The plan

- Intro: LOCAL model, synchronicity, Bellman-Ford
- Subdiameter algorithms: independent sets and matchings
- CONGEST model: pipelining, more matching, lower bounds

Distributed Algorithms

- Turing’s vision: multiple heads, multiple tapes, but central control
- Today’s technology: hook up components by communication lines
- Abstraction: network of processors exchanging messages

Some Issues

- Different component speeds, partial failures
- Turned out to be a major headache...
- ... = a rich source for research
- Higher level abstraction: Shared memory
  - Convenient for programmers (?)
  - Focuses on asynchrony and failures

Not our topic!
Our Focus: Communication

The **LOCAL** model
- Connectivity $\equiv$ a graph $G = (V, E)$
- $|V| = n, |E| = m$, diameter $= D$
- Nodes compute, send & receive msgs
- No failures (nodes or links)
- Running time:
  - DEFINE: longest message delay $\equiv$ one time unit
  - Neglect local processing time

The **LOCAL** model: Typical Tasks
- Compute functions of the topology
  - Spanning Trees: Breadth-First (shortest paths), Minimum weight (MST)
  - Maximal Independent Set, Maximal Matching
- Communication tasks
  - Broadcast, gossip, end-to-end
- In general: input/output relation
  - Dynamic version: reactive tasks

The locality of distributed problems

A problem can be solved in $T$ time in **LOCAL** iff the following algorithm works:
- Each node collects full information from its $T$-neighborhood, and computes output

Time is the “radius” of input influence.

**LOCAL** time $= $ problem locality!

- Example: $\Omega(\sqrt{\log n / \log \log n})$ lower bound on time for approx. MIS, Matching [KMW'06]

Example: Broadcast

- A source node has a message all nodes need to learn
  - Input: environment to source at time $0$
- Protocol: When a new message received, send to all neighbors (**flooding**)
- Time: $O(D)$
- Can be used to build a spanning tree:
  - Mark edge delivering first message as parent
Example: Find maximum

- Each node has an input number from $[1, M]$
- Want to find maximum
- Protocol: Whenever a new maximum discovered, announce to all neighbors
- Time to convergence: $O(D)$
- #messages: $O(nm)$
- Message size: $O(\log M)$ bits

Example: What’s $n$?

- Goal: find the number of nodes in the system
- Must assume some symmetry breaking
  - exercise
- Standard assumption: Unique IDs from $[1, N]$
  - Usually assume $N \in n^{O(1)}$
- Solution:
  - Use a broadcast from each node
- Converge in $O(D)$ time
- #messages $O(mn)$

Example: What’s $n$?

Another symmetry breaking tool: randomization

Algorithm:
1. Each node $v$ chooses a value $x_v \sim G\left(\frac{1}{2}\right)$.
2. Find $\max\{x_v \mid v \in V\}$
3. Output $2^{\max}$.
Message length: $O(\log \log n)$.
Can be off by a log $n$ factor. Repeat and report average to decrease variance.

Generic Algorithm

1. Each node broadcasts its input
2. Each node computes locally its output
- Time: $O(D)$, #messages $O(mn)$

Can do better!
1. Build a single tree (how?)
2. Apply algorithm in tree
- #messages $O(n^2)$
- Time?
Asynchrony gives trouble

• A tree may grow quickly in a skewed way...

...But when used for the second time, we may pay for the skew!

The Bellman-Ford Algorithm

• Goal: Given a root, construct a shortest-paths tree
• Protocol:
  – every node $v$ maintains a variable $d_v$
  – Root $r$ has $d_r = 0$ always
  – Non-root $v$ sets $d_v = \min\{d_u + 1: (u, v) \in E\}$
• Can show: stabilizes in time $O(D)$
B-F: Trouble With Asynchrony

Convergence in $O(D)$ time, but strange things can happen...

- Schedule $S_0$: empty schedule
- Schedule $S_{i+1}$:
  1. Allow one message: from $n - i - 1$ to $n - i$ on edge of weight $2^i$ (incoming value: $d_{n-i-1} + 2^i$)
  2. Apply $S_i$ nodes to nodes $n - i, \ldots, n$
  3. Allow another message: from $n - i - 1$ to $n - i$ on edge of weight 0 (incoming value: $d_{n-i-1}$)
  4. Apply $S_i$ nodes to nodes $n - i, \ldots, n$

Synchronous model

If all processors and links run in the same speed:
Execution consists of rounds. In each round:
1. Receive messages from previous round
   - Round 1: receive inputs
2. Do local computation
3. Send messages
Avoid skewed evolution!

