Problem Set 10 Solutions

This problem set is due **at 11:59pm** on **Friday, May 8, 2015.**

Exercise 10-1. Read the lecture slides for Lectures L19 and L20.

Problem 10-1. Leader Election in a Synchronous Ring [25 points]

Consider a collection of n *identical* processes arranged in a synchronous ring network. Each process has two sets of ports, one to each of its immediate neighbors, with its ports to its clockwise neighbor named $left$ and its ports to its counterclockwise neighbor named $right$. Thus, the processes have a common sense of orientation.

The goal is for the processes to elect a single leader: exactly one process should eventually output LEADER.

(a) [5 points] First suppose that the processes are deterministic, and that they know n (the size of the ring). Either give a correct leader election algorithm for this case, or prove that no such algorithm exists. If you give an algorithm, analyze its time and message complexity.

Solution: Impossible. Suppose for contradiction that such an algorithm exists.

We can use a standard inductive symmetry argument. Namely, prove by induction on the number r of rounds that, after r rounds, all the processes are in identical states.

Since the algorithm must solve the leader election problem, eventually, one process outputs LEADER. But then all processes do the same thing, in the same round.

(b) [10 points] Now suppose that the processes are probabilistic (i.e., randomized), and that they know n. We would like an algorithm that (a) never elects more than one leader, and (b) with probability 1, eventually elects a leader. Either give an algorithm satisfying these properties, or prove that none exists.

If you give an algorithm, analyze its time and message complexity. Your analysis should relate the complexity to the success probability. Specifically, for any ε, $0 < \varepsilon < 1$, you should provide bounds on the time and message complexity that hold with probability at least $1 - \varepsilon$.

Solution: Here a simple algorithm exists.

Lemma, similar to one from class: If n processes choose ids uniformly at random, independently, from $\{1, \ldots, n^2\}$, then with probability at least $1/2$, the chosen numbers are all distinct.

Algorithm: The algorithm works in a series of phases. In each phase, all processes choose random ids from a sufficiently large space, defined as in the lemma as $\{1, ..., n^2\}$. Then they all send their ids around the ring, (say) in the clockwise direction. After exactly n steps, every process examines the sequence of ids that it has received. There are three cases:

1. If the maximum id in the sequence is not unique, it abandons this phase and goes on to the next.
2. If the maximum id in the sequence is unique and is the process' own id, then it outputs LEADER and halts.
3. If the maximum id in the sequence is unique and is not the process' own id, then it just halts.

It should be clear that, at the first phase where all the processes choose distinct ids, exactly one process elects itself the leader, and then the algorithm halts. So certainly the algorithm never elects more than one leader.

At each phase, with probability at least $1/2$, all the chosen ids are distinct and the algorithm terminates. Since the choices in different phases are independent, the probability that the algorithm finishes within h phases is at least $1 - \frac{1}{2^h}$. Thus, with probability 1, it eventually finishes.

We analyze the time and message complexity. Each phase consists of n rounds, and sends n^2 (single-hop) messages. Consider any ε, $0 < \varepsilon < 1$. Choose h to be the smallest integer such that $\frac{1}{2^h} \leq \varepsilon$, that is, $h = \lceil \lg(1/\varepsilon) \rceil$. Then with probability at least $1 - \delta^h \geq 1 - \varepsilon$, the algorithm finishes within h phases, using $n \cdot h$ rounds and $n^2 \cdot h$ messages. That is, with probability at least $1 - \varepsilon$, the time complexity is at most $n\lceil \lg(1/\varepsilon) \rceil$ and the message complexity is at most $n^2 \lceil \lg(1/\varepsilon) \rceil$.

(c) [10 points] Finally suppose that the processes are probabilistic and they do not know n. That is, the same algorithm must work regardless of the size of the ring in which the processes are placed. We would again like an algorithm that (a) never elects more than one leader, and (b) with probability 1, eventually elects a leader. Either give an algorithm satisfying these properties, or prove that none exists. If you give an algorithm, analyze its time and message complexity as described in Part (b).

Solution: This is impossible. Suppose for contradiction that such an algorithm exists. Consider the algorithm operating in a ring S of size n, for any particular value of n. Number the processes of S as $1, \ldots, n$ based on their positions in the ring, counting clockwise. Since the algorithm eventually elects a leader in ring S with probability 1, there must be at least one execution α of S that leads to some process j outputting LEADER. That is, there is some particular mapping from processes to sequences of

random choices such that, in the execution α in which these choices are made, some process j is elected leader.

Now consider a ring R of size $2n$ consisting of two half-rings R_1 and R_2, each of size n. Number the processes of R as $1, \ldots, 2n$. Then there is an execution α' of R in which processes i and $n+i$ in R happen to make the same random choices as process i in S. In execution α', processes i and $n+i$ behave exactly like process i in execution α of S. Since process j outputs LEADER in execution α, both processes j and $n+j$ output LEADER in execution α'. This contradicts problem requirement (a).

Notice that this proof shows that we can't achieve *any* positive election probability ε, not just probability 1.

Problem 10-2. Breadth-First Search in an Asynchronous Network [25 points]

The asynchronous Breadth-First Search algorithm presented in class involves corrections that could trigger the sending of many messages, resulting in worst-case message complexity $O(nE)$ and worst-case time complexity $O(diam \cdot n \cdot d)$, until all nodes' $parent$ variables have stabilized to correct parents in a breadth-first spanning tree. (We are not considering individual processes' $parent$ outputs here, nor global termination. We are also ignoring local processing time.)

