6.852: Distributed Algorithms
Fall, 2009

Class 2
Today’s plan

• Leader election in a synchronous ring:
  – Lower bound for comparison-based algorithms.
• Basic computation in general synchronous networks:
  – Leader election
  – Breadth-first search
  – Broadcast and convergecast
• Reading: Sections 3.6, 4.1-4-2
• Next time:
  – Shortest paths
  – Minimum spanning tree
  – Maximal independent set
  – Reading: Sections 4.3-4.5
Last time

- Model for synchronous networks
- Leader election problem, in simple ring networks
- Two algorithms:
  - [LeLann], [Chang, Roberts]
    - Pass UID tokens one way, elect max
    - Proofs, using invariants
    - Time complexity: $n$ (or $2n$ for halting, unknown size)
    - Communication (message) complexity: $O(n^2)$
  - [Hirshberg, Sinclair]
    - Send UID tokens to successively-doubled distances, in both directions.
    - Message complexity: $O(n \log n)$
    - Time complexity: $O(n)$ (dominated by last phase)
Last time

• Q: Can the message complexity be lowered still more?

• Non-comparison-based algorithms
  – Wait quietly until it’s your “turn”, determined by UID.
  – Message complexity: $O(n)$
  – Time complexity: $O(u_{\text{min}} n)$, or $O(n 2^{u_{\text{min}}})$ if $n$ is unknown
Lower bounds for leader election

Q: Can we get lower time complexity?

Easy n/2 lower bound (informal):
- Suppose an algorithm always elects a leader in time < n/2.
- Consider two separate rings of size n (n odd), R_1 and R_2.
- Algorithm elects processes i_1 and i_2, each in time < n/2.

Now cut R_1 and R_2 at points furthest from the leaders, paste them together to form a new ring R of size 2n.
- Then in R, both i_1 and i_2 get elected, because the time it takes for them to get elected is insufficient for information about the pasting to propagate from the pasting points to i_1 and i_2.
Lower bounds for leader election

- Q: Can we get lower message complexity?
- More difficult $\Omega(n \log n)$ lower bound.
- Assumptions
  - Comparison-based algorithm
  - Unique start state (except for UID), deterministic.
Comparison-based algorithms

- All decisions determined only by relative order of UIDs:
  - Identical start states, except for UID.
  - Manipulate UIDs only by copying, sending, receiving, and comparing them (<, =, >).
  - Can use results of comparisons to decide what to do:
    - State transition
    - What (if anything) to send to neighbors
    - Whether to elect self leader
Lower bound proof: Overview

- For any $n$, there is a ring $R_n$ of size $n$ such that in $R_n$, any leader election algorithm has:
  - $\Omega(n)$ “active” rounds (in which messages are sent).
  - $\Omega(n/i)$ msgs sent in active round $i$ (for $i > \sqrt{n}$).
  - Thus, $\Omega(n \log n)$ msgs total.

- Choose ring $R_n$ with a great deal of symmetry in ordering pattern of UIDs.
  - For $n =$ power of 2: Bit-reversal rings.
  - For general $n$: c-symmetric rings.

- **Key lemma:** Processes whose neighborhoods “look the same” act the same, until information from outside their neighborhoods reaches them.
  - Need many active rounds to break symmetry.
Lower bound proof: Definitions

- A round is **active** if some (non-null) message is sent in the round.
- **k-neighborhood** of a process: The 2k+1 processes within distance k.

- \((u_1, u_2,\ldots, u_k) \& (v_1, v_2,\ldots, v_k)\) are **order-equivalent** provided that \(u_i \leq u_j\) iff \(v_i \leq v_j\) for all \(i,j\).
  - Implies same \(<, =, >\) relationships for all corresponding pairs.
  - Example: \((1 3 6 5 2 7 9)\) vs. \((2 7 9 8 4 10 11)\)

- Two process states \(s\) and \(t\) **correspond** with respect to \((u_1, u_2,\ldots, u_k) \& (v_1, v_2,\ldots, v_k)\) if they are identical except that occurrences of \(u_i\) in \(s\) are replaced by \(v_i\) in \(t\) for all \(i\).
  - Analogous definition for **corresponding messages**.
Lower bound proof:  Key Lemma

