bits is b

- Note: Seed s is not given. The idea is to iterate over possible s values, check if each is compatible or not, and sum the probability of finding a compatible one, given a position i and sequence b.
- e.g.,

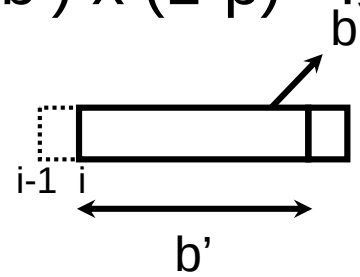


seeds	
0000	compatible
...	
1010	compatible
1011	compatible
1110	incompatible
...	

Spaced seeds and LSH

- Dynamic programming can solve this problem efficiently:

$f_s(0,b)$ = #seeds compatible with A[1...M] / #possible seeds
= #seeds compatible with A[1...M] / 2^M
$f_s(i,b) = f_s(i,0b') \times (1-p) + f_s(i,1b') \times p$



- By dynamic programming, calculate for each seed s,
- $sum_b\ f_s(L-M, b)$ = Pr(seed s matches A) = sensitivity of s
- Select the seed with highest sensitivity.
- This needs to be done only once for a given p, as a preprocessing step. It is an NP-hard problem, therefore heuristics are used to speed up.

Another observation

- Observation by previous student Mohammad Ghodsi:
- If there is an alignment of length L with k mismatches ($r=k/L$ is the error rate), there is at least one sub-alignment of length $M<L$ with error rate less than or equal to r.
- Why useful? When searching through suffix tree, discard searches when you hit to a point with much higher error rate than r.