The Linear MPC Model

- Having $S = n^{1+\varepsilon}$ makes it very easy to design algorithms that are super-fast.
- Let us reduce S to $\Theta(n)$. We call this the linear MPC model.
- In the linear model, each machine has enough space to store a very limited information about each node in the graph.
- Let us continue the MST example and see how to design algorithms in this model.

The Linear MPC Model

- The techniques needed here are quite involved.
- Consider a sorted order of edges by weight. Let the ordering be e_1, e_2, …, e_m.
- The following observation holds:

  Edge e_i is in the MST if and only if its endpoints are not in the same connected components in the graph with edge set $\{e_1, . . . , e_{i-1}\}$.

- We can use this observation directly to convert the MST problem to the connected components problem.
  - Check the observation for each of e_1 to e_m.
- However, this algorithm is quite wasteful.

The Linear MPC Model

- One improvement is as follows.
- Group the m edges into m/n groups of n edges each.
- Call $E_i = \{e_{(i-1)n+1}, . . . , e_{in}\}$ for i = 1 to m/n.
- Let $H_i = U_{j=1}^{i} E_j$. H_i is the union of the first i E_is for i=1 to m/n.
- Let F_i be the MST (Forest) corresponding to H_i.
- Notice that $|F_i| = O(n)$.
- So, given F_i and E_{i+1}, both of size O(n) edges, one machine can check which edges in E_{i+1} will be in the MST.
- We could compute each F_i in parallel!

The Linear MPC Model

- Call $E_i = \{e_{(i-1)n+1}, . . . , e_{in}\}$ for $i = 1$ to m/n.
- Let $H_i = U_{j=1}^{i} E_j$. H_i is the union of the first i E_is for i=1 to m/n.
- Let F_i be the MST (Forest) corresponding to H_i.
- Notice that $|F_i| = O(n)$.
- So, given F_i and E_{i+1}, both of size O(n) edges, one machine can check which edges in E_{i+1} will be in the MST.
- We could compute each F_i in parallel!
- But, $|H_i|$ is too big to fit in any single machine!!

The Linear MPC Model

- We just wish that computing the F_is takes as few rounds as possible.
- Here is a way to do that.
- We use the technique of graph sketching.
- We then explain how to implement the technique in the linear MPC model to compute connected components in O(1) rounds.

The Linear MPC Model

- Let A be a subset of V. Let C be the cut between A and V \ A.
- Assume for a moment that the cut has only edge crossing it.
- We wish to find the cut edge across A and V \ A using only O(log n) bits of memory.
- We use a bit trick that works as follows.

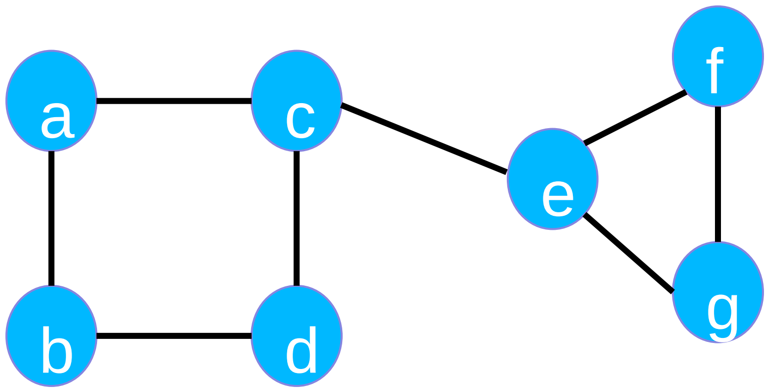
The Linear MPC Model

- Identify each edge by the concatenation the identifiers of its endpoints. For e = uv, id(e) = u o v if id(u) < id(v).
- Locally, each vertex u computes

  $XOR_u = \bigoplus_{e \in E, e \text{ has u as an end point}} id(e)$.
- Now, consider the XOR of all the vertices in A, $XOR_A = \bigoplus_{u \in A} XOR_u$.
- In XOR_A, an edge e = uv with both end points A appears in both XOR_u and XOR_v.
- On the other hand, the edge e = uv with u in A and v in V \ A, appears only once (in XOR_u).
- So, the result of XOR_A = id(e) thereby allowing us to identify the cut edge.

