Lecture 2

Sequence Alignment and Dynamic Programming
<table>
<thead>
<tr>
<th>Project</th>
<th>Sets</th>
<th>Week</th>
<th>Date</th>
<th>Topic</th>
<th>Lec</th>
<th>Topic</th>
<th>Read*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describe your previous research, areas of interest in computational biology, type of project that best fits your interests. Post in a profile that lets your classmates know you find potential partners.</td>
<td>PS1 out on L1-L5</td>
<td>1</td>
<td>Thu, Sep 8</td>
<td>Introduction</td>
<td>L1</td>
<td>Intro: Biology, Algorithms, Machine Learning, Course Overview</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Sep 9</td>
<td></td>
<td>R1</td>
<td>Recitation 1: Biology and Probability Review</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Tue, Sep 13</td>
<td>Module I: Aligning and Modeling Genomes</td>
<td>L2</td>
<td>Alignment I: Dynamic Programming, Global and local alignment</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Sep 15</td>
<td></td>
<td>L3</td>
<td>Alignment II: Database search, Rapid string matching, BLAST, BLOSUM</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Sep 16</td>
<td></td>
<td>R2</td>
<td>Recitation 2: Deriving Parameters of Alignment, Multiple Alignment</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Tue, Sep 20</td>
<td>Foundations</td>
<td>L4</td>
<td>Hidden Markov Models Part 1: Evaluation/ Parsing, Viterbi, Forward algorithms</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Sep 22</td>
<td></td>
<td>L5</td>
<td>Hidden Markov Models Part 2: Posterior Decoding, Learning, Baum-Welch</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Sep 23</td>
<td></td>
<td>R4</td>
<td>No classes - student holiday</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Tue, Sep 27</td>
<td>Project profile due Tue 9/27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Thu, Sep 29</td>
<td>Module II: Gene Expression and Networks</td>
<td>L6</td>
<td>Expression Analysis: Clustering/Classification, K-means, Hierarchical, Bayesian</td>
<td>15,16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Sep 30</td>
<td></td>
<td>L7</td>
<td>Transcript structure: GenScan, RNA-seq, Mapping, De novo Assembly, Diff Expr</td>
<td>14,15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Tue, Oct 4</td>
<td></td>
<td>R3</td>
<td>Recitation 3: Affinity Propagation Clustering and Random Forest Classification</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Oct 6</td>
<td></td>
<td>L8</td>
<td>Epigenomics: ChIP-Seq, Read mapping, Peak calling, IDR, Chromatin states</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Oct 7</td>
<td></td>
<td>R9</td>
<td>Three-dimensional chromatin interactions: 3C, 5C, HiC, Chi-Pa</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Tue, Oct 11</td>
<td>Foundations</td>
<td>L9</td>
<td>Regulation of Gene Expression</td>
<td>20,21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Oct 13</td>
<td></td>
<td>L10</td>
<td>Biological Networks: Neural Networks, Belief Networks, Deep learning</td>
<td>20,21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Oct 14</td>
<td></td>
<td>L11</td>
<td>Networks I: Bayesian inference, Variational Bayes, approximate inference</td>
<td>20,21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Tue, Oct 18</td>
<td></td>
<td>R6</td>
<td>Networks II: Probabilistic Graphical Models</td>
<td>20,21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Oct 20</td>
<td></td>
<td>R7</td>
<td>No classes - student holiday</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Oct 21</td>
<td></td>
<td>R8</td>
<td>No classes - student holiday</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Tue, Oct 25</td>
<td>Project Planning: research areas, initial ideas, type of project, mentor matching, finding partners 32D-507</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Oct 27</td>
<td></td>
<td>L12</td>
<td>DNA-ACT: The Genomic Landscape</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Tue, Oct 28</td>
<td></td>
<td>L13</td>
<td>Disease Association Mapping, GWAS, organismal phenotypes</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Oct 30</td>
<td></td>
<td>R7</td>
<td>Linkage Disequilibrium: Haplotype Phasing, Genotype Imputation</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Oct 31</td>
<td></td>
<td>R8</td>
<td>No classes - student holiday</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Tue, Nov 1</td>
<td>Foundations</td>
<td>L14</td>
<td>Comparative Genomics and evolutionary signatures</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Nov 3</td>
<td></td>
<td>L17</td>
<td>Genome Scale Evolution, Genome Duplication</td>
<td>6,6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Nov 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Tue, Nov 8</td>
<td>Midcourse report due Wed 11/23</td>
<td>No Recitation, Veterans Day</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Nov 10</td>
<td></td>
<td>L18</td>
<td>No Recitation, Veterans Day</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Nov 11</td>
<td></td>
<td>L19</td>
<td>Phylogenetics: Molecular evolution, Tree building, Phylogenetic inference</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Tue, Nov 15</td>
<td></td>
<td>L20</td>
<td>Phylogenomics: Gene/species trees, reconciliation, coalescent, ARGs</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Nov 17</td>
<td></td>
<td>R9</td>
<td>Phylogenetic distance metrics, Coalescent Process</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Nov 18</td>
<td></td>
<td>R11</td>
<td>No classes - student holiday</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Thu, Nov 22</td>
<td>Foundations</td>
<td>L21</td>
<td>Quiz: In Class Quiz (the only quiz - the class has no final exam) - covers L1-L20, R1-R9</td>
<td>29,30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Tue, Nov 29</td>
<td></td>
<td>L22</td>
<td>Resolving human ancestry and human history from genetic data</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Dec 1</td>
<td></td>
<td>L23</td>
<td>Mining human phenotypes, PheWAS, UK Biobank, meta-phenotypes+impuation</td>
<td>34,35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Dec 2</td>
<td></td>
<td>R10</td>
<td>Recitation 10: Project Feedback, results, interpretation, directions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>Tue, Dec 6</td>
<td>Foundations</td>
<td>L24</td>
<td>Engineering with CRISPR/Cas9 and related technologies</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Dec 8</td>
<td></td>
<td>L24</td>
<td>Genome Engineering with CRISPR/Cas9 and related technologies</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fri, Dec 9</td>
<td></td>
<td>R11</td>
<td>Recitation 11: Presentation Tips - Intro, discussion, Slides, Presentation skills</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Tue, Dec 13</td>
<td>Foundations</td>
<td>L25</td>
<td>Final Presentations - Part I (11am) - 32G reading room</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thu, Dec 13</td>
<td></td>
<td>L25</td>
<td>Final Presentations - Part I (1pm) - 32-141</td>
<td>37</td>
</tr>
</tbody>
</table>