Synchronous BFS tree construction

- Protocol: When first message received, mark origin as parent and send to all neighbors
  - Input: environment to source at time 0
  - Break ties arbitrarily
- Natural uniform “ball growing” around origin
- Time: $O(D)$
Synchronizer

- Can emulate synchronous environment on top of asynchronous one
- Abstraction: consistent pulse at all nodes

How:
- send a message to each neighbor in each round (send null message if nothing to send)
- Emit pulse when received messages from all neighbors

Therefore:

- Asynchronous networks are interesting only if there may be faults
  - Or when we care about messages
- We henceforth assume synchronous networks...
  - But we need to account for messages!

Generic Synchronous Algorithm

- **Any** i/o-relation solvable in diameter time:
  1. Construct a BFS tree
     (need IDs/randomization to choose root: Leader Election!)
  2. Send all inputs to root (“convergecast”)
  3. Root computes all outputs, sends back (“broadcast”)
- Ridiculous? That’s the client-server model!
  - Bread-and-butter distributed computing in the 70’s-90’s, and beyond...
- Interesting? Theoretically: sub-diameter upper and lower bounds

Subdiameter algorithms
Independent Sets

- Independent set (IS): a set of nodes, no neighbors
- Maximum: terribly hard
- Maximal: cannot be extended
  - Can be MUCH smaller than maximum IS
- Trivial sequentially!
  - Linear time
- Can this parallelized?

Turan’s theorem: There always exists an IS of size $n/(\bar{d} + 1)$. [$\bar{d} = 2m/n$ is the average degree]

Proof: By the probabilistic method.
Assign labels to nodes by a random permutation.
Let IS be the nodes whose label is a local minimum.

Lemma: $E[|IS|] \geq \frac{n}{\bar{d}+1} \cdot \bullet$

Distributed MIS algorithm

- Each node $v$ chooses a random label $\ell(v)$
  - Say, in $[1, N^{2+c}]$ to avoid collisions w.h.p.
- Local minima enter MIS, neighbors eliminated
- Repeat.

Claim: In expectation, at least half of the edges are eliminated in a round.
Proof of Claim

Say node $v$ is killed by neighbor $u$ if $\ell(u)$ is smallest in $N(v) \cup N(u)$.

An edge $(v, w)$ is killed by node $u$ if $v$ is killed by $u$.

Observation: An edge $(v, w)$ can be killed by at most two nodes: the node with minimal label in $N(v)$ and the node with minimal label in $N(w)$.

Proof of Claim (cont.)

$\Pr[v \text{ killed by } u \mid (u, v) \in E] \geq \frac{1}{d(v) + d(u)}$.

Hence: $E[\# \text{edges killed}] \geq \frac{1}{2} \sum_{u} E[\# \text{edges killed by } u]$

$\geq \frac{1}{2} \sum_{v \in V} d(v) \sum_{u \in N(v)} \frac{1}{d(u) + d(v)}$

$= \frac{1}{2} \sum_{(u, v) \in E} \left( \frac{d(v)}{d(u) + d(v)} + \frac{d(u)}{d(u) + d(v)} \right)$

$= \frac{1}{2} |E|$.

Definitions

• Input: Graph $G=(V, E)$, with weights $w : E \rightarrow \mathbb{R}^+$
• A matching: a set of disjoint edges
• Maximum cardinality matching (MCM)
• Maximum weighted matching (MWM)
Application: Switch Scheduling

Goal: move packets from inputs to their outputs

At each time step, fabric can forward
- one packet from each input
- one packet to each output
• To maximize throughput, find MCM!

