Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Dominating the Stone Age

Master's Thesis

Nicolas Mattia

nmattia@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Dr. Jara Uitto, Sebastian Brandt
Prof. Dr. Roger Wattenhofer

October 1, 2015

Abstract

This work presents different results concerning the Stone Age model, a model of networked Finite State Machines. We first introduce some notations and representations to make the design of large protocols easier, and show some basic protocols. We then introduce two methods for determining which of two connected nodes has the largest degree. Using those results we show that it is possible to emulate the Distributed Greedy Minimal Dominating Set Algorithm in the Stone Age model, giving a good approximation of a Minimal Dominating Set. Finally we show that a path network of nodes in the Stone Age model is equivalent in power to a Linear Bounded Automaton. This allows us to show that the Stone Age model can be exponentially slower than the Local Model.

Contents

Chapter 1

Introduction

In this work we investigate properties of a special model of distributed computing, the *Stone Age model*. The Stone Age was introduced [1] to represent networks of machines that are simpler compared to that of the Local Model. Some interesting aspects of the Stone Age model are presented later in this chapter.

One important difference between the Stone Age model and the Local Model, for instance, is that the Stone Age model can only count its neighbors up to b, a constant parameter (explained below). This makes it interesting to figure out which of two connected nodes has the greatest number of neighbors. Counting neighbors in a network is equivalent to discovering the degree of a node in a graph, and thus in Chapter 2 we investigate two different ways of performing a degree test between two nodes. The first method will give us an exact answer, while the second method will give us a correct answer only with some probability.

The Maximum Independent Set (MIS) and Minimum Dominating Set (MDS) Problems are two connected, well known problems. A method for solving an MIS problem with a Stone Age network was already shown [1], and so it is interesting to ask the question "What can be done with the MDS problem in the Stone Age model?". In Chapter 3 we use the degree tests in order to implement a Minimum Dominating Set algorithm.

Finally, in Chapter 4 we show some properties of the Stone Age model's computational power. We start by proving the equivalence between a Linear Bounded Automaton and a specific Stone Age network (path network), and then use this result to show that even though the Stone Age model can solve some NP problems, it will reach exponential execution times. This is interesting in regard to the Local Model, which can only ever reach linear execution times.

1.1 The Stone Age Model

The Stone Age model is a model of networked (possibly randomized) finite state machines. The nodes communicate by transmitting messages belonging to some finite communication alphabet Σ such that a message $\sigma \in \Sigma$ transmitted by node

u is delivered to its neighbors (the same σ to all neighbors). The computation is anonymous, i.e., all nodes run the exact same protocol. In this work we only consider fully synchronous networks; moreover, w.l.o.g. we assume that each node can and will only send a message describing its current state. We will often refer to a node v receiving a message from a neighbor u as v seeing u's state.

The main limitation of the Stone Age model is that nodes can only count the number of appearances of each neighbor's state up to a *bounding parameter* $b \in \{0, 1, \ldots\}$. Any value larger than b cannot be distinguished from b. In this work, we assume the following:

- The model is applicable to arbitrary network topologies.
- All nodes run the same protocol executed by a (possibly randomized) finite state-machine.
- The network operates in a fully synchronous environment.
- All features of the FSM (specifically, the state set and bounding parameter) are of constant size independent of any parameter of the network (including the degree of the node executing the FSM).

A *round*, or *time step*, is processed as follows (for every node at the same time):

1. read the neighbors' states.
2. decide new state accordingly.
3. go to new state.

This means that a node can only read a neighbor's state that was decided at the previous time step.

1.2 Notation and Protocol Representation

It is not always convenient to spell out the state space and transition function of a given Stone Age protocol, especially when some mechanisms can be abstracted. Therefore we start by presenting a visual representation of a Stone Age protocol using diagrams, quite similar to that of Finite State Machines.

1.2.1 Protocol Representation

We represent the protocol as a graph, each state as a vertex in the graph, and each transition as an edge. The edge's label is the condition that must be fulfilled for that transition to *fire*, i.e., for the Stone Age node to enter the state pointed at by the edge. A condition on a transition *can* (but must not) be equivalent to querying the neighbors' state. When referring to a specific state, it is written in `teletype` font. A condition on the number of neighbors in a specific state is then represented as

$$|\texttt{STATE}| = x \ ,$$

where x is the number of neighbors that should be in state `STATE` for the transition to fire. We allow ϵ-transitions that will fire no matter what the states of the neighbor nodes are. *Accepting* states are circled twice. See Fig. 1.1 for an example. A transition that fires with probability $p = \frac{1}{x}$ is labeled with $\frac{\epsilon}{x}$, See Fig. 1.2.

