Determining a Peptide's Sequence

- From last time we learned that we can't always use DNA to resolve peptide/protein sequences
- What else can we do?
  - Extract and purify a pure sample of the peptide/protein
  - Try to resolve the peptide sequence by analyzing this sample
- Today's approach
  - Randomly fracture the peptide
  - Assemble an answer from the pieces
Molecular Weights are the Puzzle Pieces
Structure of a Peptide Chain

- Peptides are chains of amino acids that are joined by *peptide* bonds.
- These bonds reduce the weight of each amino acid by one $\text{H}_2\text{O}$ molecule.
- The result is called a *residue*.
- A Mass Spectrograph can precisely measure the molecular weight (and charge and abundance) of any peptide chain.
- Since the molecular weight of each of the possible 20 residues is known precisely, one can ask the question, which combination of residues would give a particular weight?
- The problem is ambiguous for the entire molecule.
  - Consider all permutations of 'PIT':
    - 'PIT', 'PTI', 'ITP', 'IPT', 'TPI', and 'TIP' all weigh the same.
  - But they differ in their 2-peptide fragments:
    - 'PTI' breaks into 'PI' and 'IT', while
    - 'PTI' breaks into 'PI' and 'TI'.
An Simplified Peptide Weight table

- The actual molecular weight of an amino acid is a real number. This accounts for the relative abundances of atomic isotopes.
- We will use a simplified version that uses only integer molecular weights.
- Example:
  - Molecular weight of Glycine Amino Acid
    \[
    W(C_2H_5NO_2) = 12 \times 2 + 5 \times 1 + 14 + 16 \times 2 = 75
    \]
  - Molecular weight of Glycine Residue (Minus the H_2O lost forming the peptide bond)
    \[
    W(C_2H_5NO_2 - H_2O) = 57
    \]
- We can repeat this for all 20 Amino Acids to get a integer molecular weight table, which I call *Daltons*. 
### Table Definitions

**AminoAcid** = {
  'A': 'Alanine',  'C': 'Cysteine',  'D': 'Aspartic acid',  'E': 'Glutamic acid',
  'F': 'Phenylalanine',  'G': 'Glycine',  'H': 'Histidine',  'I': 'Isoleucine',
  'K': 'Lysine',  'L': 'Leucine',  'M': 'Methionine',  'N': 'Asparagine',
  'P': 'Proline',  'Q': 'Glutamine',  'R': 'Arginine',  'S': 'Serine',
  'T': 'Threonine',  'V': 'Valine',  'W': 'Tryptophan',  'Y': 'Tyrosine',
  '*' : 'STOP'
}

**AminoAbbrev** = {
  'A': 'Ala',  'C': 'Cys',  'D': 'Asp',  'E': 'Glu',
  'F': 'Phe',  'G': 'Gly',  'H': 'His',  'I': 'Ile',
  'K': 'Lys',  'L': 'Leu',  'M': 'Met',  'N': 'Asn',
  'P': 'Pro',  'Q': 'Gln',  'R': 'Arg',  'S': 'Ser',
  'T': 'Thr',  'V': 'Val',  'W': 'Trp',  'Y': 'Tyr',
  '*' : 'STOP'
}

# Here's a new dictionary!

**Daltons** = {
  'A': 71,  'C': 103,  'D': 115,  'E': 129,
  'F': 147,  'G':  57,  'H': 137,  'I': 113,
  'K': 128,  'L': 113,  'M': 131,  'N': 114,
  'P':  97,  'Q': 128,  'R': 156,  'S':  87,
  'T': 101,  'V':  99,  'W': 186,  'Y': 163
}
Some Issues with our Table

- We can't distinguish between Leucine (L) and Isoleucine (I). They both weight 113 d
- Nor can we distinguish Lysine (K) and Glutamine (Q), which weigh 128 d
- For long peptide chains >50, our errors can build up
- In reality, peptides can loose or gain one or more small molecules from their side chains and fractured peptide bonds
  - Gain Hydrogen ions (H, +1 Dalton)
  - Lose Water (H₂O, -18 Daltons)
  - Lose Ammonia (NH₃, -17 Daltons)
- This leads to measurements that vary around the ideal sums we assume
- Regardless of these caveats, let's keep going
The total molecular weight of our target

Tyrocidine B1 = "VKLFPWFNQY"

# The weight of Tyrocidine B1
print sum([Daltons[res] for res in TyrocidineB1])

1322

- Generally, we will assume that the peptide's total molecular weight is known
- We will use it as a terminating condition for many of our algorithms that attempt to reconstruct the measured set of weights
Ideally, what Weights should we get?

