

Distributed Fixpoints

Andrew Twigg (with Karl Krukow)

`andrew.twigg@cl.cam.ac.uk`

Fixpoints

- Have a function $F : X^n \rightarrow X^n$ over some (complete) lattice X . We wish to find an element $x^* \in X^n$ such that $x^* = F(x^*)$. If there's more than one such x^* , we want the least one. Then x^* is the least fixed point of F .

Theorem 1 *If F is continuous then it has a unique least fixed point $\text{FIX } F$, and convergence to this point is guaranteed by the iteration $x^{k+1} = F(x^k)$, with $x^0 = \perp$.*

- Have a fully-connected reliable network with n processors. Distribute $F = \langle f_1, f_2, \dots, f_n \rangle$ by having each processor i know f_i .
- The goal is for some designated processor u to compute (and know that it's computed) $(\text{FIX } F) u$.

Some Applications

- Shortest-path routing (also network flow problems)

- eg Bellman-Ford algorithm (shortest path lengths to a node p).

- $$x_i = \begin{cases} \min_j x_j + d_{ij} & i \neq p \\ 0 & i = p \end{cases}$$

- Distributed Pagerank

- A node's pagerank is its stationary probability in a random walk on the web graph (assuming it's irreducible and aperiodic)

- $x_i = \sum_j x_j p_{ij}$ where p_{ij} is probability of moving from j to i .

- Distributed Security Protocols

- Nodes reference other nodes about some node's trustworthiness.
- Define trustworthiness as the least fixed point of others' policies.

Some Applications (2)

- Distributed load balancing, traffic (Nash) equilibria, ...
- In all cases, correctness of the the fixed point algorithm implies correctness of the resulting algorithms.
 - Though we don't know of many reductions which hand back an optimal algorithm...
 - e.g. An asynchronous iteration $x_u = f_u(x)$ with the routing reduction gives the ARPANET routing algorithm.
 - Not exactly sure what kind of routing algorithm the disjoint components algorithm hands back!

Dependency Graphs

- A dependency graph $G(F)$ represents the functional dependencies between the f_i 's.
 - We'll have an edge (u, v) iff f_u depends on x_v .
 - If there's no path $u \rightsquigarrow v$ then f_u is independent of x_v .
- Let $G(F)_p$ be the subgraph of $G(F)$ reachable from p .
- Basic idea is that each processor receives data from other processors, computes $x_u = f_u(x)$ for some x and sends something (probably x_u) to some processors.
- We'll use the dependency graph to figure out which processors to send information to.

Distributed Fixpoint Algorithms

- Assume that all the functions f_i are fixed (but unknown to processors $j \neq i$).
- Some message find_u originates at processor u . We want find_u to return $(\text{FIX } F) u$ in some finite time.
- Algorithms can be asynchronous or synchronous.
 - Asynchronous: Each node repeats some local computation/communication, independently of other nodes.
 - Synchronous: A node performs some (bounded) computation and communication only after receiving a message from some node.
- Consider synchronous algorithms - the network is triggered by find_u messages.
 - May be many such messages in the network at any given time.

Algorithms (2)

- We're interested in the communication complexity of the algorithms (number of bits per link).
 - The bottleneck in large (Internet)-scale distributed systems.
 - And potentially something that could incur a cost.
- Time complexity depends on the level of parallelism achieved

Theorem 2 *Unless $P=NP$, no polynomial-time algorithm is time-optimal (achieves maximal parallelism)*

- k-colouring the dependency graph such that no monochromatic subgraph has a directed cycle is equivalent to performing an iteration in k (parallel) steps.

Chaotic Iteration

A root node u begins the computation with $\mathbf{find}_u()$:

find $_u()$

$v_u \leftarrow \perp^n$

do

$v'_u \leftarrow v_u; v_u \leftarrow (v_u/u \mapsto f_u(v'_u))$

for each child $u_1 \dots u_l$ of u

$m_i \leftarrow \mathbf{find}_{u_i}(v_u, \{u\})$

$v_u \leftarrow \sqcap (m_i)_{i=1}^l$

while $(v_u \neq v'_u)$

return v_u

For a non-root node u :

find $_u(v_u, S)$

$v'_u \leftarrow (v_u/u \mapsto f_u(v'_u))$

for each child $u_1 \dots u_l$ of u

if $(u_i \notin S)$

$m_i \leftarrow \mathbf{find}_{u_i}(v'_u, S \cup \{u\})$

$v_u \leftarrow \sqcap (m_i)_{i=1}^l$

return v_u

- The m_i 's passed around are tables of values, bounded above by the true value, i.e. $m_i \sqsubseteq \text{FIX } F$
- The algorithm is globally synchronous.
 - No node starts iteration i until every node has completed iteration $(i - 1)$.
- High communication complexity and low parallelism.