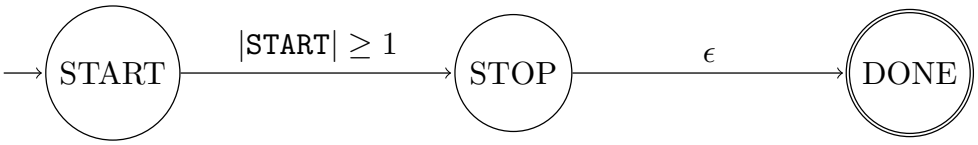


Figure 1.1: A simple Stone Age protocol. The node starts in state `START`, and goes to state `STOP` if any (one or more) of its neighbor is also in state `START`. If the node is in state `STOP` at time step τ, it will be in accepting state `DONE` at time step $\tau + 1$.

In order to reuse protocols, it is convenient to be able to include them in other diagrams. The name of a protocol is also written in `teletype` font. If `P1` is a protocol that contains a state `S1`, the state `S1` can be referred to outside of `P1` as `P1/S1`. We will sometimes call `P1` the *superstate* of `S1`. If in turn another protocol `P2` makes use of `P1` the aforementioned state `S1` can be referred to as `P2/P1/S1`. Sub-protocols are represented as a squared box. When a protocol `P1` is used in a protocol `P2`, the accepting states of `P1` act as outputs, and the box representing `P1` in `P2` should have one labeled edge for each output. The input of *sub-protocol* `P1` is `P1`'s initial state, see Fig. 1.3.

1.2.2 Protocol Execution

Now that we are able to represent protocols, we need to describe their execution too. We define some vocabulary and present a notation to simplify the execution of a protocol for a given node or a whole network.

Definition 1.1 (States and transitions)**.** Let u be a node and τ be a given time step. Let the state of a node u at time step τ be denoted by $s(u, \tau)$, and, if τ is

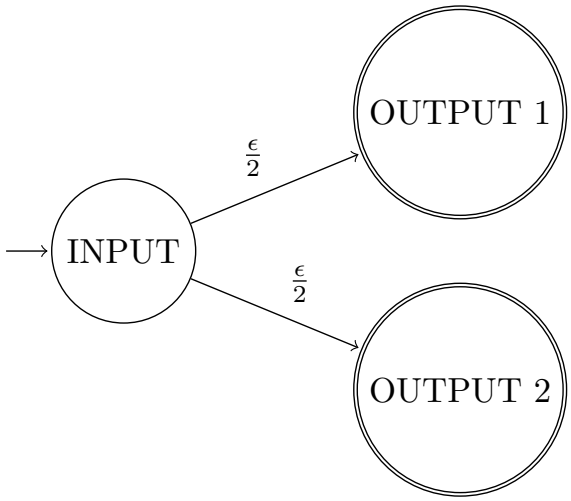


Figure 1.2: Protocol `RANDOM`, making use of randomness. It is equally likely that a node in state `INPUT` at time step τ will enter `OUTPUT 1` and `OUTPUT 2` at time step $\tau + 1$.

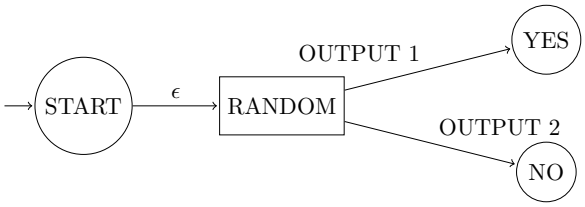


Figure 1.3: Example of a protocol using a *sub-protocol*. The first state entered is `START`, the second is `RANDOM/INPUT`.

clear from the context, let the state of the node u be denoted by $s(u)$. Also, let $\mathtt{A}$ be a state, and then:

- the node u *is in* state $\mathtt{A}$ at time step τ iff its state at τ is $\mathtt{A}$, and this is denoted by $u \overset{\tau}{-} \mathtt{A}$,
- the node u *enters* the state $\mathtt{A}$ at time step τ iff its state at τ is $\mathtt{A}$ and its state at $\tau - 1$ was not $\mathtt{A}$, and this is denoted by $u \xrightarrow{\tau} \mathtt{A}$,
- the node u *leaves* the state $\mathtt{A}$ at time step τ iff its state at τ is not $\mathtt{A}$ but its state at $\tau - 1$ was $\mathtt{A}$, and this is denoted by $u \xleftarrow{\tau} \mathtt{A}$.