* readings refer to chapters in compiled 2015 scribe notes, available in the materials folder on Stellar
** recitation topics will be adjusted to respond to lecture and student needs
Module 1: Computational foundations
- Dynamic programming: exploring exponential spaces in poly-time
- Introduce Hidden Markov Models (HMMs): Central tool in CS
- HMM algorithms: Decoding, evaluation, parsing, likelihood, scoring

This week: Sequence alignment / comparative genomics
- Local/global alignment: infer nucleotide-level evolutionary events
- Database search: scan for regions that may have common ancestry

Next week: Modeling genomes / exon / CpG island finding
- Modeling class of elements, recognizing members of a class
- Application to gene finding, conservation islands, CpG islands
Genome-wide alignments reveal orthologous segments

- Genome-wide alignments span entire genome
- Comparative identification of functional elements
Comparative genomics reveals conserved regions

- **Comparative genomics can reveal functional elements**
  - For example: exons are deeply conserved to mouse, chicken, fish
  - Many other elements are also strongly conserved: exons / regulatory?

- **Develop methods for estimating the level of constraint**
  - Count the number of edit operations, number of substitutions and gaps
  - Estimate the number of mutations (including estimate of back-mutations)
  - Incorporate information about neighborhood: conservation ‘windows’
  - Estimate the probability of a constrained ‘hidden state’: HMMs next week
  - Use phylogeny to estimate tree mutation rate, or ‘rejected substitutions’
  - Allow different portions of the tree to have different rates: phylogenetics
Evolutionary signatures for diverse functions

**Protein-coding genes**
- Codon Substitution Frequencies
- Reading Frame Conservation

**RNA structures**
- Compensatory changes
- Silent G-U substitutions

**microRNAs**
- Shape of conservation profile
- Structural features: loops, pairs
- Relationship with 3’UTR motifs

**Regulatory motifs**
- Mutations preserve consensus
- Increased Branch Length Score
- Genome-wide conservation

Stark et al, Nature 2007
**Alignment: Evolution preserves functional elements!**

**Gal10**

**Yeast** (Kellis et al., Nature 2003), **Mammals** (Xie, Nature 2005), **Fly** (Stark et al., Nature 2007)

**Alignment**

**Factor footprint**

**Conservation island**

---

**GAL1**

---

**GAL4**

---

**We can ‘read’ evolution to reveal functional elements**

---

Yeast (Kellis et al., Nature 2003), Mammals (Xie, Nature 2005), Fly (Stark et al., Nature 2007)
Today’s goal:
How do we actually align two genes?
Goal: Sequence Alignment / Dynamic Programming