Distributed Fixpoint Induction

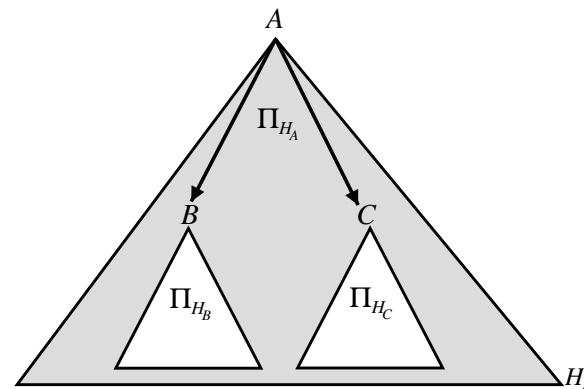
- Based on fixed point induction. Two observations:
 - For a node u , if x_i^* is indeed the least fixed point of the i th child of u (in $G(F)$), then $f_u(x_1^*, \dots, x_l^*)$ is the least fixed point of u .
 - For two children p, q of u , if the subgraphs $G(F)_p, G(F)_q$ are disjoint, then the fixpoints x_p, x_q can be computed independently and in parallel.
- So, assume the children *are* correct and divide and conquer the dependency graph by recursively computing fixpoints of disjoint components. Easy!

find_u

for each child $u_1 \dots u_l$ of u

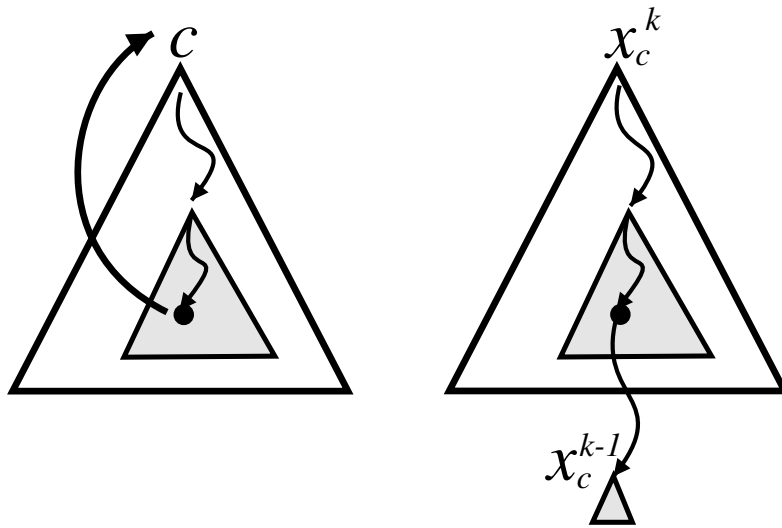
$m_i \leftarrow$ **find_{u_i}**

return $f_u(m_1, \dots, m_l)$



Distributed Fixpoint Induction (2)

- Not really that easy...the algorithm won't terminate if $G(F)$ contains a cycle.
- Basic idea: For each cycle C in $G(F)$, elect at most one node $c \in C$ to be the leader of C .
 - c cuts the cycle C into a path, with the value x_c^k at the start and x_c^{k-1} at the end:



- c then iterates the path until $x_c^k = x_c^{k-1}$ for some k .

Distributed Fixpoint Induction (3)

Each node has global variables $v_u = \perp, \text{leader:bool}$.
Node u begins with $\text{find}_u(\emptyset)$.

find $_u(S)$

if $u \in S$

comment: u is the leader of a cycle

leader \leftarrow true

return v_u

else

leader \leftarrow false

do

$v'_u \leftarrow v_u$

for each child $u_1 \dots u_l$ of u

$m_i \leftarrow \text{find}_{u_i}(S \cup \{u\})$

$v_u \leftarrow f_u(m_1, \dots, m_l)$

while (leader is true and $v_u \neq v'_u$)

return v_u

Theorem 3 *The algorithm is correct: $\text{find}_u(\emptyset) = (\text{FIX } F) u$.*

- By induction on the size of the subgraphs $G(F)_p$ visited.
- We must also be careful about concurrent find calls altering the value at a node during another call. Fortunately, this can only make things better (giving a Gauss-Seidel effect).

Distributed Fixpoint Induction (4)

Conjecture 1 *The algorithm is (asymptotically) communication-optimal if $G(F)$ contains only simple cycles.*

- The algorithm is locally-synchronous
 - Nodes begin the i th iteration upon receiving their $(i - 1)$ th iteration messages.

Observation 1 *The algorithm performs poorly on intersecting cycles*

- If $G(F) = K_n$ then the number of messages ($O(1)$ bits) is exponential in n (the corresponding lower bound is $\Omega(\text{poly}(n))$).
- The following is bad enough (entering at u ; v conducts a complete set of iterations of w, u, x for each iteration by x)