• **Lemma:** Suppose A is a comparison-based algorithm on a synchronous ring network. Suppose i and j are processes whose sequences of UIDs in their k-neighborhoods are order-equivalent.
Then at any point after $\leq k$ active rounds, the states of i and j correspond with respect to their k-neighborhoods' UID sequences.
• That is, processes with order-equivalent k-neighborhoods are indistinguishable until after “enough” active rounds.
• Enough: Information has had a chance to reach the processes from outside the k-neighborhoods.
• **Example:** 5 and 8 have order-equivalent 3-neighborhoods, so must remain in corresponding states through 3 active rounds.
Lower bound proof: Key lemma

- **Lemma:** Suppose A is a comparison-based algorithm on a synchronous ring network. Suppose i and j are processes whose sequences of UIDs in their k-neighborhoods are order-equivalent. Then at any point after \( \leq k \) active rounds, the states of i and j correspond with respect to their k-neighborhoods' UID sequences.

- **Proof:**
  - Induction on \( r = \) number of completed rounds.
  - Base: \( r = 0 \).
    - Start states of i and j are identical except for UIDs.
    - Correspond with respect to k-neighborhoods for every \( k \geq 0 \).
  - Inductive step: Assume for \( r-1 \), show for \( r \).
Key lemma

- **Lemma:** Suppose $i$ and $j$ have order-equivalent $k$-neighborhoods. Then at any point after $\leq k$ active rounds, $i$ and $j$ are in corresponding states, with respect to their $k$-neighborhoods.

- **Proof, inductive step:**
  - Assume true after round $r-1$, for all $i,j,k$.
  - Prove true after round $r$, for all $i,j,k$.
  - Fix $i,j,k$, where $i$ and $j$ have order-equivalent $k$-neighborhoods.
  - Assume $i \neq j$ (trivial otherwise).
  - Assume at most $k$ of first $r$ rounds are active.
  - We must show that, after $r$ rounds, $i$ and $j$ are in corresponding states with respect to their $k$-neighborhoods.
  - By inductive hypothesis, after $r-1$ rounds, $i$ and $j$ are in corresponding states with respect to their $k$-neighborhoods.
  - If neither $i$ nor $j$ receives a non-null message at round $r$, they make corresponding transitions, to corresponding states (with respect to their $k$-neighborhoods).
  - So assume at least one of $i,j$ receives a message at round $r$. 
Key lemma

- **Lemma**: Suppose $i$ and $j$ have order-equivalent $k$-neighborhoods. Then at any point after $\leq k$ active rounds, $i$ and $j$ are in corresponding states, with respect to their $k$-neighborhoods.

- **Inductive step, cont’d:**
  - So assume at least one of $i,j$ receives a message at round $r$.
  - Then round $r$ is **active**, and the first $r-1$ rounds include at most $k-1$ active rounds.
  - $(k-1)$-nbhds of $i-1$ and $j-1$ are order-equivalent, since they are included within the $k$-neighborhoods of $i$ and $j$.
  - By inductive hypothesis, after $r-1$ rounds:
    - $i-1$ and $j-1$ are in corresponding states wrt their $(k-1)$-neighborhoods, and thus wrt the $k$-neighborhoods of $i$ and $j$.
    - Similarly for $i+1$ and $j+1$.
  - Thus, messages from $i-1$ to $i$ and from $j-1$ to $j$ correspond.
  - Similarly for msgs from $i+1$ to $i$ and from $j+1$ to $j$.
  - So $i$ and $j$ are in corresponding states and receive corresponding messages, so make corresponding transitions and end up in corresponding states.
Lower bound proof

- So, we have shown that many active rounds are needed to break symmetry, if there are large order-equivalent neighborhoods.

- It remains to show:
  - There exist rings with many, and large, order-equivalent neighborhoods.
  - This causes large communication complexity.

- First, see how order-equivalent neighborhoods cause large communication complexity…
Corollary 1: Suppose A is a comparison-based leader-election algorithm on a synchronous ring network, and k is an integer such that for any process i, there is a distinct process j such that i and j have order-equivalent k-neighborhoods. Then A has more than k active rounds.