This can be formally written as follows:

$$
\begin{aligned}
u \overset{\tau}{-} \mathtt{A} &\iff s(u,\tau) = \mathtt{A} \\
u \xrightarrow{\tau} \mathtt{A} &\iff (s(u,\tau) = \mathtt{A}) \wedge (s(u,\tau-1) \neq \mathtt{A}) \\
u \xleftarrow{\tau} \mathtt{A} &\iff (s(u,\tau) \neq \mathtt{A}) \wedge (s(u,\tau-1) = \mathtt{A}) \ .
\end{aligned}
$$

Let V be any set of nodes and $\mathtt{S}$ be any set of states. Then let the notion of *being* (respectively *entering*, *leaving*) be extended to

- the node u *being in* (respectively *entering*, *leaving*) any state $\mathtt{s} \in \mathtt{S}$, denoted by $u \overset{\tau}{-} \mathtt{S}$ (respectively $u \xrightarrow{\tau} \mathtt{S}$, $u \xleftarrow{\tau} \mathtt{S}$),
- every node $v \in V$ *being in* (respectively *entering*, *leaving*) the state $\mathtt{A}$, denoted by $V \overset{\tau}{-} \mathtt{A}$ (respectively $V \xrightarrow{\tau} \mathtt{A}$, $V \xleftarrow{\tau} \mathtt{A}$),
- every node $v \in V$ *being in* (respectively *entering*, *leaving*) any state $\mathtt{s} \in \mathtt{S}$, denoted by $V \overset{\tau}{-} \mathtt{S}$ (respectively $V \xrightarrow{\tau} \mathtt{S}$, $V \xleftarrow{\tau} \mathtt{S}$).

This can be formally written as

$$
\begin{aligned}
u \xrightarrow{\tau} \mathtt{S} &\iff (s(u,\tau) \in \mathtt{S}) \wedge (s(u,\tau-1) \notin \mathtt{S}) \\
u \xleftarrow{\tau} \mathtt{S} &\iff (s(u,\tau) \notin \mathtt{S}) \wedge (s(u,\tau-1) \in \mathtt{S}) \\
V \xrightarrow{\tau} \mathtt{A} &\iff (s(v,\tau) = \mathtt{A}) \wedge (s(v,\tau-1) \neq \mathtt{A}), \forall v \in V \\
V \xleftarrow{\tau} \mathtt{A} &\iff (s(v,\tau) \neq \mathtt{A}) \wedge (s(v,\tau-1) = \mathtt{A}), \forall v \in V \\
V \xrightarrow{\tau} \mathtt{S} &\iff (s(v,\tau) \in \mathtt{S}) \wedge (s(v,\tau-1) \notin \mathtt{S}), \forall v \in V \\
V \xleftarrow{\tau} \mathtt{S} &\iff (s(v,\tau) \notin \mathtt{S}) \wedge (s(v,\tau-1) \in \mathtt{S}), \forall v \in V \ .
\end{aligned}
$$

1.2.3 Neighborhoods

Finally, we present the definition and notations we use for the notion of a *neighborhood* in a Stone Age network.

Definition 1.2 (Neighborhoods)**.** Let x be a node. Then let $\hat{N}(x)$ denote the *inclusive neighborhood* of x, i.e. the set of all nodes at distance at most 1 from x, including x itself. For simplicity, $N(x)$ is defined as the *exclusive neighborhood* of x (or simply *neighborhood* of x) i.e.

$$N(x) = \hat{N}(x) \setminus \{x\} .$$

Let then $\hat{N}_2(x)$ be defined as the 2-*hop inclusive neighborhood* of x, the set of all nodes at distance at most 2 from x, including x itself, i.e.

$$\hat{N}_2(x) = \bigcup_{u \in \hat{N}(x)} \hat{N}(u) ,$$

and for simplicity let the 2-*hop exclusive neighborhood* of x be defined as

$$N_2(x) = \hat{N}_2(x) \setminus \{x\} .$$

Let y be another node. Then let $N(x, y)$ be defined as the *exclusive common neighborhood* of x and y, namely

$$N(x, y) = (N(x) \cup N(y)) \setminus \{x, y\} .$$

More generally, let the *exclusive neighborhood* $N(U)$ of a set U of nodes be defined as

$$N(U) = \bigcup_{b \in U} N(x) \setminus U .$$

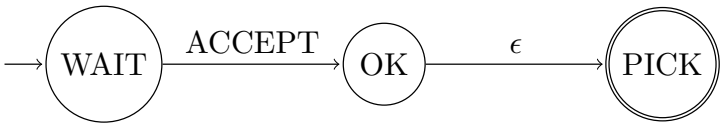
1.3 Useful Protocols

In this section we present some general purpose protocols that will prove useful in Chapters 2 and 3.

