

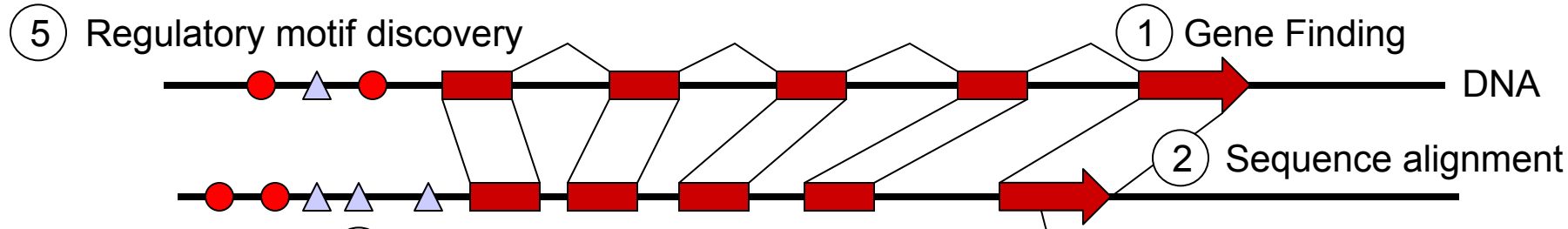
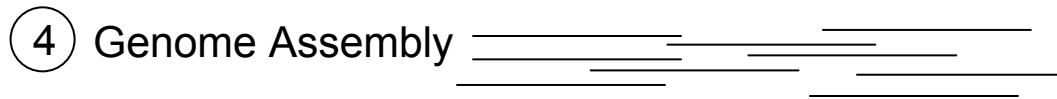
Computational Biology
6.095/6.895

String Matching

Lecture 4

Prof. Piotr Indyk

Challenges in Computational Biology



TCATGCTAT□
 TCGTGATAA□
 TGAGGATAT□
 TTATCATAT□
 TTATGATTT□

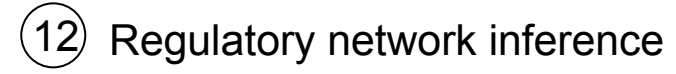
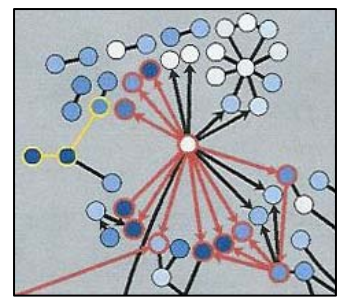
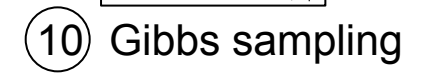
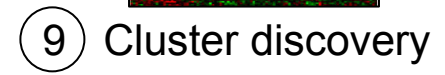
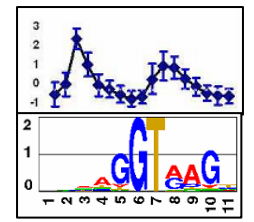
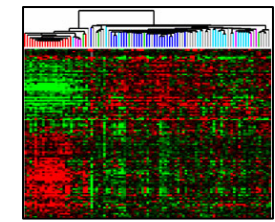
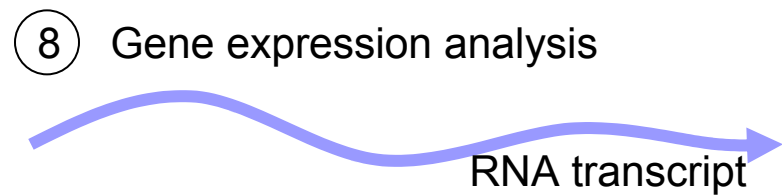
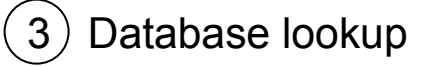
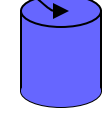


Image removed due to copyright restrictions.

Today

- Exact string matching
- Longest common substring (no gaps)