1. Introduction to sequence alignment
   - Comparative genomics and molecular evolution
   - From Bio to CS: Problem formulation
   - Why it’s hard: Exponential number of alignments

2. Introduction to principles of dynamic programming
   - Computing Fibonacci numbers: Top-down vs. bottom-up
   - Repeated sub-problems, ordering compute, table lookup
   - DP recipe: (1) Parameterization, (2) sub-problem space, (3) traversal order, (4) recursion formula, (5) trace-back

3. DP for sequence alignment
   - Additive score, building up a solution from smaller parts
   - Prefix matrix: finite subproblems, exponential paths
   - Duality: each entry $\Leftrightarrow$ prefix alignment score; path $\Leftrightarrow$ alignment

4. Advanced topics: Dynamic Programming variants
   - Linear-time bounded DP(heuristic). Linear-space DP. Gaps
   - Importance of parameterization: 2-D vs. 4-D decomposition
Genomes change over time

begin

A C G T C A T C A

mutation

A C G T G A T C A

deletion

A X G T G X T C A

A G T G T C A

insertion

T A G T G T C A

tag C G T C A

end
Goal of alignment: Infer edit operations

begin

```
A C G T C A T C A
```

? |

end

```
T A G T G T C A
```
From Bio to CS: Formalizing the problem

- Define set of evolutionary operations (insertion, deletion, mutation)
  - Symmetric operations allow time reversibility (part of design choice)

  (Exception: methylated CpG dinucleotides $\rightarrow$ TpG/CpA non-symmetric)

- Define optimality criterion (min number, min cost)
  - Impossible to infer exact series of operations (Occam’s razor: find min)

- Design algorithm that achieves that optimality (or approximates it)
  - Tractability of solution depends on assumptions in the formulation

 Note: Not all decisions are conflicting (some are both relevant and tractable) (e.g. Pevzner vs. Sankoff and directionality in chromosomal inversions)
Formulation 1: Longest common substring

- Given two possibly related strings S1 and S2
  - What is the longest common substring? (no gaps)

<table>
<thead>
<tr>
<th>S1</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>T</td>
<td>A</td>
<td>G</td>
<td>T</td>
<td>G</td>
<td>T</td>
<td>C</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

Offset: +1

<table>
<thead>
<tr>
<th>S1</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>T</td>
<td>A</td>
<td>G</td>
<td>T</td>
<td>G</td>
<td>T</td>
<td>C</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

Offset: -2

<table>
<thead>
<tr>
<th>S1</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>T</td>
<td>A</td>
<td>G</td>
<td>T</td>
<td>G</td>
<td>T</td>
<td>C</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>
Formulation 2: Longest common subsequence

• Given two possibly related strings S1 and S2
  – What is the longest common subsequence? (gaps allowed)

S1

\[
\begin{array}{ccccccc}
A & C & G & T & C & A & T & C & A \\
\end{array}
\]

S2

\[
\begin{array}{ccccccc}
T & A & G & T & G & T & C & A \\
\end{array}
\]

Related to:
Edit distance:
• Number of changes needed for S1→S2
• Uniform scoring function
Formulation 3: Sequence alignment

- **Allow gaps (fixed penalty)**
  - Insertion & deletion operations
  - Unit cost for each character inserted or deleted

- **Varying penalties for edit operations**
  - Transitions (Pyrimidine $\leftrightarrow$ Pyrimidine, Purine $\leftrightarrow$ Purine)
  - Transversions (Purine $\leftrightarrow$ Pyrimidine changes)
  - Polymerase confuses Aw/G and Cw/T more often

### Scoring function:

- **Match** $(x,x) = +1$
- **Mismatch** $(A,G) = -\frac{1}{2}$
- **Mismatch** $(C,T) = -\frac{1}{2}$
- **Mismatch** $(x,y) = -1$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>G</th>
<th>T</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>+1</td>
<td>$-\frac{1}{2}$</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>G</td>
<td>$-\frac{1}{2}$</td>
<td>+1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>T</td>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>$-\frac{1}{2}$</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td>-1</td>
<td>$-\frac{1}{2}$</td>
<td>+1</td>
</tr>
</tbody>
</table>