1.3.1 Singling Out Nodes

As Stone Age nodes are not allowed to have unique identifiers (i.e. they all run the exact same protocol), singling out a subset of a node's neighbors (such that the subset may respond differently to given messages) is not straightforward. We present a randomized protocol which alleviates the issue, based on a master/slave model. The *master* node m is set and needs to single out a number n_A of

neighbors (*slaves*) out of a given subset N_s of its neighbors (the number n_A can be at most b, the bounding parameter). The master protocol is presented in Fig. 1.4 and the slave protocol, run by all nodes in N_s, is presented in Fig. 1.5. We use *PARAMETERS* as a means to refer to something defined outside of the protocol, possibly different for every use of the protocol.



PARAMETERS: *slave*, *master*, n_A
$ACCEPT \iff |slave/\texttt{SINGLEs/TRY A}| = n_A$

Figure 1.4: Singling out protocol, master (`SINGLEm`).

Explanation. Nodes executing the slave protocol `SINGLEs` will go back and forth at random between two states `SINGLEs/TRY A` and `SINGLEs/TRY B`. Only when the master node executing the protocol `SINGLEm` realizes that exactly n_A nodes are in state `SINGLEs/CHECK A` will it send a `SINGLEm/OK` message.

Let m be a node in any superstate A_m, and let $N_s \subset N(m)$ be a subset of m's neighbors from which we want to single out $n_A \leq |N_s|$ nodes; assume moreover that every node in N_s is in a superstate A_s.

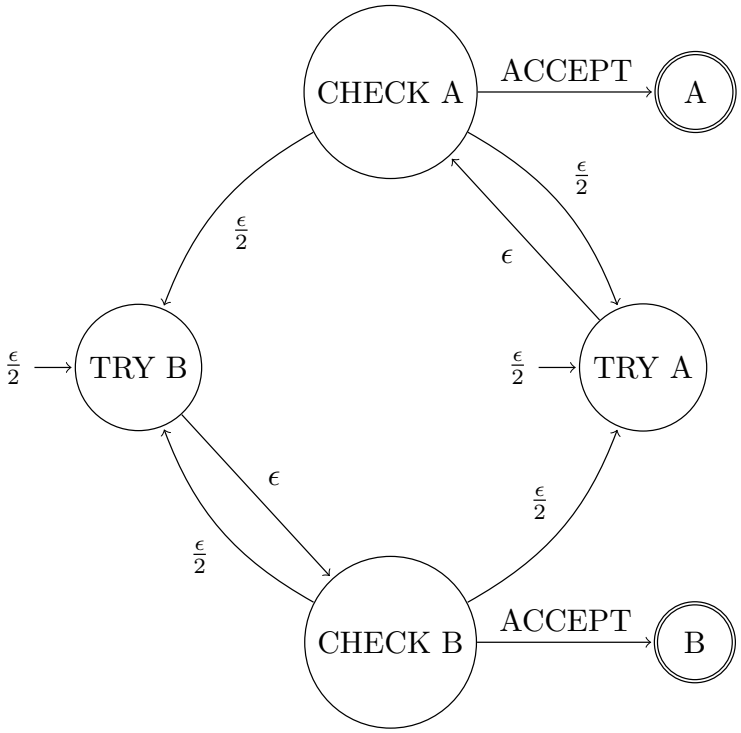
We assume that, at some time step τ_s, m enters the protocol `SINGLEm`, and the set N_s enters the protocol `SINGLEs`, i.e.

$$m \xrightarrow{\tau_s} A_m/\texttt{SINGLEm}(A_s, A_m, n_A)/\texttt{WAIT}$$
$$N_s \xrightarrow{\tau_s} \texttt{TRY}$$

with

$$\texttt{TRY} = \{A_s/\texttt{SINGLEs}(A_s, A_m)/\texttt{REST/TRY A}, \\ A_s/\texttt{SINGLEs}(A_s, A_m)/\texttt{REST/TRY B}\} .$$

Finally, we assume that there is some time step τ_f at which m enters the state `PICK` and all nodes in N_s enter either `A` or `B`, and that for the given time interval



PARAMETERS: *slave*,*master*
ACCEPT $\iff$ $|master/\texttt{SINGLEm/OK}| = 1$

Figure 1.5: Singling out protocol, slave (`SINGLEs`).

