Brief Announcement: Self-stabilizing Silent Disjunction in an Anonymous Network

Ajoy K. Datta[1], Stéphane Devismes[2], and Lawrence L. Larmore[1]

[1] Department of Computer Science, University of Nevada Las Vegas, USA
[2] VERIMAG UMR 5104, Université Joseph Fourier, France

1 Problem

Given a network of processes $\mathcal{G}$, where each process has a fixed *input bit*, $Input(x)$, the *disjunction problem* is for each process to compute $Output = \bigvee_{x \in \mathcal{G}} Input(x)$, the disjunction of all input bits in the network.

A *distributed solution* to the disjunction problem is a distributed algorithm which computes an *output bit* for each process, such that all output bits are equal to *Output*. The solution given in this paper, the distributed algorithm DISJ, correctly solves the disjunction problem if the network is connected. DISJ is self-stabilizing [1,2], meaning that a correct output configuration is reached in finite time after arbitrary initialization, and is silent, meaning that eventually the computation of DISJ will halt. DISJ works under the *unfair* scheduler (daemon). DISJ is uniform, meaning that every process has the same program, and is anonymous, meaning that processes are not required to have distinguished IDs. The *round complexity* of DISJ is $O(n)$, where n is the size of the network. We use the composite model of computation [2]. We are not aware of any closely related work in the literature. Although we use some of the same techniques in this paper that are used for leader election, the disjunction problem in an anonymous network cannot be solved by using a leader election algorithm, nor by using an algorithm to construct a spanning tree. In fact, there is no distributed algorithm which elects a leader or which constructs a spanning tree for general anonymous networks.

2 DISJ

Our algorithm, DISJ, solves the disjunction problem in an anonymous connected network, $\mathcal{G}$. The fundamental idea of DISJ is to build a *local BFS tree* rooted at every process whose input bit is 1. Each process will join the tree rooted at the nearest process with input bit 1; ties will be broken arbitrarily. The construction of the BFS trees is by flooding.

The main difficulty with this method is the possibility that, in the initial configuration (which is arbitrary) there could be "fictitious" BFS trees. It is necessary to delete all such fictitious trees. This is an easy task if $Output = 1$,

A.W. Richa and C. Scheideler (Eds.): SSS 2012, LNCS 7596, pp. 46–48, 2012.

but is difficult if $Output = 0$, where there is a danger that the algorithm will never eliminate all fictitious trees. Fictitious trees continually delete themselves from the root end; our problem is to ensure that the tree does not grow as fast at the leaf end as it deletes itself from the root end.

The method we use to ensure deletion of fictitious trees is derived from the *color wave* method of [3]. Each process in a tree, whether true or fictitious, has a *color*, either 0 or 1. A process can only recruit a new process to the tree if its color is 1, and the recruited process will initially have color 0. Colors change in pipelined convergecast waves. These rules guarantee that fictitious trees will lose processes from the root end approximately twice as fast as they recruit new leaves.

In our algorithm, colors change in convergecast waves which cannot pass each other. Each wave is behind its predecessor by some positive amount. In order for a wave to reach the root of a tree, all preceding waves must be *absorbed* by the root. Only a process whose input bit is 1 can absorb waves. A fictitious tree will not be rooted at a process with input bit 1, and thus color waves will not be absorbed. "Color lock," the situation where the waves are maximally crowded and cannot move up, will eventually stop the growth of any fictitious tree.

We are able to prove, using fairly straightforward methods, that DISJ converges in $O(d)$ rounds if $Output = 1$, where d is the diameter of the network. On the other hand, DISJ requires $O(n)$ rounds to delete all fictitious trees if $Output = 0$. The worst case round complexity for DISJ is thus $O(n)$.

We use the concept of *energy* introduced in [3] to prove that DISJ is self-stabilizing and has time complexity $O(n)$. $Energy(x)$ is a positive integer for each process x whose output bit is 1. If $Output = 0$, $Energy(x) \leq 2n$ for all x, and the maximum value of *Energy* decreases by at least 1 during every round, and must eventually reach zero. At that point, every process has output bit 0.

In the case that $Output = 1$, all BFS trees will be in their final form within $O(d)$ rounds, where d is the diameter of the network. All color waves will then stop within $O(d)$ additional rounds, after which DISJ will be silent.

In DISJ, each process x has the following variables.

$x.out$, Boolean, the *output bit* of x. When DISJ converges, all values of $x.out$ are equal to *Output*. During execution of DISJ, each process has output bit 1 if and only if it is currently a member of a tree.

$x.level$, which is either a non-negative integer or ∞. If $Outpt = 1$, $x.level$ converges to the distance from x to the nearest process whose input bit is 1. If $Output = 0$, $x.level = \infty$ for all x after convergence.

$x.parent \in N(x) \cup \{\perp\}$, the parent of x in its BFS tree, where $N(x)$ is the set of neighbors of x. If $Input(x) = 1$ or $Output = 0$, then $x.parent = \perp$ when DISJ converges.

$x.color \in \{0, 1\}$, the *color* of x.

$x.done$, Boolean, used to indicate that DISJ is finished and the color waves should stop.

DISJ has five actions, as follows.

Reset: A process which detects that it is an erroneous state, or needs to decrease its level, executes this action, setting $x.out$ to 0 and $x.parent$ to $\perp$, becoming a *free* process.

Initialize: A free process whose output bit is zero becomes the root of a new BFS tree.

Join: A free process x whose output bit and input bits are both 0, but which has a neighbor whose output bit is 1, joins a BFS tree by linking to a neighbor.

Change Color: A process x which is a member of a BFS tree changes color, from 0 to 1 or from 1 to 0. In order to change color, all children of x (in its BFS tree) must have color opposite to x, while either the color of $x.parent$ equals $x.color$, or x is the root of a BFS tree. When a root changes color, a color wave is deleted, as we explain below.

Finish: In the case that $Output = 1$, the network eventually consists of the disjoint union of one or more BFS trees, one rooted at each process whose input bit is 1. In order for DISJ to be silent, we must freeze the color waves. This action enables a convergecast *finishing wave* to move up each BFS tree; when that wave reaches the root, that root will no longer execute the change color action. Eventually, DISJ will halt.

Normal Growth. A true tree grows by processes executing the Join action. Only a process of color 1 can recruit a neighbor, and the new recruit is given the color 0. That new recruit may wait two rounds before it can recruit new processes itself. If there are no fictitious trees, color waves move up the tree, and the new recruit will eventually be enabled to be the recruiter.

Fictitious Trees. A fictitious tree recruits as a true tree, but the color waves are unable to move up indefinitely; thus its growth eventually stops. After a fictitious tree becomes "color locked," it deletes itself by repeated execution of the Reset action. Within $O(n)$ rounds, there will be no more fictitious trees; within $O(d)$ additional rounds, a legitimate configuration will be reached.

References

1. Dijkstra, E.: Self stabilizing systems in spite of distributed control. Communications of the Association of Computing Machinery 17, 643–644 (1974)
2. Dolev, S.: Self-Stabilization. The MIT Press (2000)
3. Datta, A.K., Larmore, L.L., Vemula, P.: Self-stabilizing leader election in optimal space under an arbitrary scheduler. Theoretical Computer Science 412(40), 5541–5561 (2011)