**Transitions:**
- $A \leftrightarrow G$, $C \leftrightarrow T$ common (lower penalty)

**Transversions:**
- All other operations

- **purine**
- **pyrimid.**
Formulation 4: Varying gap cost models

1. Linear gap penalty
   – Same as before
2. Affine gap penalty
   – Big initial cost for starting or ending a gap
   – Small incremental cost for each additional character
3. General gap penalty
   – Any cost function
   – No longer computable using the same model
4. Frame-aware gap penalty
   – Multiples of 3 disrupt coding regions
5. Seek duplicated regions, rearrangements, …
   – Etc
How many alignments are there?

• Longest ‘non-boring’ alignment: n+m entries
  – Otherwise a gap will be aligned to a gap → condense

• Alignment is equivalent to gap placement
  – \((n+m)\choose n\) ways to choose S1 placement
  – At each position yes/no answer of placing character
  – Exponential number of possible placements

• Exponential number of sequence alignment
  – Enumerating and scoring each of them not an option
  – Need faster solution for finding best alignment

Need polynomial algorithm to find best alignment amongst an exponential number of possible alignments!
Goal: Sequence Alignment / Dynamic Programming

1. Introduction to sequence alignment
   - Comparative genomics and molecular evolution
   - From Bio to CS: Problem formulation
   - Why it’s hard: Exponential number of alignments

2. Introduction to principles of dynamic programming
   - Computing Fibonacci numbers: Top-down vs. bottom-up
   - Repeated sub-problems, ordering compute, table lookup
   - DP recipe: (1) Parameterization, (2) sub-problem space, (3) traversal order, (4) recursion formula, (5) trace-back

3. DP for sequence alignment
   - Additive score, building up a solution from smaller parts
   - Prefix matrix: finite subproblems, exponential paths
   - Duality: each entry $\leftrightarrow$ prefix alignment score; path $\leftrightarrow$ alignment

4. Advanced topics: Dynamic Programming variants
   - Linear-time bounded DP (heuristic). Linear-space DP. Gaps
   - Importance of parameterization: 2-D vs. 4-D decomposition
A simple introduction to the principles of Dynamic Programming

Turning exponentials into polynomials
Computing Fibonacci Numbers

• Fibonacci numbers

\[ F_6 = F_5 + F_4 = (F_4 + F_3) + (F_3 + F_2) = \ldots = (3 + 2) + (2 + 1) = 5 + 3 = 8 \]
Fibonacci numbers are ubiquitous in nature:

- Rabbits per generation
- Leaves per height
- Romanesque spirals
- Nautilus size
- Coneflower spirals
- Leaf ordering

Leonardo Pisano, also known as Fibonacci, introduced the sequence that bears his name.
Computing Fibonacci numbers: Top down

• Fibonacci numbers are defined recursively:
  – Python code

```
def fibonacci(n):
    if n==1 or n==2: return 1
    return fibonacci(n-1) + fibonacci(n-2)
```

• Goal: Compute \( n^{th} \) Fibonacci number.
  – \( F(0)=1, F(1)=1, F(n)=F(n-1)+F(n-2) \)
  – 1,1,2,3,5,8,13,21,34,55,89,144,233,377,…

• Analysis:
  – \( T(n) = T(n-1) + T(n-2) = (\ldots) = O(2^n) \)
Computing Fibonacci numbers: Bottom up

- Bottom up approach
  - Python code

```python
def fibonacci(n):
    fib_table[1] = 1
    fib_table[2] = 1
    for i in range(3, n+1):
        fib_table[i] = fib_table[i-1] + fib_table[i-2]
    return fib_table[n]
```

- Analysis: T(n) = O(n)
Lessons from iterative Fibonacci algorithm

• **What did the iterative solution do?**
  – Reveal identical sub-problems
  – Order computation to enable result reuse
  – Systematically filled-in table of results
  – Expressed larger problems from their subparts

• **Ordering of computations matters**
  – Naïve top-down approach very slow
    • results of smaller problems not available
    • repeated work
  – Systematic bottom-up approach successful
    • Systematically solve each sub-problem
    • Fill-in table of sub-problem results in order.
    • Look up solutions instead of recomputing