Proof: By contradiction.

- Suppose A elects i in at most k active rounds.
- By assumption, there is a distinct process j with an order-equivalent k-neighborhood.
- By Key Lemma, i and j are in corresponding states, so j is also elected—a contradiction.
Corollary 2: Suppose A is a comparison-based algorithm on a synchronous ring network, and k and m are integers such that the k-neighborhood of any process is order-equivalent to that of at least m-1 other processes. Then at least m messages are sent in A's k^{th} active round.

Proof:
- By definition, some process sends a message in the k^{th} active round.
- By assumption, at least m-1 other processes have order-equivalent k-neighborhoods.
- By the Key Lemma, immediately before this round, all these processes are in corresponding states. Thus, they all send messages in this round, so at least m messages are sent.
Highly symmetric rings

- That’s how order-equivalent neighborhoods yield high communication complexity.
- Now, show existence of rings with many, large order-equivalent neighborhoods.
- For powers of 2: Bit-reversal rings
  - UID is bit-reversed process number.
  - Example:
    - For every segment of length $n/2^b$, there are (at least) $2^b$ order-equivalent segments (including original segment).
    - So for every process $i$, there are at least $n/4k$ processes (including $i$) with order-equivalent $k$-neighborhoods, for $k < n/4$.
    - More than $n/8$ active rounds.
    - Number of messages $\geq n/4 + n/8 + n/12 + ... + 2 = \Omega(n \log n)$
C-symmetric rings

- **c-symmetric ring**: For every $l$ such that $\sqrt{n} < l < n$, and every sequence $S$ of length $l$ in the ring, there are at least $\left\lfloor \frac{cn}{l} \right\rfloor$ order-equivalent occurrences.

- **[Frederickson-Lynch]** There exists $c$ such that for every positive integer $n$, there is a c-symmetric ring of size $n$.

- Given c-symmetric ring, argue similarly to before.
General Synchronous Networks
General synchronous networks

- Not just rings, but arbitrary digraphs.
- Basic tasks, such as broadcasting messages, collecting responses, setting up communication structures.
- Basic algorithms.
- No lower bounds.
- Algorithms are simplified versions of algorithms that work in asynchronous networks. We’ll revisit them in asynchronous setting.
General synchronous network assumptions

- Digraph G = (V,E):
  - V = set of processes
  - E = set of communication channels
  - distance(i,j) = shortest distance from i to j
  - diam = max distance(i,j) for all i,j
  - Assume: Strongly connected (diam is finite), UIDs

- Set M of messages
- Each process has states, start, msgs, trans, as before.
- Processes communicate only over digraph edges.
- Generally don’t know the entire network, just local neighborhood.
- Local names for neighbors.
  - No particular order for neighbors, in general.
  - But (technicality) if incoming and outgoing edges connect to same neighbor, the names are the same (so the node “knows” this).
Leader election in general synchronous networks

• **Assume:**
  – Use UIDs with comparisons only.
  – No constraints on which UIDs appear, or where they appear in the graph.
  – Processes know (upper bound on) graph diameter.

• **Required:** Everyone should eventually set status $\in \{\text{leader, non-leader}\}$, exactly one leader.

• Show basic **flooding algorithm**, sketch proof using invariants, show **optimized version**, sketch proof by relating it to the basic algorithm.

• **Basic flooding algorithm:**
  – Every round: Send max UID seen to all neighbors.
  – Stop after diam rounds.
  – Elect self iff own UID is max seen.
Leader election in general synchronous networks

- **states**
  - \( u \), initially UID
  - \( \text{max-uid} \), initially UID
  - \( \text{status} \in \{ \text{unknown}, \text{leader}, \text{not-leader} \} \), initially unknown
  - \( \text{rounds} \), initially 0

- **msgs**
  - if \( \text{rounds} < \text{diam} \) send \( \text{max-uid} \) to all out-nbrs

- **trans**
  - increment \( \text{round} \)
  - \( \text{max-uid} := \max (\text{max-uid}, \text{UIDs received}) \)
  - if \( \text{round} = \text{diam} \) then
    - \( \text{status} := \text{leader} \) if \( \text{max-uid} = u \), not-leader otherwise
Leader election in general network

