1 Details about Project:

We need to be thinking about topics for the project. There are some suggestions on the website. The website also has a sign-up sheet for us to find partners – list your name and what you would like to work on. Once we know (or think we know) what we want to do, we should talk to Bernhard to get some more advice and pointers on where to start.

We are not expected to get a publishable result in 2 months. However, we should pick something that has the potential to maybe develop into something publishable. The following are a few sentences about each of the suggestions on the website.

• Finding independent sets. An independent set in a graph is a subset of the vertices such that no two vertices in the set are neighbours. There are two subtypes:
  
  – Maximum independent set. This is the largest possible independent set in the graph.
  – Maximal independent set. This is an independent set that can’t be extended (without taking out any vertex).

The first paper on the course webpage deals with large independent sets in sparse graphs. Shows that taking a random independent set generally gives you a good size. Probabilistic method, old paper.

Another paper on the website for this topic is from SODA2015. Much more recent. Deals with maximal independent sets.

Really simple distributed algorithm that breaks symmetry, SODA2016

Guy’s paper: output is p-complete. Choose random order for placing nodes in the set. It shows you can take \( O(\log^2 n) \) time, in parallel.

• Lovasz Local Lemma. Like a local union bound. We will study this in class a bit later. If you have independent bad events, there is some chance that they won’t happen (this is the probabilistic method). However, if they are dependent in some way, the best thing you can do is use union bound. Lovasz: if events are not all dependent on each other, but each event only depends on a small subset, then you can take a union bound on these small subsets. Then if no local subset kills us, then over all, there is
a chance that they won’t happen. The question that is interesting is how to generate/find these sets and probabilities.

- **Chernoff Bounds.** Spectral graph theory, simple proofs but a bit of more complicated math (matrix Chernoff bounds). Can get really cool sparsification results.

- **Gossip.** We already talked about it in class. Gossip vs sub-modularity. Paper from about 5 years ago. Optimizing over gossip protocols. Imagine we have a market based on gossip, and we want to know how to invest our money at the beginning. Which nodes should we ‘infect’, or ‘invest’ in? Investing function is sub modular. That is, gain goes down when adding extra nodes. We can optimize over sub modular functions and get nice approximation guarantees.

- **Discrepancy minimization.** We have nodes, and a list of subsets of them. We want to colour nodes, say red and blue. We want every set to have balanced colours in expectation. Minimizing discrepancy – that is, the maximum difference between red and blue in the sets. Analyzing, for different types of sets, with different characteristics, how good we can get the discrepancy. There have been some interesting non-constructive results that are very old. Then there have also been algorithms that match these results. Bernhard can point us to more papers (that he didn’t put down on the website). Some of these papers need some more advanced knowledge of math.

- **Hypergraph and vertex connectivity.** Sampling of edges in a graph preserves cuts (we saw this in class). We can try to do this on vertices. We sample vertices randomly. Say we define a vertex cut by the number of nodes we need to delete to disconnect the graph. But randomly sampling vertices doesn’t preserve cuts. There are some interesting results that are relatively recent. Can also ask similar questions about hyper edges.

- **Coding for interactive communication** Error correction for conversations.

- **Matroid Secretary and online matching.** Say we are interviewing people for a secretary position. After interviewing a person, you know how good they are, but you don’t know about the next people. You need to decide right then whether to accept them or reject them, and you can’t take back your decision later. The best strategy is to reject few first ones, then wait until you get someone that’s better than the best you’ve seen, and hire them. We can also think of this in a graph: edges are coming one by one, can choose to take them or not. Each has a weight and you want to end up with a spanning tree, but you don’t know the weight of future edges. How do you choose the edges? Or, another similar problem is matching servers to tasks. Can’t tell the future, and want to have as many tasks done as possible in the end. Turns out randomization helps.
Deterministically, you can get at best $1/2$ by a greedy algorithm. But with randomization you can do better (give each task random priority and choose based on that).

- There are many other great topics!

## 2 Network Coding

### 2.1 Quick Recap

What did we prove last time?

In DAGs, if we have $k$ messages to be sent, and we assume that the min-cut from the source to any sink is greater than $k$, in a large enough field, choosing random linear combinations of the messages achieves the best possible result (delivers all $k$ messages to all sinks) w.h.p.