<table>
<thead>
<tr>
<th>fib_table</th>
</tr>
</thead>
<tbody>
<tr>
<td>F[1]</td>
</tr>
<tr>
<td>F[2]</td>
</tr>
<tr>
<td>F[3]</td>
</tr>
<tr>
<td>F[4]</td>
</tr>
<tr>
<td>F[5]</td>
</tr>
<tr>
<td>F[6]</td>
</tr>
<tr>
<td>F[7]</td>
</tr>
<tr>
<td>F[8]</td>
</tr>
<tr>
<td>F[9]</td>
</tr>
<tr>
<td>F[10]</td>
</tr>
<tr>
<td>F[12]</td>
</tr>
</tbody>
</table>
Dynamic Programming in Theory

• **Hallmarks of Dynamic Programming**
  – **Optimal substructure**: Optimal solution to problem (instance) contains optimal solutions to sub-problems
  – **Overlapping subproblems**: Limited number of distinct subproblems, repeated many many times

• **Typically for optimization problems** (unlike Fib example)
  – Optimal choice made locally: \( \max( \text{subsolution score}) \)
  – Score is typically added through the search space
  – Traceback common, find optimal path from indiv. choices

• **Middle of the road in range of difficulty**
  – Easier: greedy choice possible at each step
  – DynProg: requires a traceback to find that optimal path
  – Harder: no opt. substr., e.g. subproblem dependencies
### Hallmarks of optimization problems

<table>
<thead>
<tr>
<th>Greedy algorithms</th>
<th>Dynamic Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Optimal substructure</strong>&lt;br&gt;An optimal solution to a problem (instance) contains optimal solutions to subproblems.</td>
<td></td>
</tr>
<tr>
<td><strong>2. Overlapping subproblems</strong>&lt;br&gt;A recursive solution contains a “small” number of distinct subproblems repeated many times.</td>
<td></td>
</tr>
<tr>
<td><strong>3. Greedy choice property</strong>&lt;br&gt;Locally optimal choices lead to globally optimal solution</td>
<td><strong>Greedy Choice is not possible</strong>&lt;br&gt;Globally optimal solution requires trace back through many choices</td>
</tr>
</tbody>
</table>
Dynamic Programming in Practice

• Setting up dynamic programming
  1. Find ‘matrix’ parameterization (# dimensions, variables)
  2. Make sure sub-problem space is finite! (not exponential)
     • If not all subproblems are used, better off using memoization
     • If reuse not extensive, perhaps DynProg is not right solution!
  3. Traversal order: sub-results ready when you need them
     • Computation order matters! (bottom-up, but not always obvious)
  4. Recursion formula: larger problems = F(subparts)
  5. Remember choices: typically F() includes min() or max()
     • Need representation for storing pointers, is this polynomial!

• Then start computing
  1. Systematically fill in table of results, find optimal score
  2. Trace-back from optimal score, find optimal solution
Goal: Sequence Alignment / Dynamic Programming

1. Introduction to sequence alignment
   - Comparative genomics and molecular evolution
   - From Bio to CS: Problem formulation
   - Why it’s hard: Exponential number of alignments

2. Introduction to principles of dynamic programming
   - Computing Fibonacci numbers: Top-down vs. bottom-up
   - Repeated sub-problems, ordering compute, table lookup
   - DP recipe: (1) Parameterization, (2) sub-problem space, (3) traversal order, (4) recursion formula, (5) trace-back

3. DP for sequence alignment
   - Additive score, building up a solution from smaller parts
   - Prefix matrix: finite subproblems, exponential paths
   - Duality: each entry $\leftrightarrow$ prefix alignment score; path $\leftrightarrow$ alignment

4. Advanced topics: Dynamic Programming variants
   - Linear-time bounded DP (heuristic). Linear-space DP. Gaps
   - Importance of parameterization: 2-D vs. 4-D decomposition
(3) How do we apply dynamic programming to sequence alignment?
Key insight #1: Score is additive, smaller to larger

• Compute best alignment recursively
  – For a given aligned pair \((i, j)\), the best alignment is:
    • Best alignment of \(S1[1..i]\) and \(S2[1..j]\)
    • + Best alignment of \(S1[i..n]\) and \(S2[j..m]\)
  – Proof: cut-and-paste argument (see 6.046)

This allows a single recursion (top-left to bottom-right) instead of two recursions (middle-to-outside top-down)
Key insight #2: compute scores recursively

\[ \text{Compute alignment of CGT vs. TG exactly once} \]
Key insight #3: sub-problems are repeated $\rightarrow$ reuse!