Start configuration
Leader election in general network
Leader election in general network

Round 1 (trans)
Leader election in general network

Round 2 (start)
Leader election in general network

Round 2 (msgs)
Leader election in general network

Round 2 (trans)
Leader election in general network

Round 3 (start)
Leader election in general network

Round 3 (msgs)
Leader election in general network

Round 3 (trans)
Leader election in general network

Round 4 (start)
Leader election in general network

Round 4 (msgs)
Leader election in general network

Round 4 (trans)
Leader election in general network

• Basic flooding algorithm (summary):
  – Assume diameter is known (diam).
  – Every round: Send max UID seen to all neighbors.
  – Stop after diam rounds.
  – Elect self iff own UID is max seen.

• Complexity:
  – Time complexity (rounds): diam
  – Message complexity: diam |E|

• Correctness proof?
Key invariant

- **Invariant:** After round $r$, if $\text{distance}(i,j) \leq r$ then $\text{max-uid}_j \geq \text{UID}_i$.

- **Proof:**
  - Induction on $r$.
  - Base: $r = 0$
    - $\text{distance}(i,j) = 0$ implies $i = j$, and $\text{max-uid}_i = \text{UID}_i$.
  - Inductive step: Assume for $r-1$, prove for $r$.
    - If $\text{distance}(i,j) \leq r$ then there is a node $k$ in $\text{in-nbrs}_j$ such that $\text{distance}(i,k) \leq r - 1$.
    - By inductive hypotheses, after round $r-1$, $\text{max-uid}_k \geq \text{UID}_i$.
    - Since $k$ sends its max to $j$ at round $r$, $\text{max-uid}_j \geq \text{UID}_i$ after round $r$. 
Reducing the message complexity

- Slightly optimized algorithm:
  - Don't send same UID twice.
  - New state var: new-info: Boolean, initially true
  - Send max-uid only if new-info = true
  - new-info := true iff max UID received > max-uid
Leader election in general network

Start configuration
Leader election in general network

Round 1 (msgs)
Leader election in general network

Round 1 (trans)
Leader election in general network

Round 2 (start)
Leader election in general network

Round 2 (msgs)
Leader election in general network

Round 2 (trans)
Leader election in general network

Round 3 (start)
Leader election in general network

Round 3 (msgs)
Leader election in general network

Round 3 (trans)
Leader election in general network

Round 4 (start)
Leader election in general network

Round 4 (msgs)
Leader election in general network

Round 4 (trans)
Leader election in general network

- Slightly optimized algorithm (summary):
  - Don't send same UID twice
  - New state variable: new-info: Boolean, initially true
  - Send max-uid just when new-info = true
  - new-info := true iff max UID received > max-uid
  - Can improve communication cost drastically, though not the worst-case bound, diam |E|.

- Correctness Proof?
  - As before, or:
    - Can use another important proof method for distributed algorithms: simulation relations.
Simulation relation

- Relates new algorithm formally to an original one that has already been proved correct.
- Correctness then carries over to new algorithm.
- Often used to show correctness of optimized algorithms.
- Can repeat in several stages, adding more optimizations.

- “Run the two algorithms side by side.”
- Define simulation relation between states of the two algorithms:
  - Satisfied by start states.
  - Preserved by every transition.
  - Outputs should be the same in related states.
Simulation relation for the optimized algorithm

• Key invariant of the optimized algorithm:
  – If \( i \in \text{in-nbrs}_j \) and \( \text{max-uid}_i > \text{max-uid}_j \) then \( \text{new-info}_i = \text{true} \).
  – That is, if \( i \) has better information than \( j \) has, then \( i \) is planning to send it to \( j \) on the next round.
  – Prove by induction.

• Simulation relation: All state variables of the basic algorithm (all but \( \text{new-info} \)) have the same values in both algorithms.

• Start condition: By definition.

• Preserved by every transition:
  – Key property: \( \text{max-uids} \) are always the same in the two algorithms.
  – Consider \( i \in \text{in-nbrs}_j \).
  – If \( \text{new-info}_i = \text{true} \) before the step, then the two algorithms behave the same with respect to \((i,j)\).
  – Otherwise, only the basic algorithm sends a message. However, by the invariant, \( \text{max-uid}_i \leq \text{max-uid}_j \) before the step, and the message has no effect.
Why all these proofs?