$[\tau_s, \tau_f]$ no other node in the neighborhood of N_s enters the master protocol, i.e.

$$\text{there are no } x, \tau \text{ with } x \in N(N_s), \tau \in [\tau_s, \tau_f] \text{ s.t. } x \xrightarrow{\tau} \texttt{M} \,.$$

For simplicity, we denote the slave protocol by `S`, such that

$$\texttt{S} = A_s\texttt{/SINGLEs}(A_s, A_m) \,.$$

Then, in order to show the protocol works as intended, we need the following observation:

Observation 1.3. *At any time, the nodes in N_s are either all* trying*, all* checking *or have reached an accepting state; in other words, for all $\tau \in [\tau_0, \tau_f]$ exactly one of the following is true:*

- $\bigcup_{x \in N_s} s(x) \subseteq \{\texttt{S/TRY A}, \texttt{S/TRY B}\}$
- $\bigcup_{x \in N_s} s(x) \subseteq \{\texttt{S/CHECK A}, \texttt{S/CHECK B}\}$
- $\bigcup_{x \in N_s} s(x) \subseteq \{\texttt{S/A}, \texttt{S/B}\}$.

Lemma 1.4 (Single-Out Lemma)**.** *At time step τ_f, exactly n_A nodes of N_s will enter `A` and the $|N_s| - n_A$ other nodes will enter `B`, i.e. there exists τ_f with $\tau_s < \tau_f < \infty$ such that*

$$\begin{aligned} m &\xrightarrow{\tau_f} A_m\texttt{/SINGLEm}(A_s, A_m, n_A)\texttt{/PICK} \\ N_{\texttt{A}} &\xrightarrow{\tau_f} A_s\texttt{/SINGLEs}(A_s, A_m)\texttt{/A} \\ N_{\texttt{B}} &\xrightarrow{\tau_f} A_s\texttt{/SINGLEs}(A_s, A_m)\texttt{/B} \end{aligned}$$

with $N_{\texttt{A}}, N_{\texttt{B}}$ two sets of nodes such that

$$\begin{aligned} N_{\texttt{A}} \cup N_{\texttt{B}} &= N_s \\ N_{\texttt{A}} \cap N_{\texttt{B}} &= \emptyset \\ |N_{\texttt{A}}| &= n_{\texttt{A}} \,. \end{aligned}$$

Proof. The accepting condition at time step τ of the state `WAIT`, $ACCEPT(\tau)$, asks for exactly n_A nodes from N_s to be in the state `S/TRY A`. Those n_A nodes can be chosen as some $\tilde{N}_A$, and the condition can then be rewritten as $\exists \tilde{N}_{\texttt{A}}, \tilde{N}_{\texttt{B}}$ with $\tilde{N}_{\texttt{A}} \subseteq N_s, \tilde{N}_{\texttt{B}} = N_s \setminus \tilde{N}_{\texttt{A}}$ and $|\tilde{N}_{\texttt{A}}| = n_A$ such that

$$\forall x \in \tilde{N}_{\texttt{A}}, x \overset{\tau}{-} \texttt{S/TRY A}$$

$$\nexists y \in \tilde{N}_{\texttt{B}}, y \overset{\tau}{-} \texttt{S/TRY A} \,.$$

If all nodes of $\tilde{N}_A$ are in state `S/TRY A` it holds by Observation 1.3 that all nodes in $\tilde{N}_B$ must be in state `S/TRY B`. This means that if τ_f is almost reached (or rather that the nodes are in an accepted configuration), the nodes in N_s will be split into two groups $\tilde{N}_A, \tilde{N}_B$ actually satisfying the conditions on N_A, N_B such that

$$N_A = \tilde{N}_A$$
$$N_B = \tilde{N}_B \ .$$

Then, it is clear that

$$N_s \xrightarrow{\tau_f} \{\texttt{S/A}, \texttt{S/B}\} \iff m \xrightarrow{\tau_f - 1} \texttt{M/OK} \iff ACCEPT(\tau_f - 2) \text{ holds}$$

and moreover N_A and N_B are defined and exist. Since each configuration of nodes in state `TRY` at step $\tau = 2\tau'$ is independent of the previous ones and all configurations have equal probabilities, there exist τ_f and $\Delta\tau = \tau_f - \tau_0$, where $\Delta\tau$ follows a geometric distribution

$$\Delta\tau \sim (1-p)^{\Delta\tau - 1} p$$

with parameter $p = \frac{\binom{n_s}{n_a}}{2^{n_s}}$, $n_s = |N_s|$, and thus it holds that $\tau_f < \infty$ w.p. 1.

□

1.3.2 Relaying Messages

Let three nodes u, v and w be connected by the edges (u, v) and (v, w). For a given duration $[\tau_s, \tau_f]$ we want u and w to be able to read each other's state as if they were connected by an edge (u, w). In Lemma 1.5 we prove the existence of a protocol `RELAY` that does just that, given that a few conditions are met, and that the notion of time step is extended.