$\Rightarrow$ Identical sub-problems! We can reuse our work!
Solution #1 – Memoization

- Create a big dictionary, indexed by aligned seqs
  - When you encounter a new pair of sequences
  - If it is in the dictionary:
    - Look up the solution
  - If it is not in the dictionary
    - Compute the solution
    - Insert the solution in the dictionary

- Ensures that there is no duplicated work
  - Only need to compute each sub-alignment once!

Top down approach
Solution #2 – Dynamic programming

- Create a big table, indexed by \((i,j)\)
  - Fill it in from the beginning all the way till the end
  - You know that you’ll need every subpart
  - Guaranteed to explore entire search space
- Ensures that there is no duplicated work
  - Only need to compute each sub-alignment once!
- Very simple computationally!

Bottom up approach
Key insight #4: Optimal prefix almt score ⇔ Matrix entry

<table>
<thead>
<tr>
<th></th>
<th>$S_1[1..i]$</th>
<th>i</th>
<th>$S_1[i..n]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_2[1..j]$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>j</td>
<td></td>
<td>$s$</td>
<td></td>
</tr>
<tr>
<td>$S_2[j..m]$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Key insight #5: Optimal alignment $\Leftrightarrow$ Matrix path

Best alignment $\Leftrightarrow$ Best path through the matrix

Goal: Find best path through the matrix
DP approach: iteratively grow best alignment soltn

• Compute all alignment scores from the bottom up
  – Define $M[i,j]$ prefix alignment score of $S_1[1..i]$ and $S_2[1..j]$
  – Fill up table recursively from smaller to bigger alignments
• Express alignment of $S_1[1..i+1]$ and $S_2[1..j+1] \Rightarrow M[i+1,j+1]$
  – One of three possibilities: (1) extend alignment from $M[i,j]$
    (2) extend from $M[i-1,j]$, or (3) extend from $M[i,j-1]$
  – Only a local computation, takes $O(1)$ time!
• Proof of correctness (cut-and-paste argument from 6.006)
  – Best alignment of $S_1[1..i+1]$ and $S_2[1..j+1]$ must be
    composed of best alignments of smaller prefix
  – Proof: otherwise could replace sub and get better overall
Computing alignments recursively: \( M[i,j] = F(\text{smaller}) \)

- **Local** update rules, only look at neighboring cells:
  - Compute next alignment based on previous alignment
  - Just like Fibonacci numbers: \( F[i] = F[i-1] + F[i-2] \)
  - Table lookup avoids repeated computation

- Computing the score of a cell from smaller neighbors
  \[
  M(i,j) = \max \{ M(i-1, j-1) + \text{score}, \quad M(i-1, j) - \text{gap}, \quad M(i, j-1) - \text{gap} \}
  \]
  - Only three possibilities for extending by one nucleotide: a gap in one species, a gap in the other, a (mis)match

- Compute scores for prefixes of increasing length
  - Start with prefixes of length 1, extend by one each time, until all prefixes have been computed
  - When you reach bottom right, alignment score of \( S_1[1..m] \) and \( S_2[1..n] \) is alignment of full \( S_1 \) and full \( S_2 \)
  - (Can then trace back to construct optimal path to it)
Dynamic Programming for sequence alignment

• **Setting up dynamic programming**
  1. Find ‘matrix’ parameterization
     • Prefix parameterization. Score(S₁[1..i], S₂[1..j]) \( \Rightarrow \) \( M(i,j) \)
     • (i,j) only prefixes vs. (i,j,k,l) all substrings \( \Rightarrow \) simpler 2-d matrix
  2. Make sure sub-problem space is finite! (not exponential)
     • It’s just \( n^2 \), quadratic (which is polynomial, not exponential)
  3. **Traversal order:** sub-results ready when you need them
  4. **Recursion formula:** larger problems = Func(subparts)
     • Need formula for computing \( M[i,j] \) as function of previous results
     • Single increment at a time, only look at \( M[i-1,j], M[i,j-1], M[i-1,j-1] \)
       corresponding to 3 options: gap in \( S_1 \), gap in \( S_2 \), char in both
     • Score in each case depends on gap/match/mismatch penalties
  5. **Remember choice:** \( F() \) typically includes min() or max()
     • Remember which of three cells (top, left, diag) led to maximum
Step 1: Setting up the scoring matrix $M[i,j]$

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>A</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Initialization:**
- Top left: 0