The Linear MPC Model

- Example



Edge	End points	ID
e1	a, c	000010
e2	a, b	000001
e3	b, d	001011
e4	c,d	010011
e5	e, f	100101
e6	f, g	101110
e7	e, g	100110
e8	c, e	010100

Node	Edges	XOR_v
a	e1, e2	000011
b	e2, e3	001010
c	e1, e4, e8	000101
d	e3, e4	011000
e	e5, e7, e8	100111
f	e5, e6	001011
g	e6, e7	111000

XOR_A = XOR(000011, 001010, 000101, 011000) = 010100 = ID(e8)

The Linear MPC Model

- We need to extend the above idea for the case where the cut has multiple cross edges.
- If the cut has more than one edge, then all such edges appear only once in XOR_A.
- The resulting XOR_A may actually end up being the id of some edge that is not a cut edge!
- So, we need a couple more techniques.

The Linear MPC Model

- Our first technique is to replace the id of vertices with $O(\log n)$ bit random strings of $\{0, 1\}$.
- With this, Pr(There exists an edge e, $XOR_A = id(e)) <= \Sigma_e \Pr(XOR_A = id(e)) <= {}^nC_2 \times (½)^{12\log n} = 1/n^{10}$.
- This low probability helps us tide over the trouble of using actual vertex identifiers.

The Linear MPC Model

- For our second technique, we do the following.
- We do not know the number of edges crossing the cut.
- But, the number lies between 0 to n.
- So, we will try all possible estimates for the above number. All in parallel!
- The estimate is good if the actual number of edges crossing the cut is within ½ to 2 of the estimated number.
- In essence, we try estimates such as {1, 2, 4,…, n}.

The Linear MPC Model

- Let k be the actual number of cut edges.
- Run each estimate k' in [1, 2, 4, …, n] in parallel.
- With a given k', we mark edges with a probability of 1/k'.
- Consider the k' such that k'/2 <= k <= k'
- With a marking probability of 1/k', the expected number of cut edges marked is 1.
- We can reuse the case of identifying one cut edge if we get to mark just one of the k cut edges and no other edges are marked.
- Let us calculate the probability for the above event.

The Linear MPC Model

- Let S be the set of cut edges marked.
- $\Pr(|S| = 1) = k \cdot (1/k') (1-1/k')^{(k-1)}$

  $>= (k'/2) \cdot (1/k') (1-1/k')^{(k-1)}$.

  $>= (1/2) \cdot (1/4)$ as $1 - x >= 4^{-x}$.

  $>= 1/8$.

The Linear MPC Model

- A few more nitty-gritty details are needed beyond the two techniques.
- In particular, we will need multiple independent runs with each k’.
- With a success probability of at least 1/8, Chernoff bounds tell us that O(log n) runs will help us.
- The overall technique is called graph sketching.

The Linear MPC Model

- Since the memory needed per sketch per node is O(log n), the linear MPC model can support the technique.
- Read the details from the posted resources.

- Putting together everything, we get:
- The MST of a graph G can be obtained in O(1) rounds in the linear memory MPC model.

The Sub-linear MPC Model

- In the sub-linear MPC model, the only best known solution for the MST problem so far is to use the Boruvka algorithm.
- Recall that Boruvka’s algorithm is what is used in the GHS algorithm also.
- The round complexity is O(log n).

The Sub-Linear Memory Model

- We will now show how to improve the number of rounds from O(log n) to O(log D) where D is the diameter of the graph.
- There are two techniques that we will use.
- The first technique is called graph exponentiation.
- The second technique is graph labelling.

The Sub-Linear MPC Model

- In this model, recall that each machine has a space of n^ε for some ε, $0 < \varepsilon < 1$.
- Let us assume that each vertex has access to a memory of n^ε.
- Assume also that the graph is indeed a single connected component.
- In other words, we are verifying if the graph is connected.

Graph Exponentiation

- Consider a graph which consists of n/D paths of length D where D <= n^{ε}.
- Within O(log D) MPC rounds, every vertex can gather the entire neighborhood using graph exponentiation as follows.
- In every communication round, each node u informs its neighbors of the nodes contained in N(u).

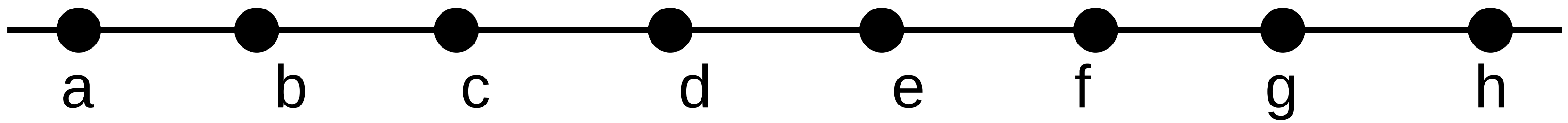
Graph Exponentiation

- Consider a graph which consists of n/D paths of length D where D <= n^{ε}.
- Within O(log D) MPC rounds, every vertex can gather the entire neighborhood using graph exponentiation as follows.
- In every communication round, each node u informs its neighbors of the nodes contained in N(u).
- Then, every node can add the new nodes it learned about in its neighborhood by adding a virtual edge to each such node.