Note: Bipartite Graphs

• In many applications, nodes are partitioned into two subsets (input/output, boys/girls)
• Bipartite graphs: $G = (V_1 \cup V_2, E)$ where $V_1 \cap V_2 = \emptyset$ and $E \subseteq V_1 \times V_2$
• Matching is simpler in this case
  – Bipartite MCM: max flow
  – Bipartite MWM: min cost flow

Distributed Matching

Clearly, MCM must take diameter time!
• Information traverses the system: must decide between the following alternatives
  -
  or
MCM: Reduction to MIS

- **Edge graph:** If $G = (V, E)$ then in $EG(G)$:
  - The node set is $E$
  - $(e, e')$ are connected in $EG(G)$ if they share a node in $G$
- **Observation:** $M$ is a matching in $G$ $\iff$ $M$ is independent in $EG(G)$

![Graph](https://via.placeholder.com/150)

Approximate Distributed Matching

**Theorem:** Maximal matching is $\frac{1}{2}$-approximate MCM.

**Proof:** Let $M$ be a maximal matching, and $M^*$ an MCM. Observe that
1. Any edge in $M$ touches $\leq 2$ edges in $M^*$.
2. Any edge in $M^*$ touches $\geq 1$ edge in $M$.

Map each edge in $M$ to the $M^*$ edges it touches. By (1): #edges mapped to is at most $2|M|$. By (2), all MCM edges are mapped to, i.e., $|M^*| \leq 2|M|$.

Augmenting Paths

- **Basic concept:** augmenting path for a matching $M$
  - alternating $M$-edges and non-$M$ edges
  - starts and ends with unmatched nodes
- **Flipping membership in $M$ increases the size of matching**

**Theorem:** if all augmenting paths w.r.t. $M$ have length at least $2k-1$ then $|M| \geq (1-1/k) \cdot |MCM|$
MCM: Generic Approximation

Why works? Hopcroft-Karp.

Distributed implementation:
1. Nodes collect map of neighborhood to distance 2k+1
2. Appoint leader for each AP (say, endpoint with smaller ID)
3. Leaders simulate MIS algorithm of conflict graph

Complexity:
• Time: \(O(k)\) rounds for \((1 - 1/k)\)-approximation
• Messages are large... (neighborhood to distance 2k+1)
• #paths is \(n^{O(k)}\) ⇒ much computation, even larger messages

Ergo: The CONGEST model

• Same as LOCAL, but 
  messages may contain up to \(B\) bits
• Usually \(B = O(\log n)\)
  – Allows messages to carry \(O(1)\) IDs and variables
    of magnitude \(O(n^{B})\)
  – Similar to the “word model” in RAM
  – Exact value of \(B\) usually doesn’t matter
• Captures network algorithms more faithfully

Canonical Algorithm for CONGEST

Same algorithm:
1. Construct a BFS tree
2. Send all input to root
3. Root computes all outputs, sends back

Running time: \(O(D + \#inputs + \#outputs)\).

Why? In CONGEST, messages may be delayed!
• Pipelining...

Basic Pipelining

Theorem: Suppose \(k\) messages travel on shortest paths. Then no message is delayed more than \(k - 1\) steps. (Any starting times!)

• Proof: Place a token on a message \(m\). If it is delayed by \(m'\) and then meets \(m'\) again, let token take detour with \(m'\).
Basic Pipelining

Theorem: Suppose $k$ messages travel on shortest paths. Then no message is delayed more than $k - 1$ steps. (Any starting times!)

• Proof: Place a token on a message $m$. If it is delayed by $m'$ and then meets $m'$ again, let token take detour with $m'$.

Proof (cont.)

• Before and after token’s detour:
  – Same endpoints, same start time, same finish time
  – Same length (shortest paths!)
  – Hence same number of delays
• But the delay at switching point eliminated
• Consider a sequence of detours: the vector of times-of-delay decreases lexicographically
• Hence if we repeatedly switch, process ends

Proof (end)

• Applying detour maintains #delays
• Eventually cannot apply detour:
  – Recall: Detour applicable if token is delayed by $m'$ and then meets $m'$ again.
• If detour not applicable, no message delays token twice
• Final message does not delay token at all.
• Hence #delays $\leq$ # other messages.