- We will make the optimistic assumption that we will fracture our given peptide chain into all of its constituent parts.
- For a 10 peptide chain:
  - 10 single peptides
  - 9, 2-peptide chains
  - 8, 3-peptide chains
  - 7, 4-peptide chains
  - 6, 5-peptide chains
  - 5, 6-peptide chains
  - 4, 7-peptide chains
  - 3, 8-peptide chains
  - 2, 9-peptide chains
  - 1, 10-peptide chain
- This gives an upper bound of $\binom{10}{2} = 45$ molecular weights, but relativity both the peptide chains and their weights may not be unique.
- The collection of all possible sub-peptide molecular weights from a peptide is called the peptide’s *Theoretical Spectrum*. 

8
Code for computing a Theoretical Spectrum

```python
def TheoreticalSpectrum(peptide):
    # Generate every possible fragment of a peptide
    spectrum = set()
    for fragLength in xrange(1,len(peptide)+1):
        for start in xrange(0,len(peptide)-fragLength+1):
            seq = peptide[start:start+fragLength]
            spectrum.add(sum([Daltons[res] for res in seq]))
    return sorted(spectrum)

print TyrocidineB1
spectrum = TheoreticalSpectrum(TyrocidineB1)
print len(spectrum)
print spectrum

VKLFPWFNQY
51
```

• Why are we using a set rather than a list? Notice that we end up returning a list.
Fragments and their Spectrums

```python
peptide = 'TyrocidineB1'
fragList = []

for fragLength in xrange(1, len(peptide)+1):
    for start in xrange(0, len(peptide)-fragLength+1):
        seq = peptide[start:start+fragLength]
        fragList.append((sum([Daltons[res] for res in seq]), seq))

print len(fragList)
N = 0
lastWeight = 0

for weight, frag in sorted(fragList):
    print ('%12s %4d %s' % (frag, weight, ('' if weight == lastWeight) else '' ),)
    N += 1
    if (N % 5 == 0):
        print
    lastWeight = weight
```

| P:  | 97 |
| Q:  | 128* |
| V:  | 99 |
| L:  | 113 |
| N:  | 114 |
| K:  | 128 |
| F:  | 147 |
| Y:  | 163 |
| W:  | 186 |
| KL: | 241 |
| QY: | 291 |
| WF: | 333 |
| VKL: | 340 |
| FN: | 261 |
| PW: | 283 |
| QN: | 242 |
| FP: | 244 |
| LF: | 260 |
| LFP: | 357 |
| KLF: | 388 |
| FNQ: | 389 |
| NQY: | 485 |
| FPW: | 430 |
| PW: | 436* |
| WFN: | 447 |
| KLFP: | 485 |
| VKLF: | 487 |
| LFPW: | 543 |
| PKFNQ: | 544 |
| FNQY: | 552 |
| WFNQ: | 575 |
| FPWF: | 577 |
| VKLF: | 584 |
| KLFSP: | 671 |
| PKFNQ: | 672 |
| LFPWF: | 690 |
| FPWF: | 691 |
| WFNQ: | 738 |
| VKLFP: | 779 |
| LFPFNQ: | 804 |
| KLFPWF: | 818 |
| FPWF: | 819 |
| PKFNQ: | 835 |
| VKLFPWF: | 917 |
| KLFPFNQ: | 932 |
| LFPWFNQ: | 932* |
| FPWFNQ: | 982 |
| VKLFPWFNQ: | 1031 |
| KLFSPNQ: | 1060 |
| LFPWFNQY: | 1095 |
| VKLFPWFNQ: | 1159 |
| KLFSPNQY: | 1223 |
| VKLFPWFNQY: | 1322 |
peptide = 'PLAY'
spectrum = TheoreticalSpectrum(peptide)
print len(spectrum), spectrum