Graph Exponentiation

- In every communication round, each node u informs its neighbors of the nodes contained in N(u).
- Then, every node can add the new nodes it learned about in its neighborhood by adding a virtual edge to each such node.
- This way, in round j of the procedure, node u will be informed about all nodes and edges in its 2^j -hop neighborhood.
- This procedure requires only O(log D) rounds to obtain the D-hop neighborhood.

Example : Graph Exponentiation



Node	Round 0	Round 1	Round 2	Round 3
a	b	a, c		
b	a, c	a, c, d,		
c	b, d	b, d, a, e	a, c, d, e, b, f, g	
d	c, e	c, e, b, f		
e	d, f	d, f, c, g		
f	e, g	d, e, g, h		
g	f, h	f, h, e		
h	g	f, h		

Graph Exponentiation

- In every communication round, each node u informs its neighbors of the nodes contained in N(u).
- Then, every node can add the new nodes it learned about in its neighborhood by adding a virtual edge to each such node. This way, in round j of the procedure, node u will be informed about all nodes and edges in its 2^j -hop neighborhood.
- This procedure requires only O(log D) rounds to obtain the D-hop neighborhood.
- The node can then compute connectivity of the collected subgraph in a single MPC round locally.
- Note that this will require a global memory of O(nD).

Graph Exponentiation

- In round j of the procedure, node u will be informed about all nodes and edges in its 2^j -hop neighborhood.
- This procedure requires only O(log D) rounds to obtain the D-hop neighborhood.
- The node can then compute connectivity of the collected subgraph in a single MPC round locally.
- Note that this will require a global memory of O(nD).

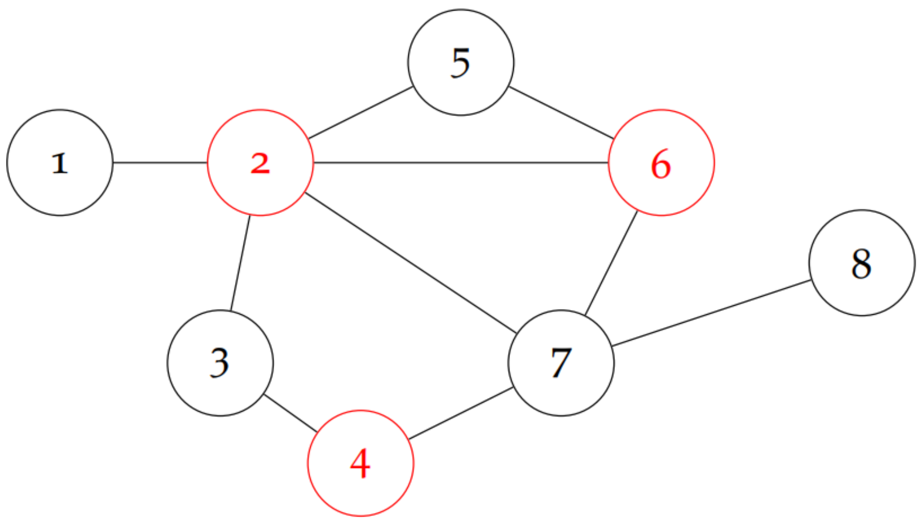
Graph Exponentiation

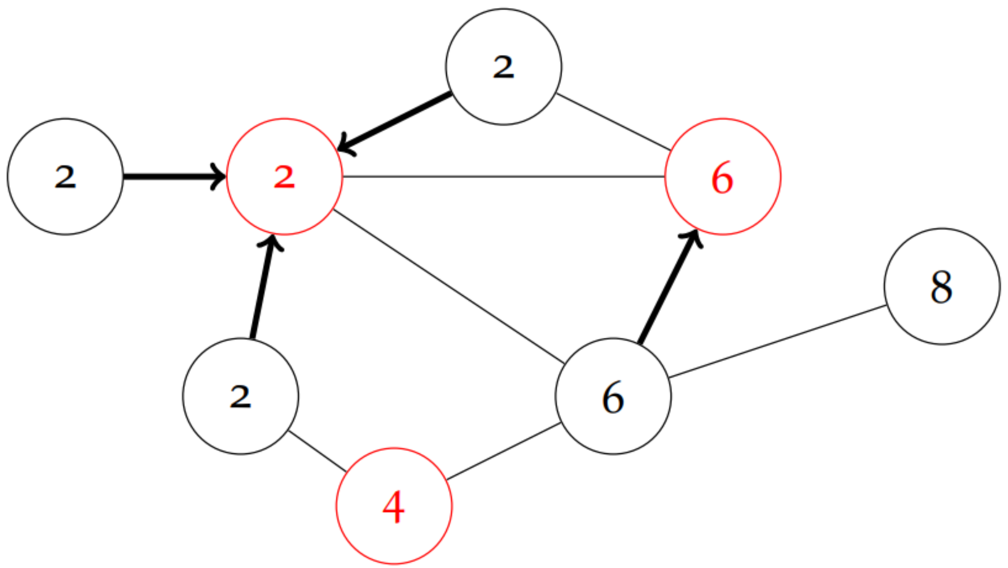
- If the D-hop neighborhood of a vertex fits into memory, then one can gather the entire neighborhood in O(log D) MPC rounds.
- However, for general graphs, one cannot hope to collect the entire D-hop neighborhood of a vertex into a single machine.
  - This D-hop neighborhood can exceed the space in a machine.
- So, this technique works for graphs with low enough degree.