• Distributed algorithms can be quite complicated, subtle.
• Easy to make mistakes.
• So careful reasoning about algorithm steps is generally more important than for sequential algorithms.
Other problems besides leader election…

• Breadth-first search
• Breadth-first spanning trees, shortest-paths spanning trees,…
• Minimum spanning trees
• Maximal independent sets
Breadth-first search

• **Assume:**
  – Strongly connected digraph, UIDs.
  – No knowledge of size, diameter of network.
  – Distinguished source node \( i_0 \).

• **Required:** Breadth-first spanning tree, rooted at source node \( i_0 \).
  – Branches are directed paths in the given digraph.
  – Spanning: Includes every node.
  – Breadth-first: Node at distance \( d \) from \( i_0 \) appears at depth \( d \) in tree.
  – Output: Each node (except \( i_0 \)) sets a parent variable to indicate its parent in the tree.
Breadth-first search
Breadth-first search
Breadth-first search algorithm

- **Mark** nodes as they get incorporated into the tree.
- Initially, only $i_0$ is marked.
- **Round 1**: $i_0$ sends **search** message to out-nbrs.
- **At every round**: An unmarked node that receives a **search** message:
  - Marks itself.
  - Designates one process from which it received **search** as its parent.
  - Sends **search** to out-nbrs at the next round.

- **Q**: What state variables do we need?
- **Q**: Why does this yield a BFS tree?
Breadth-first search

Round 1 (start)
Breadth-first search

Round 1 (msgs)
Breadth-first search

Round 1 (trans)
Breadth-first search

Round 2 (start)
Breadth-first search

Round 2 (msgs)
Breadth-first search

Round 2 (trans)
Breadth-first search

Round 3 (start)
Breadth-first search

Round 3 (msgs)
Breadth-first search

Round 3 (trans)
Breadth-first search

Round 4 (start)
Breadth-first search

Round 4 (msgs)
Breadth-first search

Round 4 (trans)
Breadth-first search

Round 5 (start)
Breadth-first search

Round 5 (msgs)
Breadth-first search

Round 5 (trans)
Breadth-first search algorithm

- **Mark** nodes as they get incorporated into the tree.
- Initially, only $i_0$ is marked.
- **Round 1:** $i_0$ sends **search** message to out-nbrs.
- **At every round:** An unmarked node that receives a search message:
  - Marks itself.
  - Designates one process from which it received search as its parent.
  - Sends search to out-nbrs at the next round.
- Yields a BFS tree because all the branches are created synchronously.
- **Complexity:** Time = diam + 1; Messages = $|E|$
BFS, bells and whistles

- Child pointers?
  - Easy with bidirectional communication.
  - What if not?
    - Could use BFS to search for parents.
    - High message bit complexity.

- Termination?
  - With bidirectional communication?
    - “Convergecast”
  - With unidirectional communication?
Applications of BFS

- Message broadcast:
  - Can broadcast a message while setting up the BFS tree ("piggyback" the message).
  - Or, first establish a BFS tree, with child pointers, then use it for broadcasting.
    - Can reuse the tree for many broadcasts
    - Each takes time only $O(\text{diameter})$, messages $O(n)$.

- For the remaining applications, assume bidirectional edges (undirected graph).
Applications of BFS

- Global computation:
  - Sum, max, or any kind of data aggregation: Convergecast on BFS tree.
  - Complexity: Time $O(\text{diameter})$; Messages $O(n)$

- Leader election (without knowing diameter):
  - Everyone starts BFS, determines max UID.
  - Complexity: Time $O(\text{diam})$; Messages $O(n |E|)$ (actually, $O(\text{diam} |E|)$).

- Compute diameter:
  - All do BFS.
  - Convergecast to find height of each BFS tree.
  - Convergecast again to find max of all heights.
Next time

- More distributed algorithms in general synchronous networks:
  - Shortest paths (Bellman-Ford)
  - Minimum spanning trees
  - Maximal independent sets (just summarize)
- Reading: Sections 4.3-4.5.