.TATTGAATTTTCAAAAATTCTTACTTTTTTTTTTGG**ATG**GACGCAAAGAAGTTTAATAATCATATTAC**ATG**GCATTACCACCATATA
ATCCATATCTAATCTTACTTAT**ATG**TTGTGGAA**ATG**TAAAGAGCCCCATTATCTTAGCCTAAAAAACCTTCTCTTTGGAACTTTC
AATACGCTTAACTGCTCATTGCTATATTGAAGTACGGATTAGAAGCCGCCGAGCGGGCGACAGCCCTCCGACGGAAGACTCTCCTC
GCGTCCCTCGTCTTACC GGTCGCGTTCCTGAAACGCAG**ATG**TGCCTCGCGCCGCACTGCTCCGAACAATAAAGATTCTACAATACT
TTTT**ATG**GTT**ATG**AAGAGGAAAAATTGGCAGTAACCTGGCCCCACAAACCTTCAAATTAACGAATCAAATTAACAACCATAGG**ATG**
ATGCGATTAGTTTTTTAGCCTTATTTCTGGGGTAATTAATCAGCGAAGCG**ATG**ATTTTTTGATCTATTAACAGATATATA**ATG**GAA
CTGCATAACCACTTTAACTAATACTTTCAACATTTTCAGTTTGTATTACTTCTTATTCAA**ATG**TCATAAAAGTATCAACAAAAAAT
TAATATACCTCTATACTTTAACGTCAAGGAGAAAAAACTATA**ATG**ACTAAATCTCATT CAGAAGAAGTGATTGTACCTGAGTTCAA
TAGCGCAAAGGAATTACCAAGACCATTGGCCGAAAAGTGCCCGAGCATAATTAAGAAATTTATAAGCGCTT**ATGATG**CTAAACCGG
TTGTTGCTAGATCGCCTGGTAGAGTCAATCTAATTGGTGAACATATTGATTATTGTGACTTCTCGGTTTTACCTTTAGCTATTGAT
GAT**ATG**CTTTTGCGCCGTCAAAGTTTTGAACGAGAAAAATCCATCCATTACCTTAATA**ATG**CTGATCCCAAATTTGCTCAAAGGAA
CGATTTGCCGTTGGACGGTTC**ATG**TCACAATTGATCCTTCTGTGTCGGACTGGTCTAATTACTTTAA**ATG**TGGTCTCC**ATG**TTG
ACTCTTTTCTAAAGAACTTGCACCGGAAAGGTTTGCCAGTGCTCCTCTGGCCGGGCTGCAAGTCTTCTGTGAGGGTG**ATG**TACCA
GGCAGTGGATTGTCTTCTTTCGGCCGCATTCAATTTGTGCCGTTGCTTTAGCTGTTGTTAAAGCGAAT**ATG**GGCCCTGGTTATCAT**AT**
CAAGCAAAATTTA**ATG**CGTATTACGGTCGTTGCAGAACATT**ATG**TTGGTGTAA**ATG**GCAGT**ATG**GATCAGGCTGCCTCTGTTT
GTGAGGAAGATC**ATG**CTCTATACGTTGAGTTCAAACCGCAGTTGAAGGCTACTCCGTTTAAATTTCCGCAATTA
AGCTTTGTTATTGCGAACACCCTTGTGTATCTAACAAAGTTTGAAACCGCCCCAACCAACTATAATTTAAGAGTGGTAGAAGTCAC
AGCTGCAA**ATG**TTTTAGCTGCCACGTACGGTGTGTTTTACTTTCTGGAAAAGAAGGATCGAGCACGAATAAAGGTAATCTAAGAG
TC**ATG**AACGTTTATT**ATG**CCAGATATCACAACTTTCCACACCCTGGAACGGCGATATTGAATCCGGCATCGAACGGTTAACAAAG
CTAGTACTAGTTGAAGAGTCTCTCGCCAATAAGAAACAGGGCTTTAGTGTGACG**ATG**TCGCACAATCCTTGAATTGTTCTCGCGA
ATTCACAAGAGACTACTTAACAACATCTCCAGTGAGATTTCAAGTCTTAAAGCTATATCAGAGGGCTAAGC**ATG**TGTATTCTGAAT
TAAGAGTCTTGAAGGCTGTGAAATTA**ATG**ACTACAGCGAGCTTTACTGCCGACGAAGACTTTTTCAAGCAATTTGGTGCCTTG**ATG**
GAGTCTCAAGCTTCTTTCGATAAACTTTACGA**ATG**TTCTTGTCCAGAGATTGACAAAATTTGTTCCATTGCTTTGTCAA**ATG**GATC
ATGGTTCCCGTTTGACCGGAGCTGGCTGGGGTGGTTGTACTGTTCACTTGGTTCCAGGGGGCCCAA**ATG**GGCAACATAGAAAAGGTAA
AAGCCCTTGCCA**ATG**AGTTCTACAAGGTCAAGTACCCTAAGATCACTG**ATG**CTGAGCTAGAAA**ATG**CTATCATCGTCTCTAAACCA
TTGGGCAGCTGTCTAT**ATG**AATTATAAGTATACTTCTTTTTTTTACTTTGTT CAGAACA
ACTTCTCATTTTTTTTCTACTCATAACT
.GCATCACAAAATACGCAATAATAACGAGTAGTAACACTTTTTATAGTT CATA**ATG**CTTCAACTACTTAATA**ATG**ATTGT**ATG**ATA
TTTTCA**ATG**TAAGAGATTTTCGATTATCCACAACTTTAAAACACAGGGACAAAATTTCTTGAT**ATG**CTTTCAACCGCTGCGTTTTGG
.CCTATTCTTGAC**ATG**AT**ATG**ACTACCATTTTGTATTGTACGTGGGGCAGTTGACGTCTTATCAT**ATG**TCAAAGTCATTTGCGAAG
TTGGCAAGTTGCCAACTGACGAG**ATG**CAGTAAAAAGAGATTGCCGTCTTGAAACTTTTTGTCTTTTTTTTTTCCGGGGACTCTAC
AACCTTTGTCCTACTGATTAATTTTGTACTGAATTTGGACAATTCAGATTTTAGTAGACAAGCGCGAGGAGGAAAAGAA**ATG**GACA
AAATTC**ATG**GACAAGAAGATAGGAAAAAAGCTTTCACCGATTTCTTAGACCGGAAAAAGTCGT**ATG**ACATCAGA**ATG**A
ATTTTCAAGTTAGACAAGGACAAAATCAGGACAAATTTGTAAAGATATAATAAACTATTTGATTCAGCGCCAATTTGCCCTTTTCCA
TCCATTAAATCTCTGTTCTCTTACTTATA**ATGATG**ATTAGGTATCATCTGTATAAACTCCTTTCTTAATTTCACTCTAAAGCAT
CCATAGAGAAGATCTTTCGGTTCGAAGACATTCCTACGCATAATAAGAATAGGAGGGAATA**ATG**CCAGACAATCTATCATTACATT
.GCGGCTCTTCAAAAAGATTGAACTCTCGCCAAC**ATG**GAATCTTCCA**ATG**AGACCTTTGCGCCAATA**ATG**TGGATTTGGAAAAA
TATAAGTCATCTCAGAGTAATATAACTACCGAAGTTT**ATG**AGGCATCGAGCTTTGAAGAAAAGTAAGCTCAGAAAAACCTCAATA
CTCATTCTGGAAGAAAATCTATT**ATG**AAT**ATG**TGGTTCGTTGACAAATCAATCTTGGGTGTTTCTATTCTGGATTCAATTT**ATG**TACA
AGGACTTGAAGCCCGTCGAAAAAGAAAGGCGGGTTCCTGCTGCTAGCAATTTATTGTTACTTCTGCTTCTGCTG**ATG**TTTCAATATC
ACTTGGCAAATTCAGCTACAGGTCTACAACCTGGGTCTAAATTTGGTGGCAGTGTGGATAACAATTTGGATTGGGTACGGTTTTCGT
.TGCCTTTTCTTCTTTTTCCTCTAGAGTTGGATCTGCTTATCATTGTTCATTCCCTATATCACTCAGAGCATCAATTCCTATTTTTCT