Label Contraction

- The basic idea of label contraction is as follows.
- Mark each vertex independently with probability p = 1/2.
- For each unmarked vertex with a marked neighbor, relabel itself to one of the marked neighbor’s label.
- By treating vertices with the same label as a supernode, we see that label contraction essentially contracts vertices in each round via relabelling.

Label Contraction





Label Contraction

- Claim: In expectation, at most ¾ fraction of the existing labels remain after each iteration.
- Proof uses simple probability as follows.
- After the iteration, the only remaining labels are those of marked vertices.
- Consider a node v with a label l(v).
- Each vertex is marked independently with probability p = 1/2.
- For an unmarked vertex v, the probability that it does not have a marked neighbor is $(1/2)^{deg(v)} <= ½$.

Label Contraction

- The probability that a vertex v is unmarked and has no marked neighbors is at most ¼.
  - Unmarked with a probability of ½ and has no marked neighbors with a probability of ½.
- Hence, Pr[l(v) remains after one iteration]

  = Pr(v is marked, or v is unmarked and has no marked neighbor)

  <= Pr(v is marked) + Pr(v is unmarked and has no marked neighbor)

  <= ½ + ¼ = ¾.
- The expectation follows.

Label Contraction

- Can also use tail inequality to show that the above happens with high probability.
- Thus, in O(log n) rounds, all end points of any edge share the same label.

Combining the Two Techniques

- The exponentiation technique works best when the graph is sparse.
- The label propagation technique works best when the graph is dense.
  - Quick convergence of labels.
- Let us now see how to combine both of these techniques.

Combining the Two Techniques

- The approach proceeds in phases.
- In each phase, the following happens.
  - From each node, perform graph exponentiation until the number of edges leaving the collected set of vertices exceeds $n^{\varepsilon/2}$. Collapse the collected vertices into a supernode.
  - Perform label contraction by marking vertices independently with probability $p = \log n / n^{\varepsilon/2}$.

Combining the Two Technique

- The first step of each phase runs in O(log D) rounds. (or fewer)
- The second step can be analyzed as earlier.
- Notice that after the first step, the degree of each (super)vertex is at least $n^{\varepsilon/2}$.
- So, the probability that a label does not get removed in a step is at most $p + (1-p)(1-p)^{n^{\varepsilon/2}}$.
- The above quantity is in O(p) with $p = \log n/n^{\varepsilon/2}$.
- We can deduce a high probability result so that in each step the number of labels drop by a factor of $O(n^{\varepsilon/2})$.
- The number of phases is therefore $O(1/\varepsilon)$.

Sublinear MPC

- The above steps and techniques assumed the knowledge of the diameter D of the graph.
- This is not easy to obtain.
- One way to circumvent this difficulty is to try for multiple possible D values.
- Try with multiple estimates D' of D such that D' is equal to 2^{2^i} for i= 1, 2, …
- Run the algorithm for all D' in parallel.
- Two guidelines in finding what all possible values to try.
  - Safe: MPC constraints are not violated.
  - Checkable: Quickly check if the solution with some choice is correct.

Summary of the MPC model and MST

- Super-linear memory: O(1/e) rounds, easy technique.
- Linear memory: O(1) rounds, but very complicated and lots of techniques.
- Sub-linear memory: O(log D) rounds.

MPC Model – Other Graph Algorithms

- Wide body of literature wrt MPC algorithms for various graph problems including MIS.
- Many of these results use randomization.
- Recent successes include derandomization as well.
- Variants of the MPC model also under study.
- A good thesis topic with lots of interesting problems yet to be solved.