fragList = []
for fragLength in xrange(1,len(peptide)+1):
    for start in xrange(0,len(peptide)-fragLength+1):
        seq = peptide[start:start+fragLength]
        fragList.append((sum([Daltons[res] for res in seq]), seq))

print len(fragList)
N = 0
lastWeight = 0
for weight, frag in sorted(fragList):
    print "%12s: %40s" % (frag, weight, "** if (weight == lastWeight) else " "),
    N += 1
    if (N % 5 == 0):
        print
    lastWeight = weight

10 [71, 97, 113, 163, 184, 219, 234, 281, 347, 444]
10
Can we Invert the Process of creating a Spectrum?

- In essence, the problem of inferring a peptide chain from the set of mass values reported by a Mass Spectrometer is the inverse of the code we just wrote

  **Easy Problem:** Peptide Sequence $\rightarrow$ Spectrum
  **Hard Problem:** Peptide Sequence $\leftarrow$ Spectrum

- Why is computing a spectrum from a peptide sequence easy? $O(N^2)$?
- Why is computing a peptide sequence from a spectrum hard? $O(?)$
How might you approach this problem?

- Can you think of a Brute-Force way of solving this problem?
- Here's one:
  1. For every peptide sequence with the target peptide's molecular weight
  2. Compute the sequence's Theoretical Spectrum
  3. If it matches the one given, report this peptide as a possible solution

- Which step in this algorithm is the hard part?
Impressions?

- Not so bad for a first attempt, but how will it perform for longer peptides?
- We are getting the expected answer as well as answers with the indistinguishable amino acids substituted
- We are also getting the sequence reversed? Is this a surprise?
- We could code around this, but for today we'll just include the reversed peptide chain as a possible answer

Could we do better?

- The brute force method does not make good use of the spectrum it is given
- It only ever considers the largest value from this table
- How might we make use of the other values?
Impressions?

- Not so bad for a first attempt, but how will it perform for longer peptides?
- We are getting the expected answer as well as answers with the indistinguishable amino acids substituted
- We are also getting the sequence reversed? Is this a surprise?
- We could code around this, but for today we'll just include the reversed peptide chain as a possible answer

Could we do better?

- The brute force method does not make good use of the spectrum it is given
- It only ever considers the largest value from this table
- How might we make use of the other values?
Improving on Brute Force

- We could extend our prefix using only residues that appear in our spectrum
- The weight of every new prefix that we consider should also be in our spectrum

Actual fragments: P L A Y PL LA AY PLA LAY PLAY

Growing and Checking prefixes:

<table>
<thead>
<tr>
<th>A</th>
<th>I</th>
<th>L</th>
<th>P</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI = LA</td>
<td>IA = LA</td>
<td>LA = LA</td>
<td>PI = PL</td>
<td>YA = AY</td>
</tr>
<tr>
<td>ATP = PLA</td>
<td>IAP = PLA</td>
<td>LAP = PLA</td>
<td>PIA = PLA</td>
<td>YAI = LAY</td>
</tr>
<tr>
<td>AIPY = PLAY</td>
<td>IAPY = PLAY</td>
<td>LAPY = PLAY</td>
<td>PIAY = PLAY</td>
<td>YAIP = PLAY</td>
</tr>
<tr>
<td>AY = LAY</td>
<td>IAY = LAY</td>
<td>LAY = LAY</td>
<td>YAL = LAY</td>
<td></td>
</tr>
<tr>
<td>AYP = PLAY</td>
<td>IAYP = PLAY</td>
<td>LAYP = PLAY</td>
<td>YALP = PLAY</td>
<td></td>
</tr>
<tr>
<td>AL = LA</td>
<td>IP = PL</td>
<td>LP = PL</td>
<td>PL = PL</td>
<td></td>
</tr>
<tr>
<td>ALP = PLA</td>
<td>IPA = PLA</td>
<td>LPA = PLA</td>
<td>PLA = PLA</td>
<td></td>
</tr>
<tr>
<td>ALPY = PLAY</td>
<td>IPAY = PLAY</td>
<td>LPAY = PLAY</td>
<td>PLAY = PLAY</td>
<td></td>
</tr>
<tr>
<td>AY = AY</td>
<td>AYI = LAY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AYIP = PLAY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AYL = LAY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AYLP = PLAY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
def ImprovedPossiblePeptide(spectrum, prefix=''):  
    global peptideList 
    if (len(prefix) == 0): 
        peptideList = [] 
        current = sum([Daltons[res] for res in prefix]) 
        target = max(spectrum) 
        if (current == target):  
            peptideList.append(prefix) 
        elif (current < target): 
            for residue in Daltons.iterkeys(): 
                # make sure that this residue appears in our spectrum 
                if (Daltons[residue] not in spectrum) 
                    continue 
                # make sure that adding this residue to the sequence we have so far appears in our spectrum 
                extend = prefix + residue 
                if (sum([Daltons[res] for res in extend]) not in spectrum): 
                    continue 
                ImprovedPossiblePeptide(spectrum, extend) 