First, we assume that the following holds for the time interval $[\tau_s, \tau_f]$:

- u never leaves a given (finite) state set S_u, and w never leaves another given (also finite) state set S_w disjoint from S_u
- no other neighbor of v than u and w enters any state of S_u or S_w.

In other words, we define τ_s and τ_f as follows

$$u \xrightarrow{\tau_s} S_u$$
$$w \xrightarrow{\tau_s} S_w$$
$$u \xleftarrow{\tau_f} S_u$$
$$w \xleftarrow{\tau_f} S_w$$

for given S_u, S_w with

$$S_u \cup S_w = \emptyset$$

and so that it holds that there is no $\tau \in [\tau_s, \tau_f]$ such that

$$u \xleftarrow{\tau} S_u$$
$$w \xleftarrow{\tau} S_w$$

or such that, for any x in $N(v)$ with $x \neq v, w$

$$x \xrightarrow{\tau} S_u \cup S_v \ .$$

Finally, let w' and u' be two neighboring nodes. When the node w' enters a state at time-step τ, u' is able to read it from its port $p_{u'}(w')$ at time step $\tau + 1$. A similar behavior can be simulated for w and u by allowing the relay node v to change states twice as fast as the other nodes: if the node w enters a state at time step τ, we allow the relay node v to read it from its port $p_v(w)$ at time step τ', with $\tau < \tau' < \tau + 1$, allowing v to report the node w's state at time step τ' already.

Note that we don't need a new description for the finite automaton. The process explained above can be implemented by simply doubling every state in every protocol, except for the relay protocol.

Lemma 1.5 (Relay node)**.** *There exists a protocol* `RELAY` *of constant size such that if v has a* clock *twice as fast as that of both u and w, then u can infer $p_v(w)$ by reading $p_u(v)$ and w can infer $p_v(u)$ by reading $p_w(v)$.*

Proof. The protocol `RELAY` is $S_{\mathtt{R}} = S_u \times S_w$ with transitions $\delta(p_v(u), p_v(w)) = (p_v(u), p_v(w))$. At any *slow* time step (s_u, s_w) is in both ports $p_u(v)$ and $p_w(v)$. Since $S_u \cup S_w = \emptyset$, it is clear from $p_x(v), x \in \{u, w\}$ what state the other node (u or w) is in. Also, $|S_{\mathtt{R}}| = |S_u||S_w|$ is constant.

□

Chapter 2

Degree Test Protocols

In this chapter we present two protocols for determining which of two given nodes has the greater number of neighbors.

2.1 Slow Highest-Degree Test

We consider two connected nodes comparing their degrees and executing the Slow Highest-Degree (`SHD`) protocol found in Fig. 2.1 (with a given protocol `P` we use the notation `*/P` to refer to `P` from *any* superstate, and `P/*` to refer to *any* state for which `P` is a superstate). Every time either of those two nodes singles out one of its neighbors, it waits until the other node has also singled out one of its own neighbors. Once it is done, both start over, each excluding its newly singled out neighbor. The protocol stops once a node has run out of neighbors to single out. To avoid draws, we define one node to have priority over the other one. The neighbors of the nodes being tested must execute protocol `NEIGHBOR SHD` found in Fig. 2.2.

Let two nodes u and v be at distance 1 from one another, and be in any two (possibly same) superstates A_U and A_V. Let then u and v both enter the Protocol `SHD` at the same time step τ_s, i.e.

$$u \xrightarrow{\tau_s} A_U/\texttt{SHD}(true)$$
$$v \xrightarrow{\tau_s} A_V/\texttt{SHD}(false)\ .$$

Then we assume that they both leave the protocol at some time step τ_f in the future, i.e.

$$u \xleftarrow{\tau_f} A_U/\texttt{SHD}(true)$$
$$v \xleftarrow{\tau_f} A_V/\texttt{SHD}(false)$$

with $\tau_f > \tau_s$. Finally, let u and v both have a set of neighboring nodes N_u and N_v, which we call the *considered neighbors* of u or v, that enter Protocol `NEIGHBOR SHD` (`NEIGHBOR`) at time step τ_s, which themselves cannot have