### 2.2 Today

We will consider different graphs. We leave the DAG setting and go to more sophisticated networks. The algorithm will stay pretty much the same.

The algorithm we had so far:

- We send packets of the form:
  \[
  \gamma_1 \ast (\alpha_1, \alpha_2, \ldots, \alpha_k, \sum_{i=1}^k \alpha_i m_i), \\
  \gamma_2 \ast (\beta_1, \beta_2, \ldots, \beta_k, \sum_{i=1}^k \beta_i m_i)
  \]
  
  That is, we start with a standard basis vector, followed by the message. As we progress, the first $k$ slots will actually list the coefficients that were used.

- Think about gossip protocols. Now we have a general graph. From the source, say we keep the packets of the same form. We now send it out to a random neighbour. As an intermediate node, once i get prompted to send something, i pick a random linear combination of everything i have received so far, and send it to a random neighbour.

- If you receive a packet, store it. If you need to send a packet, randomly combine anything stored.

- Note that network coding is a method of deciding ‘what should i send’, and the protocol applied (in this case, the gossip protocol we learned earlier) decides separately who to send to. Who you talk to and what you say are two completely separate questions. This is topology oblivious – i don’t need to know where my packets came from and where they are going to.

- So we can combine this with previous gossip protocols that we have learned. In each round, we pick a random node and send them our message (assuming a well-connected graph). We’ve seen that this works really well for sending one message. How well does it work for $k$ messages? What are upper and lower bounds for how quickly everyone gets all the messages?

**Upper Bound:**
We know we can do this because this can simply be done with routing. Spread one message at a time, run the gossip algorithm $k$ times. 

**Lower Bound:**

$O(\log n)$. Number of nodes that know everything at most doubles in each round. This is the same lower bound for gossip.

$\Omega(k + \log n)$. In expectation, each node will receive one message at a time. We need to get $k$ messages to all of them. Think of it as taking $\log n$ rounds to get the first message, then afterwards the messages arrive one by one in each round.

We will show that network coding gives us $O(k + \log n)$, matching our lower bound. This will also work for a field of size 2 (with just XORs).

**Historical Note:** First proof showed $O(k \sqrt{\log n} + \log n)$, in a very complicated way.

### 2.2.1 Proof

**Definition 1** A node ‘knows’ $\alpha \in F_q^k$ if it has received/stored a packet with non-perpendicular coefficient $\beta$ (i.e. $<\alpha, \beta> \neq 0$).

**Why does this definition make sense?**

If we’re asking about a certain direction, and every single package I received has been perpendicular to that direction, then I know nothing about it.

**How many packets do I need to receive until I ‘know’ all basis vectors?**

It could be just 1. The all ones vector (the XOR of all packets) has something about each basis vector. So we can think of ‘knowing’ as ‘not being completely ignorant’. It is not enough information to know the entire vector.

**Lemma 2** If $v$ knows $\alpha$ and sends to $u$, then $u$ knows $\alpha$ with probability $\geq 1 - 1/q$.

**Proof** There is exactly 1 coefficient we can pick from $F$ that will null out our knowledge. $\blacksquare$

**Lemma 3** If $v$ knows all $q^k$ $\alpha$’s, then $v$ can decode.

**Proof** Look at the space spanned by all the packets. If they span the entire space, then we are happy. If they don’t, then there is at least one packet that is perpendicular. So we have a lower dimensional space. So we can find one vector to blame it on, and we can’t decode. However, if we can’t find someone to blame, then we can decode. $\blacksquare$

**Lemma 4** Random linear network coding with gossip completes in $O(k + \log n)$ rounds w.h.p.
**Proof**  Pick one $\alpha \in F_q^k$. Initially, only the source knows about it. In the next round, one other node knows $\alpha$ with probability $1 - 1/q$. In the next round, these two nodes contact two random nodes, and now more people will know. So everyone will know $\alpha$ in $\log n$ rounds w.h.p.

$E[T_\alpha] = \log n$ (the expected time until everyone will know $\alpha$ is $\log n$).

$\Pr[T_\alpha > O(k + \log n)] < \exp(-(k + \log n))$

Now we are pretty much done...

$\Pr[T_\alpha > O(k + \log n)] < \exp(-(k + \log n)) < 4^k n^{100}$

Union bound over all $2^k \alpha$. Then with probability $\geq 1 - 2^{-k n - 100}$ all nodes know. ■