Other Combinatorial Structures

- $N(n,p)$ subset of integers \{1,2, ..., n\}
- where integer is selected with probability p.
- For example, $N(365, \frac{1}{10})$ has an expected value 36.5

Sudoku (http://en.wikipedia.org/wiki/Sudoku)

- We can generate Sudoku problems by creating a full grid and randomly removing entries
- In Sudoku books, they grade puzzles based on difficulty (easy, medium, hard)
  - We can create a set of simple rules for each of these difficulty levels
  - If a puzzle can be solved with the simple rules than we can say it’s easy
- There is a threshold (grows with $n$) for how many items you need to remove to make one of these puzzles of a given difficulty level.

CNF $f(x_1, x_2, ..., x_n) = (x_1 + x_2 + x_3)(x_4 + x_5 + x_6)$

- n variables
- k literals per clause
- cn clauses

Question: Is there an assignment to the variables for which the formula is true?

- As you increase the number of clauses the function becomes more difficult to satisfy. Those with very few clauses are very easy to satisfy.
- Again, there is a threshold for the number of clauses versus how difficult the problem becomes.

Question: How do we generate test data for an algorithm to solve these types of problems?

- We find an upper and lower bound on the threshold of the number of clauses.

First we will find an upper bound, $r_k$, on the threshold.

For a given assignment a random clause is satisfied with probability $1 - \frac{1}{2^k}$

With $cn$ independent clauses, the probability all are satisfied is $\left(1 - \frac{1}{2^k}\right)^{cn}$

Since there are $2^n$ assignments:

$E(\# \text{ of satisfying assignments}) = 2^n \left(1 - \frac{1}{2^k}\right)^{cn}$

If $c = 2^k \ln(2)$,
\[ E = 2^n \left( 1 - \frac{1}{2^k} \right)^{2^{k \ln(2)}} = 2^n e^{-n \ln(2)} = 2^n 2^{-n} = 1 \]

If \( c > e^{k \ln(2)} \) then \( E(#) \to 0 \)
Almost surely no satisfying assignments

If \( c < e^{k \ln(2)} \) then \( E(#) \to \infty \)
However, there is a problem with the second moment argument since the variables are not independent.

**Unit Clause Heuristic** - we can prove the lower bound \( r_k \geq \frac{2^k}{k} \)
\[ \frac{2^k}{k} \leq \text{threshold} \leq 2^{k \ln(2)} \]

For \( k=3, \quad 2.667 \leq \text{threshold} \leq 5.54 \)

- If \( \exists \) clause with one literal, then set it true
- Delete all clauses with the literal
- Delete complement of literal wherever it occurs
- Otherwise pick literal at random and set it to true

As long a you have less than \( \frac{2^k}{k} \) this algorithm will find a solution with high probability.

Next Topic (Components in graphs)
Example from hw:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>...</th>
<th>1436</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>179</td>
<td>50</td>
<td>25</td>
<td>14</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

We'd like to prove that we always get a large component given enough edges.
Note that:
- When \( p=0 \) we get isolated vertices
- As we increase \( p \) we start getting small trees \( \leq \text{log}(n) \)
- When we increase \( p \) to \( \frac{1}{n} \) we get cycles
- At \( \frac{1}{n} \), we get components of size \( n^{2/3} \)
- Increase \( p > \frac{1}{n} \) get giant component \( cn \)
- As we increase \( p \) greater than \( \frac{1}{n} \), the giant components will swallow up the large components one giant component and isolated vertices.
The graph becomes connected at \( \approx \frac{\ln(n)}{4n} \) multiplied by some constant. (Professor was unsure of the constant in front).

Now let's say we have \( G\left( n, \frac{d}{n} \right) \). We can pick a vertex, and then look at all the vertices connected to it.

Discovered: \( v, v_1, v_2, v_3, v_4 \)
Explored: \( v \)

Note that the size of the set of vertices discovered is initially \( d_i \) where \( i \) is the number of steps. Also, the number of vertices initially explored is size \( i \).

frontier

frontier = discovered - explored
On average, we find $d$ more vertices with each vertex explored but when you get to $\frac{d-1}{d}n$ you only find 1 new vertex for each explored.

Algorithm will either halt and find a small component or run for a while and find the large component.

Question: How do we know there is only one large component? (To be proved)
Next lecture we will prove that there is a gap between $\log(n)$ and $cn$