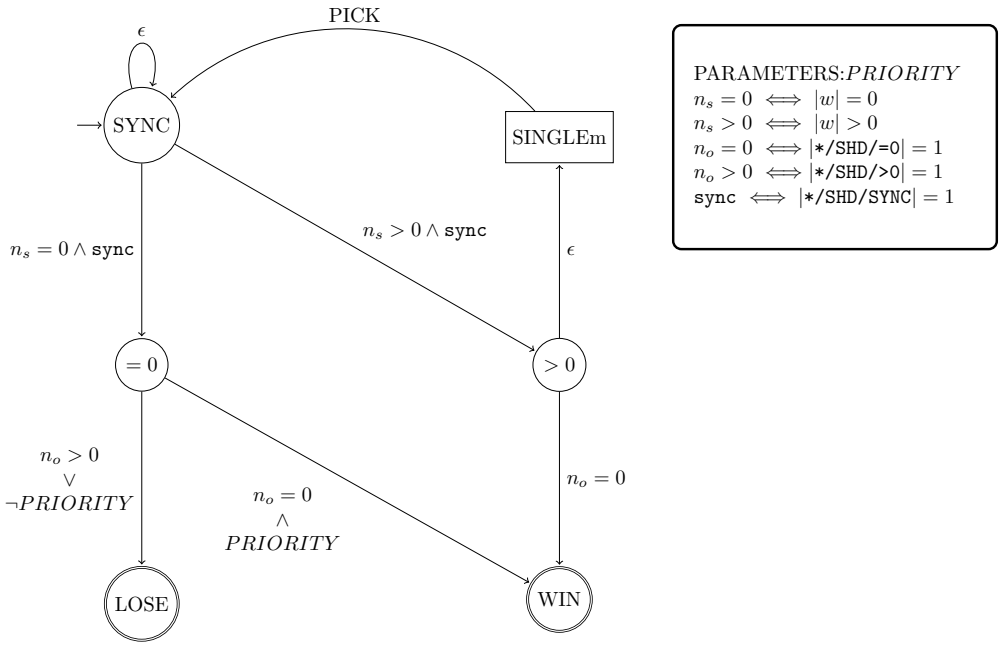


Figure 2.1: Protocol executed by an SHD testing node (`SHD`).

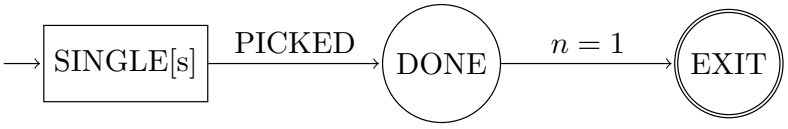


Figure 2.2: Protocol executed by the neighbors of an SHD testing node (`NEIGHBOR SHD`).

any other neighbors than u and v in states A_U/SHD($true$) and A_V/SHD($false$), respectively. That is, for given

$$N_u \subseteq N(u)$$
$$N_v \subseteq N(v)\ ,$$

then

$$N_u \xrightarrow{\tau_s} \texttt{NEIGHBOR}$$
$$N_v \xrightarrow{\tau_s} \texttt{NEIGHBOR}$$

and for all $\tau \in [\tau_s, \tau_f]$ it must hold that there are no nodes $x \in N_2(u)$, $y \in N_2(v)$ such that

$$x \overset{\tau}{-} A_U\texttt{/SHD(}true\texttt{)}$$
$$y \overset{\tau}{-} A_V\texttt{/SHD(}false\texttt{)}\ .$$

Lemma 2.1 (Slow Highest-Degree Test)**.** *In finite time w.p. 1, exactly one node of u and v will exit with* WIN *and exactly one node will exit with* LOSE*; moreover either u has a number of considered neighbors greater than or equal to that of v and u exits with* WIN *or v has a number of considered neighbors strictly greater than that of u and u exits with* LOSE*, i.e.*

$$u \xrightarrow{\tau_f} A_U\texttt{/SHD(}true\texttt{)/WIN} \text{ and } v \xrightarrow{\tau_f} A_V\texttt{/SHD(}false\texttt{)/LOSE} \iff |N_u| \geq |N_v|$$

and

$$u \xrightarrow{\tau_f} A_U\texttt{/SHD(}true\texttt{)/LOSE} \text{ and } v \xrightarrow{\tau_f} A_V\texttt{/SHD(}false\texttt{)/WIN} \iff |N_u| < |N_v|$$

hold, and moreover $\tau_s < \tau_f < \infty$.

Proof. We use $N_u(\tau)$ and $N_v(\tau)$ to denote the considered neighbors of u, respectively v, that have yet to be *accounted for* at time step τ. We say that a considered neighbor is accounted for if it has entered N2/DONE, i.e.

$$N_u(\tau) = \{x \mid x \in N_u, s(x,\tau) \notin \texttt{*/N2/DONE}\}$$
$$N_u(\tau) = \{x \mid x \in N_u, s(x,\tau) \notin \texttt{*/N2/DONE}\}\ .$$

Also, for simplicity, we denote the number of considered neighbors to be accounted for by $n_u(\tau) = |N_u(\tau)|$ and $n_u(\tau) = |N_u(\tau)|$.