**Update Rule:**
$$M(i,j) = \max \{ \}$$

**Termination:**
- Bottom right
Step 2: Filling in the optimal scores from top left

Initialization:
• Top left: 0

Update Rule:
\[ M(i,j) = \max\{ \]
• \[ M(i-1, j) - 2 \text{ gap} \]
• \[ M(i, j-1) - 2 \text{ gap} \]
• \[ M(i-1, j-1) + 1 \text{ match} \]
\[ \}

Termination:
• Bottom right

Path segment that lead to the optimal choice
Step 3: Trace back pointers to construct alignment

**Initialization:**
- Top left: 0

**Update Rule:**
\[ M(i,j) = \max \{ \]
- \[ M(i-1,j) - 2 \text{ \text{gap}} \]
- \[ M(i,j-1) - 2 \text{ \text{gap}} \]
- \[ M(i-1,j-1) - 1 \text{ \text{mismatch}} \]
- \[ M(i-1,j-1)+1 \text{ \text{match}} \]
\[ \} \]

**Termination:**
- Bottom right

Path segments that lead to locally optimal choices
Path segments that lead to the globally optimal solution
Genome alignment in an excel spreadsheet

https://www.dropbox.com/s/ksh4qfl5eb182p6/Lecture02_DP%20Alignment%20In%20Excel.xlsx?dl=0
Genome alignment in an excel spreadsheet

Local score of matching characters $S_1[i]$ and $S_2[j]$

Is the max alignment score coming from the top ("|"), from the left ("--") or from the diagonal up ("\") (show all of them, cuz we can).

Is the $[i,j]$ part of an optimal path? (i.e. are chars $S_1[i]$ and $S_2[j]$ aligned to each other in an optimal path) (also count number of optimal paths/alignment through $[i,j]$, cuz we can)

Construct the optimal alignment for sequence $S_1$ by adding in characters or gaps to increasingly large suffixes (and arbitrarily choose one path when multiple using nested if's)

Construct the optimal alignment for sequence $S_2$ similarly to $S_1$
What is missing? (5) Returning the actual path!

- We know how to compute the best score
  - Simply the number at the bottom right entry
- But we need to remember where it came from
  - Pointer to the choice we made at each step
- Retrace path through the matrix
  - Need to remember all the pointers

Time needed: $O(m \times n)$
Space needed: $O(m \times n)$
Goal: Sequence Alignment / Dynamic Programming

1. Introduction to sequence alignment
   - Comparative genomics and molecular evolution
   - From Bio to CS: Problem formulation
   - Why it’s hard: Exponential number of alignments

2. Introduction to principles of dynamic programming
   - Computing Fibonacci numbers: Top-down vs. bottom-up
   - Repeated sub-problems, ordering compute, table lookup
   - DP recipe: (1) Parameterization, (2) sub-problem space, (3) traversal order, (4) recursion formula, (5) trace-back

3. DP for sequence alignment
   - Additive score, building up a solution from smaller parts
   - Prefix matrix: finite subproblems, exponential paths
   - Duality: each entry $\Leftrightarrow$ prefix alignment score; path $\Leftrightarrow$ alignment

4. Advanced topics: Dynamic Programming variants
   - Linear-time bounded DP (heuristic). Linear-space DP. Gaps
   - Importance of parameterization: 2-D vs. 4-D decomposition
If time permits…

(4) Extensions to basic DP solution
Bounded Dynamic Programming

**Initialization:**

\[ F(i,0), F(0,j) \text{ undefined for } i, j > k \]

**Iteration:**

For \( i = 1 \ldots M \)

For \( j = \max(1, i-k) \ldots \min(N, i+k) \)

\[ F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i, j-1) - d, \text{ if } j > i-k(N) \\ F(i-1, j) - d, \text{ if } j < i+k(N) \end{cases} \]

**Termination:**

Same

Slides credit: Serafim Batzoglou
Can we do better than $O(n^2)$ in the general case?