Exact String Matching

- **Input:** Two strings $T[1..n]$ and $P[1..m]$, containing symbols from alphabet Σ .

E.g. :

– $\Sigma = \{A, G, T, C\}$

– $T[1..9] = \text{“CAGTACATCGA...”}$

– $P[1..3] = \text{“AGT”}$

- **Goal:** find all “shifts” $1 \leq s \leq n-m$ such that $T[s+1..s+m] = P$
- Ideas ?

Simple Algorithm

```
for  $s \leftarrow 0$  to  $n-m$   
     $Match \leftarrow 1$   
    for  $j \leftarrow 1$  to  $m$   
        if  $T[s+j] \neq P[j]$  then  
             $Match \leftarrow 0$   
            exit loop  
    if  $Match=1$  then output  $s$ 
```

Analysis

- Running time of the simple algorithm:
 - Worst-case: $O(nm)$
 - Average-case (random text): $O(n)$ (expectation)
 - T_s = time spent on checking shift s
(the number of comparisons until 1st mismatch)
 - $E[T_s] \leq ? 2$
 - $E[\sum_s T_s] = \sum_s E[T_s] = O(n)$

Worst-case

- Is it possible to achieve $O(n)$ for *any* input ?
 - Knuth-Morris-Pratt'77: deterministic
 - Karp-Rabin'81: randomized
- Digression: what is the difference between
 - Algorithm that is fast on a random input (as seen on the previous slide)
 - Randomized algorithm (as in the rest of this lecture)

Karp-Rabin Algorithm

- **Idea:** *semi-numerical* approach:
 - Consider all **m**-mers:
 $T[1\dots m], T[2\dots m+1], \dots, T[m-n+1\dots n]$
 - Map each $T[s+1\dots s+m]$ into a *number* t_s
 - Map the pattern $P[1\dots m]$ into a *number* p
 - Report the **m**-mers that map to the same value as p
- **Problem:** how to map all **m**-mers in $O(n)$ time ?