This problem involves designing a better asynchronous Breadth-First Search algorithm, one that does not make any corrections. Thus, once a process sets its $parent$ variable, it can output the value since that is its final decision.

Assume that the network graph is connected and contains at least two nodes.

(a) [18 points] Describe carefully, in words, an algorithm in which the root node v_0 coordinates the construction of the tree, level by level. Your algorithm should take time $O(diam^2 \cdot d)$ until all the $parent$ variables are set.

(*Hint:* The root node can conduct broadcast-convergecast waves to build the successive levels in the tree. Four types of messages should be enough, e.g., $search$ messages that test for new nodes, $parent(b)$ (b a Boolean) for parent/nonparent responses, and $ready$ and $done$ messages for broadcasting and convergecasting signals on the tree.)

Solution: Node v_0 initiates successive phases of the algorithm. At each phase d, exactly the nodes at distance d from v_0 get incorporated into the BFS tree.

At phase 1*:*
Node v_0 sends $search$ messages to all its neighbors. The neighbors record v_0 as their parent, record that they are new nodes, and send $parent$ responses back to v_0. When v_0 receives $parent$ responses from all its neighbors, it is ready to start phase 2.

At phase d*,* $d \geq 2$*:*
Node v_0 broadcasts $ready$ messages down all the branches of the tree built so far, until they reach the new nodes. Each new node sends $search$ messages to all its neighbors. When a node receives a $search$ message, if it does not already have a parent, it sets its

$parent$ variable to the sender's id, records that it is new, and sends a $parent$ response. If it already has a parent, it sends a $nonparent$ response.

When a new node has received responses to all its $search$ messages, it sends a $done(b)$ message to its parent, where $b =$ TRUE if the node has received at least one $parent$ response, $=$ FALSE otherwise. The $done$ messages get convergecast up the tree; each node sets the bit b in the message it sends to its parent to the "or" of the bits it received from its children.

When node v_0 has received $done$ messages from all its children, it begins phase $d+1$ if any of the messages contain value 1; otherwise it halts.

(b) [7 points] Analyze the time and communication complexity of your algorithm, and compare them to the costs of the asynchronous BFS algorithm presented in class.

Solution: The time complexity is $O(diam^2 \cdot d)$. Each phase takes time $O(diam \cdot d)$, and there are $O(diam)$ phases.

The message complexity is $O(E + diam \cdot n)$. Each edge is traversed once in each direction with $search$ and $parent$ messages. The $ready$ and $done$ messages traverse tree edges only, so there are $O(n)$ of these per phase.

Solution:

If it is instructive here is some code for part (a) that we think might help understand this problem more deeply.

Process v_0

State variables:
for each $v \in \Gamma(v_0)$, $send(v)$, a queue, initially $(search)$
$responded \subseteq \Gamma(v_0)$, initially $\emptyset$
$newinfo$, a Boolean, initially $false$

Transitions:

input $receive(search)_{v,v_0}$, $v \in \Gamma(v_0)$
Effect: add $parent(false)$ to $send(v)$

input $receive(parent(true))_{v,v_0}$, $v \in \Gamma(v_0)$
Effect:
$responded := responded \cup \{v\}$
if $responded = \Gamma(v_0)$ then
    for each $w \in \Gamma(v_0)$, add $ready$ to $send(w)$
    $responded := \emptyset$

$newinfo := false$

input $receive(done(b))_{v,v_0}$, b a Boolean, $v \in \Gamma(v_0)$
Effect:
$responded := responded \cup \{v\}$
$newinfo := newinfo \vee b$
if $responded = \Gamma(v_0)$ and $newinfo$ then
    for each $w \in \Gamma(v_0)$, add $ready$ to $send(w)$
    $responded := \emptyset$
    $newinfo := false$

output $send(m)_{v,v_0}$, m a message, $v \in \Gamma(v_0)$
Precondition: $m = head(send(v))$
Effect: remove head of $send(v)$

Process u, $u \neq v_0$

State variables:
$parent \in \Gamma(u) \cup \{\perp\}$, initially $\perp$
$children \subseteq \Gamma(u)$, initially $\emptyset$
$newnode$ a Boolean, initially $false$
for each $v \in \Gamma(u)$, $send(v)$, a queue, initially empty
$responded \subseteq \Gamma(u)$, initially $\emptyset$
$newinfo$, a Boolean, initially $false$

Transitions:

input $receive(search)_{v,u}$, $v \in \Gamma(u)$
Effect:
if $parent = \perp$ then
    $parent := v$
    $newnode := true$
    add $parent(true)$ to $send(v)$
else add $parent(false)$ to $send(v)$

input $receive(parent(b))_{v,u}$, b a Boolean, $v \in \Gamma(u)$
Effect:
if b then
    $children := children \cup \{v\}$
    $newinfo := true$
$responded := responded \cup \{v\}$
if $responded = \Gamma(u)$ then add $done(newinfo)$ to $send(parent)$

input $receive(ready)_{v,u}$, $v \in \Gamma(u)$
Effect:
if $newnode$ then for each $w \in \Gamma(u)$, add $search$ to $send(w)$
else for each $w \in children$, add $ready$ to $send(w)$
$responded := \emptyset$
$newinfo := false$

input $receive(done(b))_{v,u}$, b a Boolean, $v \in \Gamma(u)$
Effect:
$responded := responded \cup \{v\}$
$newinfo := newinfo \vee b$
if $responded = children$ then add $done(newinfo)$ to $send(parent)$

output $send(m)_{u,v}$, m a message, $v \in \Gamma(u)$
Precondition: $m = head(send(v))$
Effect: remove head of $send(v)$

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2015