Canonical Algorithm for CONGEST

Same algorithm:
1. Construct a BFS tree
2. Send all input to root
3. Root computes all outputs, sends back

Running time: $O(D + \#inputs + \#outputs)$. Why?
• In a tree, all paths are shortest, length $O(D)$.
• Each piece of input is a message
Back to Matching

Generic algorithm: input is $G$, $\epsilon > 0$

$M = \emptyset$

For $k := 1$ to $\left\lfloor \frac{1}{\epsilon} \right\rfloor$ do

Create “conflict graph” $CG(k)$:

- nodes := augmenting paths of length $2k - 1$
- edges := pairs of intersecting augmenting paths

Find MIS in $CG(k)$

Augment $M$ by flipping edges of paths in MIS //well defined!

Matching in CONGEST?

Recall MIS: random label per AP, local minima win

Observations:

- Leader can select its own local winner
- Winner can be constructed rather than discovered
- All that’s needed is that each node knows how many APs it belongs to

And that’s easy in bipartite graphs!

Counting APs in Bipartite Graphs

Goal: count how many shortest APs of prescribed length end at gray nodes.

Idea: BFS.

- start with unmatched white nodes (1)
- each node sums all first incoming numbers
  - later messages ignored
- white nodes send sum to all neighbors
- gray nodes send sum to their mate
- last nodes know exactly how many APs end with them

Counting APs in Bipartite Graphs

Goal: pick a uniformly random AP among all ending at a specific node.

Idea: inductive construction.

- start at leader (bottom grey)
- at grey nodes, pick edge with probability proportional to number on its far end
- at white nodes, follow matching edge

This defines a uniformly chosen random winner path for each leader.

Remains: resolve conflicts between leaders.
Algorithm for Bipartite Graphs

- Count number of augmenting paths for each leader
- A leader \( v \) of \( m \) paths picks a number \( w_v \), distributed like the minimum of \( m \) uniform variables (easy).
- Token selects next edge with probability proportional to #paths that lead to that edge.
- Each node records the smallest \( w_v \) it has seen
- After creating path, token backtracks unless killed
- best path joins MIS, etc.

Algorithm for Bipartite Graphs

- #nodes in conflict graph = #APs of length \( k \)
  - at most \( n \Delta^{k/2} \leq n^{O(k)} \).
- Hence
  - Random labels have \( O(k \log n) \) bits (\( k \) messages)
  - #iterations is \( O(k \log n) \)
- Iteration \( k \) is emulated in \( O(k) \) steps
- Time complexity: \( O(k^2 \log n) \) for a given \( k \), \( O(\varepsilon^{-3} \log n) \) over all since \( k = 1, 2, ..., 2/\varepsilon \).

General Graphs

- Idea: reduction to bipartite
- Means:
  - Color nodes black or white randomly
  - Ignore monochromatic edges
  - Apply bipartite algorithm
- Can prove: Repeating \( 2^{O(1/\varepsilon)} \) times suffice
  (w.h.p.)

**Theorem:** For any constant \( 0 < \varepsilon < 1 \), in any graph, 
(1 – \( \varepsilon \))-MCM can be computed distributively in time \( O(\log n) \) using messages of size \( O(\log n) \).
Bad Graphs for **CONGEST**

- Low diameter: there’s always a shortcut
  - Good enough for **LOCAL**
- In **CONGEST**: when shortcuts are narrow, low diameter not enough to transmit massive data
- State of the art: graphs of diameter $\log n$ for problems that need to transport $\sqrt{n}$ bits
  - Extends to diameter 3 with weaker lower bounds

---

**Bad Graphs for CONGEST**: Basic Construction

- Bulk: $\sqrt{n}$ paths of length $\sqrt{n}$ each
- Connect corresponding nodes by a star
- Build a tree whose leaves are the star centers $\Rightarrow O(\log n)$ diameter

**File Transfer Problem**: transmit $\sqrt{n}$ bits from sender to receiver
- PR’99: Must take $\Omega(\sqrt{n})$ time!

---

**Bad Graphs for CONGEST**: Applications

- To prove a time lower-bound on $\Pi$, reduce “file transfer” to $\Pi$:
  - $\Pi$ = MST (edge weights) [PR’99]
  - Stable Marriage (rankings) [KP’09]
- Strengthening: Can’t even approximate MST
Conclusion

• Simple abstractions to model networks
• Some nice algorithmic techniques
• Many open problems
  – Heterogeneous link bandwidth
  – Complexity of applications in low-diameter networks
  – Incorporating faults into model

Thanks!