Implementation

- **Attempt I:**

- Assume $\Sigma = \{0, 1\}$

- (for $\{A, G, T, C\}$ convert: $A \rightarrow 00$, $G \rightarrow 01$, $A \rightarrow 10$, $G \rightarrow 11$)

- Think about each $T[s+1 \dots s+m]$ as a number in binary representation, i.e.,

$$t_s = T[s+1]2^{m-1} + T[s+2]2^{m-2} + \dots + T[s+m]2^0$$

- Find a fast way of computing t_{s+1} given t_s

- Output all s such that t_s is equal to the number p represented by P

Magic formula

- How to transform

$$t_s = \underline{T[s+1]2^{m-1}} + T[s+2]2^{m-2} + \dots + T[s+m]2^0$$

into

$$t_{s+1} = T[s+2]2^{m-1} + T[s+3]2^{m-2} + \dots + \underline{T[s+m+1]2^0} ?$$

- Three steps:
 - Subtract $T[s+1]2^{m-1}$
 - Multiply by 2 (i.e., shift the bits by one position)
 - Add $T[s+m+1]2^0$
- Therefore: $t_{s+1} = (t_s - T[s+1]2^{m-1}) * 2 + T[s+m+1]2^0$

Algorithm

$$t_{s+1} = (t_s - T[s+1]2^{m-1}) * 2 + T[s+m+1]2^0$$

- Can compute t_{s+1} from t_s using 3 arithmetic operations
- Therefore, we can compute all t_0, t_1, \dots, t_{n-m} using $O(n)$ arithmetic operations
- We can compute a number corresponding to P using $O(m)$ arithmetic operations
- Are we done ?

Problem

- To get $O(n)$ time, we would need to perform each arithmetic operation in $O(1)$ time
- However, the arguments are m -bit long !
- If m large, it is unreasonable to assume that operations on such big numbers can be done in $O(1)$ time
- We need to reduce the number range to something more manageable

Hashing

- We will instead compute

$$t'_s = T[s+1]2^{m-1} + T[s+2]2^{m-2} + \dots + T[s+m]2^0 \pmod{q}$$

where q is an “appropriate” prime number

- One can still compute t'_{s+1} from t'_s :

$$t'_{s+1} = (t'_s - T[s+1]2^{m-1}) * 2 + T[s+m+1]2^0 \pmod{q}$$

- If q is not large, we can compute all t'_s (and p') in $O(n)$ time

Problem

- Unfortunately, we can have **false positives**, i.e., $T[s+1\dots s+m] \neq P$ but $t_s \bmod q = p \bmod q$
- Our approach:
 - Use a **random q**
 - Show that the probability of a false positive is small \rightarrow randomized algorithm

False positive probability

- Consider any $t_s \neq p$. We know that both numbers are in the range $\{0 \dots 2^m - 1\}$
 - How many primes q are there such that
$$t_s \bmod q = p \bmod q \equiv (t_s - p) = 0 \pmod{q} ?$$
 - Such prime has to divide $x = (t_s - p) \leq 2^m$
 - Represent $x = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$, p_i prime, $e_i \geq 1$
- What is the largest possible value of k ?
- Since $2 \leq p_i$, we have $x \geq 2^k$
 - But $x \leq 2^m$
 - $k \leq m$
- There are $\leq m$ primes dividing x