spectrum = TheoreticalSpectrum('PLAY') 
%time ImprovedPossiblePeptide(spectrum) 
print len(peptideList), "PLAY" in peptideList 
print peptideList 
%time matches = TestPeptides(peptideList, spectrum) 
print matches, "PLAY" in matches 

CPU times: user 0 ns, sys: 1 ms, total: 1 ms 
Wall time: 968 µs 
16 True 
CPU times: user 0 ns, sys: 0 ns, total: 0 ns 
Wall time: 444 µs 
["PIAY", "PLAY", "YAIP", "YALP"] True
Impact of a small change

- Provides a HUGE performance difference
- Yet another example of Branch-and-Bound
- We improved both the enumeration and verification phases, but the difference was much more significant in the enumeration step

```python
for peptide in peptideList:
    print peptide,

PLAY PLAY YAIP YALP

TheoreticalSpectrum('PLAY')
[71, 97, 113, 163, 184, 210, 234, 281, 347, 444]

TheoreticalSpectrum('LAPY')
[71, 97, 113, 163, 168, 184, 260, 281, 331, 444]

print sum([Daltons[res] for res in 'AP'])  # Suffix of 'LAP' prefix
print sum([Daltons[res] for res in 'APY'])   # Suffix of 'LAPY'
print sum([Daltons[res] for res in 'Py'])    # Suffix of 'LAPY'
168
331
260
```

- There are still differences in the spectrums, yet every prefix was in the spectrum when we added it. What are we missing?
- Suffixes!
We can do Even Better

- All suffixes of each prefix that we consider should also be in our spectrum

```python
def UltimatePossiblePeptide(spectrum, prefix=''):  
  global peptidelist
  if (len(prefix) == 0):
    peptidelist = []
    current = sum([Daltons[res] for res in prefix])
    target = max(spectrum)
  if (current == target):
    peptidelist.append(prefix)
  elif (current < target):
    for residue in Daltons.iterkeys():
      extend = prefix + residue
      # test every new suffix created by adding this new residue
      # Note: this includes the residue itself as the length 1 suffix
      suffix = [extend[i:] for i in xrange(len(extend))]
      for fragment in suffix:
        if (sum([Daltons[res] for res in fragment]) not in spectrum):
          break
      else:
        UltimatePossiblePeptide(spectrum, extend)

spectrum = TheoreticalSpectrum('PLAY')
%time UltimatePossiblePeptide(spectrum)
print len(peptidelist), peptidelist, "PLAY" in peptidelist
%time matches = TestPeptides(peptidelist, spectrum)
print matches, "PLAY" in matches

CPU times: user 3 ms, sys: 0 ns, total: 3 ms
Wall time: 1.93 ms
4 ['PIAY', 'PLAY', 'YAIP', 'YALP'] True
CPU times: user 1 ms, sys: 0 ns, total: 1 ms
Wall time: 186 µs
['PIAY', 'PLAY', 'YAIP', 'YALP'] True
```