- Reduced Orthogonal Vectors to PATTERN
- Reduced PATTERN to EDIT DISTANCE
- Proved EDIT DISTANCE is a SETH-hard problem

In this paper we provide evidence that the (near-)quadratic running time bounds known for this problem might, in fact, be tight. Specifically, we show that if the edit distance can be computed in time $O(n^{2-\delta})$ for some constant $\delta > 0$, then the satisfiability of conjunctive normal form (CNF) formulas with $N$ variables and $M$ clauses can be solved in time $M^{O(1)2^{(1-\epsilon)N}}$ for a constant $\epsilon > 0$. The latter result would violate the Strong Exponential Time Hypothesis (SETH), introduced in [IP01, IPZ01], which postulates that such algorithms do not exist. The rationale behind this hypothesis is that, despite decades of research on fast algorithms for satisfiability and related problems, no algorithm was yet shown to run in time faster than $2^{N(1-o(1))}$. Because of this state of affairs, SETH has served as the basis for proving conditional lower bounds for several important computational problems, including k-Dominating Set [PW10], the diameter of sparse graphs [RW13], local alignment [AWW14], dynamic connectivity problems [AW14], and the Frechet distance computation [Bri14]. Our paper builds on these works, identifying a new important member of the class of “SETH-hard” problems.

- Faster edit dist. algorithm probably not a good term project
Linear space alignment

It is easy to compute $F(M, N)$ in linear space

Allocate (column[1])
Allocate (column[2])

For $i = 1 \ldots M$
    If $i > 1$, then:
        Free (column[$i-2$])
        Allocate (column[$i$])

For $j = 1 \ldots N$
    $F(i, j) = \ldots$

What about the pointers?
Finding the best back-pointer for current column

- Now, using 2 columns of space, we can compute for $k = 1 \ldots M$, $F(M/2, k)$, $F^r(M/2, N-k)$

PLUS the backpointers
• Now, we can find $k^*$ maximizing $F(M/2, k) + F_r(M/2, N-k)$
• Also, we can trace the path exiting column $M/2$ from $k^*$
Recursively find midpoint for left & right

- Iterate this procedure to the left and right!
Total time cost of linear-space alignment

Total Time: \( cMN + cMN/2 + cMN/4 + \ldots = 2cMN = O(MN) \)

Total Space: \( O(N) \) for computation,
\( O(N+M) \) to store the optimal alignment
Goal: Sequence Alignment / Dynamic Programming

1. Introduction to sequence alignment
   – Comparative genomics and molecular evolution
   – From Bio to CS: Problem formulation
   – Why it’s hard: Exponential number of alignments

2. Introduction to principles of dynamic programming
   – Computing Fibonacci numbers: Top-down vs. bottom-up
   – Repeated sub-problems, ordering compute, table lookup
   – DP recipe: (1) Parameterization, (2) sub-problem space, (3) traversal order, (4) recursion formula, (5) trace-back

3. DP for sequence alignment
   – Additive score, building up a solution from smaller parts
   – Prefix matrix: finite subproblems, exponential paths
   – Duality: each entry \( \Leftrightarrow \) prefix alignment score; path \( \Leftrightarrow \) alignment

4. Advanced topics: Dynamic Programming variants
   – Linear-time bounded DP (heuristic). Linear-space DP. Gaps
   – Importance of parameterization: 2-D vs. 4-D decomposition
Additional insights

Why the 2-dimentional parameterization worked
Summary

• Dynamic programming
  – Reuse of computation
  – Order sub-problems. Fill table of sub-problem results
  – Read table instead of repeating work (ex: Fibonacci)

• Sequence alignment
  – Edit distance and scoring functions
  – Dynamic programming matrix
  – Matrix traversal path ⇔ Optimal alignment

• Thursday: Variations on sequence alignment
  – Local/global alignment, affine gaps, algo speed-ups
  – Semi-numerical alignment, hashing, database lookup

• Recitation:
  – Dynamic programming applications
  – Probabilistic derivations of alignment scores
Goal: Sequence Alignment / Dynamic Programming

1. Introduction to sequence alignment
   - Comparative genomics and molecular evolution
   - From Bio to CS: Problem formulation
   - Why it’s hard: Exponential number of alignments

2. Introduction to principles of dynamic programming
   - Computing Fibonacci numbers: Top-down vs. bottom-up
   - Repeated sub-problems, ordering compute, table lookup
   - DP recipe: (1) Parameterization, (2) sub-problem space,
     (3) traversal order, (4) recursion formula, (5) trace-back

3. DP for sequence alignment
   - Additive score, building up a solution from smaller parts
   - Prefix matrix: finite subproblems, exponential paths
   - Duality: each entry $\leftrightarrow$ prefix alignment score; path $\leftrightarrow$ alignment

4. Advanced topics: Dynamic Programming variants
   - Linear-time bounded DP (heuristic). Linear-space DP.
   - Gaps
   - Importance of parameterization: 2-D vs. 4-D decomposition