Algorithm

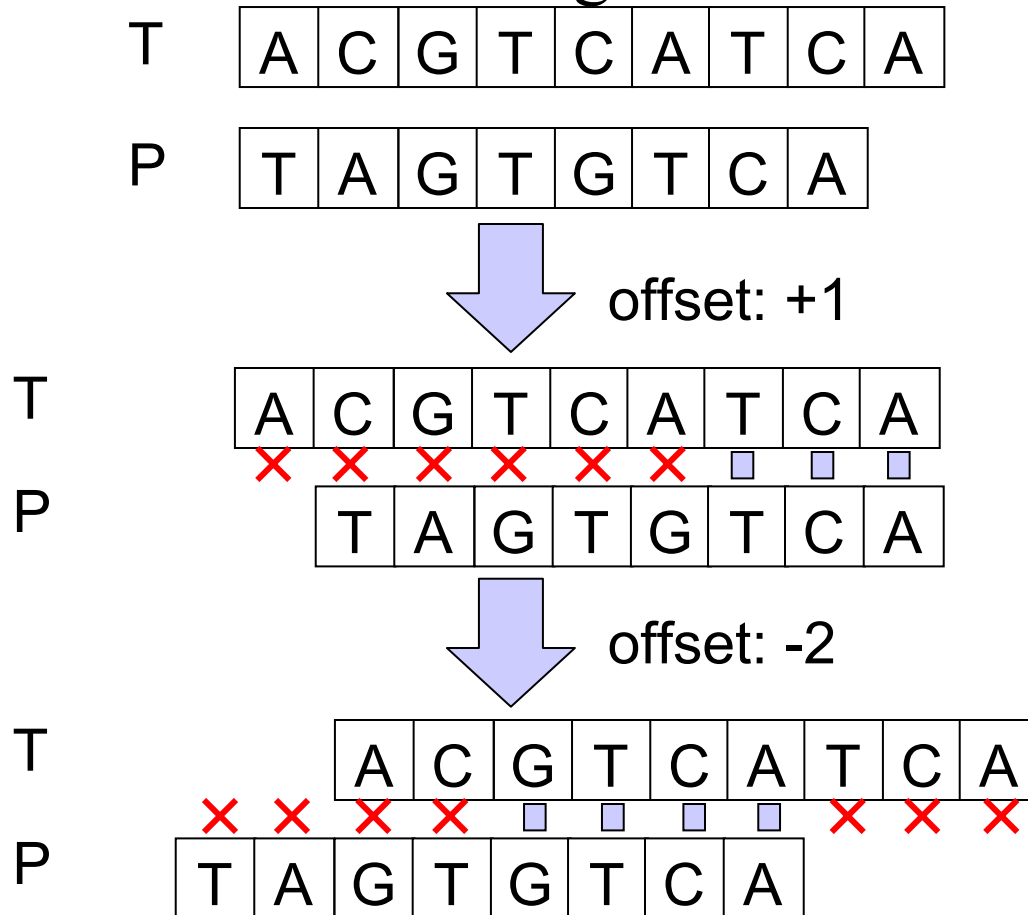
- Algorithm:
 - Let Π be a set of $2nm$ “small” primes
 - Choose q uniformly at random from Π
 - Compute $t_0 \bmod q, t_1 \bmod q, \dots$, and $p \bmod q$
 - Report s such that $t_s \bmod q = p \bmod q$
- Analysis:
 - For each s , the probability that $T[s+1 \dots s+m] \neq P$ but $t_s \bmod q = p \bmod q$ is at most $m/2nm = 1/2n$
 - The probability of *any* false positive is at most $(n-m)/2n \leq 1/2$

Ignored “Details”

- How do we know that such Π exists ?
(That is, a set of $2nm$ “small” primes)
- How do we choose a random prime from Π
in $O(n)$ time ?

Aligning two (ungapped) strings

- Given two possibly related strings T and P
 - What is the longest common substring? (no gaps)



Aligning two sequences

- Longest common substring (LCS) problem (no gaps):
 - **Input:** Two strings $T[1\dots n]$ and $P[1\dots m]$, $n \geq m$
 - **Goal:** Largest k such that
$$T[i+1\dots i+k] = P[j+1\dots j+k]$$
for some i, j
- How can we solve this problem efficiently ?
 - **Hint:** How can we check if LCS has length $\geq k$?

Checking for a common **m**-mer

- Algorithm:
 - Compute hashes t'_0, t'_1, \dots from **T**
 - Compute hashes p'_0, p'_1, \dots from **P**
 - Check if $t'_i = p'_j$ for some pair i, j
 - Sort all hashes
 - Scan the sorted list to find equal hashes created from **T** and **P**

Analysis

Radix Sort

- Algorithm:
 - Compute hashes t'_0, t'_1, \dots from T $O(n)$
 - Compute hashes p'_0, p'_1, \dots from P $O(m)$
 - Check if $t'_i = p'_j$ for some pair i, j
 - Sort all hashes ~~$O(n \log n)$~~ $O(n)$
 - Scan the sorted list to find equal hashes created from T and P $O(n)$
- Total:** $O(n)$

Longest common substring

- We can check if LCS has length $\geq k$ in $O(n)$ time
- How to find the largest such k ?
- *Binary search !*
- Total time: $O(n \log n)$