- A little slower, but our list is pruned significantly
- All of theses have identical spectrums
Now let's return to our *real* peptide

```python
spectrum = TheoreticalSpectrum(TyrocidineB1)
print len(peptidelist)
print TyrocidineB1 in peptidelist
print matches = TestPeptides(peptidelist, spectrum)
print len(matches)
print TyrocidineB1 in matches

CPU times: user 39 ms, sys: 9 ms, total: 48 ms
Wall time: 40 ms
16
True
CPU times: user 2 ms, sys: 0 ns, total: 2 ms
Wall time: 1.75 ms
16
True
```

```python
for i, peptide in enumerate(peptidelist):
    if (i % 4 == 3):
        print

VKIFPFKNKY VKIFPFKNQY VKLFIFKNKY VKLFIFKNQY
VQIFPFKNKY VQIFPFKNQY VQLFIFKNKY VQLFIFKNQY
YKNFIFPIKV YKNFIFPIQV YKNFIFPLKV YKNFIFPLQV
YQNFIFPIKV YQNFIFPIQV YQNFIFPLKV YQNFIFPLQV
```

All of these peptides give also give us our desired spectrum
Great, but our assumptions are a little Naïve

- In reality, Mass Spectrometers don’t report the Theoretical Spectrum of a peptide
- Instead they report a measured or *Experimental Spectrum*
- This spectrum might *miss* some fragments
- It might also report *false* fragments
  - From Contaminants
  - New peptides formed by unintended reactions between fragments
- The result is that some of the masses that appear may be misleading, and some that we want might be missing
- We need to develop algorithms for reporting candidate protein sequences that are robust to noise
## Example experimental spectrum for Tyrocidine B1

<table>
<thead>
<tr>
<th>Mass</th>
<th>Mass</th>
<th>Mass</th>
<th>Mass</th>
<th>Mass</th>
<th>Mass</th>
</tr>
</thead>
<tbody>
<tr>
<td>97</td>
<td>99</td>
<td>113</td>
<td>114</td>
<td>128</td>
<td>147</td>
</tr>
<tr>
<td>186</td>
<td>200</td>
<td>227</td>
<td>241</td>
<td>242</td>
<td>244</td>
</tr>
<tr>
<td>261</td>
<td>283</td>
<td>291</td>
<td>333</td>
<td>340</td>
<td>357</td>
</tr>
<tr>
<td>389</td>
<td>405</td>
<td>430</td>
<td>447</td>
<td>457</td>
<td>485</td>
</tr>
<tr>
<td>543</td>
<td>544</td>
<td>552</td>
<td>575</td>
<td>577</td>
<td>584</td>
</tr>
<tr>
<td>671</td>
<td>672</td>
<td>690</td>
<td>691</td>
<td>731</td>
<td>738</td>
</tr>
<tr>
<td>804</td>
<td>818</td>
<td>819</td>
<td>835</td>
<td>906</td>
<td>917</td>
</tr>
<tr>
<td>982</td>
<td>1031</td>
<td>1060</td>
<td>1095</td>
<td>1159</td>
<td>1223</td>
</tr>
</tbody>
</table>

**False Masses:** present in the experimental spectrum, but not in the theoretical spectrum

**Missing Masses:** present in the theoretical spectrum, but not in the experimental spectrum
Example experimental spectrum for Tyrocidine B1

97, 99, 113, 128, 147, 163,
186, 200, 227, 241, 242, 244, 260,
261, 283, 291, 333, 340, 357,
405, 430, 447, 457, 487,
543, 544, 552, 575, 577, 584, 659,
671, 672, 690, 691, 731, 738, 770,
804, 818, 819, 835, 906, 917, 932,
982, 1031, 1095, 1159, 1322

**False Masses:** We don't know which these are

**Missing Masses:** And these values don't appear
An aside: Faking an Experimental Spectrum

```python
# generate a synthetic experimental spectrum with 10% Error
import random
random.seed(1961)
spectrum = TheoreticalSpectrum(TyrocidineB1)

# Pick around -10% at random to remove
missingMass = random.sample(spectrum[:-1], 6)  # keep largest mass
print "Missing Masses = ", missingMass

# Add back another -10% of false, but actual, peptide masses
falseMass = []
for i in xrange(5):
    fragment = ''.join(random.sample(Daltons.keys(), random.randint(2, len(TyrocidineB1)-2)))
    weight = sum([Daltons[residue] for residue in fragment])
    falseMass.append(weight)
print "False Masses = ", falseMass
expersperimentalSpectrum = sorted(set([mass for mass in spectrum if mass not in missingMass] + falseMass))

Missing Masses = [1150, 114, 601, 186, 819, 357]
False Masses = [457, 200, 731, 906, 659]

print experimentalSpectrum

A Golf Tournament Analogy

- After the first couple of rounds of a major golf tournament a cut is made of all golfers who are so far back from the leader that it is deemed they are unlikely to ever finish in the money.
- These cut golfers are removed from further consideration.
- This choice is heuristic:
  - It is possible that a player just below the cut could have two exceptional rounds, but that is considered unlikely.
- What is the equivalent of a score in our peptide finding problem?
  - The number of matching masses in the candidate peptide's Theoretical Spectrum and the Experimental Spectrum.
  - Normalized score, why?
  - $\frac{\text{len}(\text{ Intersection of candidate and experimental spectrums})}{\text{len}(\text{union of candidate and experimental spectrums})}$
  - Jaccard Index for sets.
- In our peptide golf game a round will be considered a one peptide extension of a active set of player peptides.
- We will do cuts on every round, keeping to top 5% of finishers or the top 5 players, which ever is more.
- Why 5%? It is arbitrary, but on each round we will extend the current set of players by one of 20 amino acids, thus increasing the number of peptides by a factor of 20, so reducing by 5% leaves the poolsize relatively stable.
def leaderboardFindPeptide(noisySpectrum, cutThreshold=0.85):
    # Golf Tournament Heuristic
    spectrum = set(noisySpectrum)
    target = max(noisySpectrum)
    players = [''.join(peptide) for peptide in itertools.product(Daltons.keys(), repeat=2)]
    round = 1
    currentLeader = [0.0, '']
    while True:
        print "\%d Players in round \%d [\%5.4f]" % (len(players), round, currentLeader[0])
        leaderboard = []
        for prefix in players:
            testSpectrum = set(TheoreticalSpectrum(prefix))
            totalWeight = max(testSpectrum)
            score = len(spectrum & testSpectrum)/float(len(spectrum | testSpectrum))
            if (score > currentLeader[0]):
                currentLeader = [score, prefix]
            elif (score == currentLeader[0]):
                currentLeader += [prefix]
            if (totalWeight < target):
                leaderboard.append((score, prefix))
        remaining = len(leaderboard)
        if (remaining == 0):
            print "Done, no sequences can be extended"
            break
        leaderboard.sort(reverse=True)
        # Prune the larger of the top 5% or the top 5 players
        cut = leaderboard[max(min(5, remaining-1) * int(remaining * cutThreshold))][0]
        players = [p for s, p in leaderboard if s >= cut for r in Daltons.iterkeys()]
        round += 1
        return currentLeader

spectrum = TheoreticalSpectrum(TyrocidineB1)
experimentalSpectrum = [mass for mass in spectrum if mass not in missingMass] + falseMass
%time winners = leaderboardFindPeptide(experimentalSpectrum)
print winners
print len(winners) - 1, "Candidate residues with", winners[0], 'matches'
print TyrocidineB1, TyrocidineB1 in winners

480 Players in round 1 [0.0000]
480 Players in round 2 [0.0000]
1280 Players in round 3 [0.1280]
1560 Players in round 4 [0.2480]
2000 Players in round 5 [0.2745]
2600 Players in round 6 [0.3654]
3328 Players in round 7 [0.4615]
Next Time

- This method works well, but it relies on heuristics, and thus might miss the best answer
- Our methods are still making a lot of simplifying assumptions
- Relying on exact matches might mislead us
- We will continue to explore ways of assembling peptide sequences from a given experimental spectrum