Then note that if at any time step both u and v are in their respective SYNC state, then they will both leave the state SYNC at the following time step:

$$u, v \overset{\tau}{-} \texttt{SYNC} \iff u, v \xleftarrow{\tau+1} \texttt{SYNC}\ .$$

Since SYNC is the initial state of SHD, both u and v are in state SYNC when the protocol starts, i.e.

$$u, v \overset{\tau_s}{=} \texttt{SYNC} \,. \tag{2.1}$$

Note that iff u and v are in state SYNC and have neighbors to be accounted for then they will, two time steps after leaving SYNC, enter the protocol SINGLEm. When they leave SINGLEm they return to SYNC, u will have singled out one of the nodes in N_u, and v one of the nodes in N_v; by Lemma 1.4 exactly one of the neighbors that were in SINGLEs will now be in DONE, and both u and v will now have a neighbor less to be accounted for. This means that the difference of neighbors to be accounted for between u and v will remain constant, and from (2.1) it follows that this difference is also equal to the difference between u and v's number of considered neighbors.

In other words, let τ_i be the i^{th} time that both u and v are in state SYNC. Then if such a τ_i exists it follows from (2.1) that

$$n_u(\tau_i) - n_v(\tau_i) = w(u) - w(v) \tag{2.2}$$

and moreover, if $n_u(\tau_i) > 0, n_v(\tau_i) > 0$ then it also holds that

$$\exists \tau_{i+1} \text{ such that } u \overset{\tau_{i+1}}{=} \texttt{SYNC and } y \overset{\tau_{i+1}}{=} \texttt{SYNC} \tag{2.3}$$

and

$$n_u(\tau_{i+1}) = n_u(\tau_i) - 1, n_v(\tau_{i+1}) = n_v(\tau_i) - 1 \,. \tag{2.4}$$

Clearly, there exists a τ_1, and moreover $\tau_1 = \tau_s$, since both u and v entered the protocol (and thus SYNC) at the same time step.

This will *consume* all the considered neighbors of u and v, always making sure a neighbor has been consumed on both u and v before consuming the next one. Eventually, either u or v's number of neighbors to be accounted for will reach zero. Since u has been defined with $PRIORITY$ (from entering SHD($true$)), it will reach WIN even if its span is equal to that of v. We define

$$z = \min\{w(u), w(v)\} + 1$$

to be the minimal span *plus one*. Describing the total consumption of u of v's considered neighbors amounts is as follows: by equations (2.3) and (2.4) there exists τ_z with

$$\min\{n_u(\tau_z), n_v(\tau_z)\} = 0$$

and thus there exists a τ_f with $\tau_f = \tau_z + 1$. Lemma 1.4 showed `SINGLEm` and `SINGLEs` operate within finite time, and thus for any τ_i it holds that $\tau_i < \infty$ and thus

$$\tau_f = (\tau_2 - \tau_1) + (\tau_3 - \tau_2) + \cdots + (\tau_z - \tau_{z-1}) = \tau_z - \tau_1 < \infty \ .$$

□

2.2 Fast Highest Degree Test

We now present an alternative degree test. This protocol determines the highest degree node only with some probability (dependent on both nodes' degrees) but does not rely on the singling out protocol. Since the nodes do not need to single out each neighbor, the execution is faster.

At each time step, each neighbor of the two tested nodes might quit with probability $\frac{1}{2}$. The first tested node that has no neighbor left loses. Moreover, if at any point both nodes are left without neighbors, the game starts over, this time starting from the last configuration of neighbors before the draw. An implementation for the tested nodes and neighbor nodes are given in Fig. 2.3 and Fig. 2.4 respectively.

We now assume that two nodes u and v with respective degree δ_u and δ_v enter the protocol `FHD` while their neighbors enter the protocol `NEIGHBOR FHD`. The rest of the section is dedicated to the analysis of the protocol, especially proving the following result:

Theorem 2.2. *Exactly one node of u and v exits with `WIN` and exactly one node exits with `LOSE`. Moreover, the probability of the node with the highest degree to exit with `WIN` is*

$$\frac{\max\{\delta_u, \delta_v\}}{\delta_u + \delta_v} \ .$$

We will first show that the execution of the protocol can be modeled as a Discrete-Time Markov Chain. Then we will prove Theorem 2.2 using the Markov Chain's hitting probabilities.

Given both δ_u and δ_v, representing respectively u and v's number of neighbors, we first define C, the space of any given configuration of the number of neighbors of u and v which are up:

$$C = \{0, 1, ..., \delta_u\} \times \{0, 1, ..., \delta_v\} \ .$$

Now, we can represent a configuration (a, b) preceded by a configuration (a', b') and denote it by

$$\langle a, b \,|\, a', b' \rangle \in C \times C$$

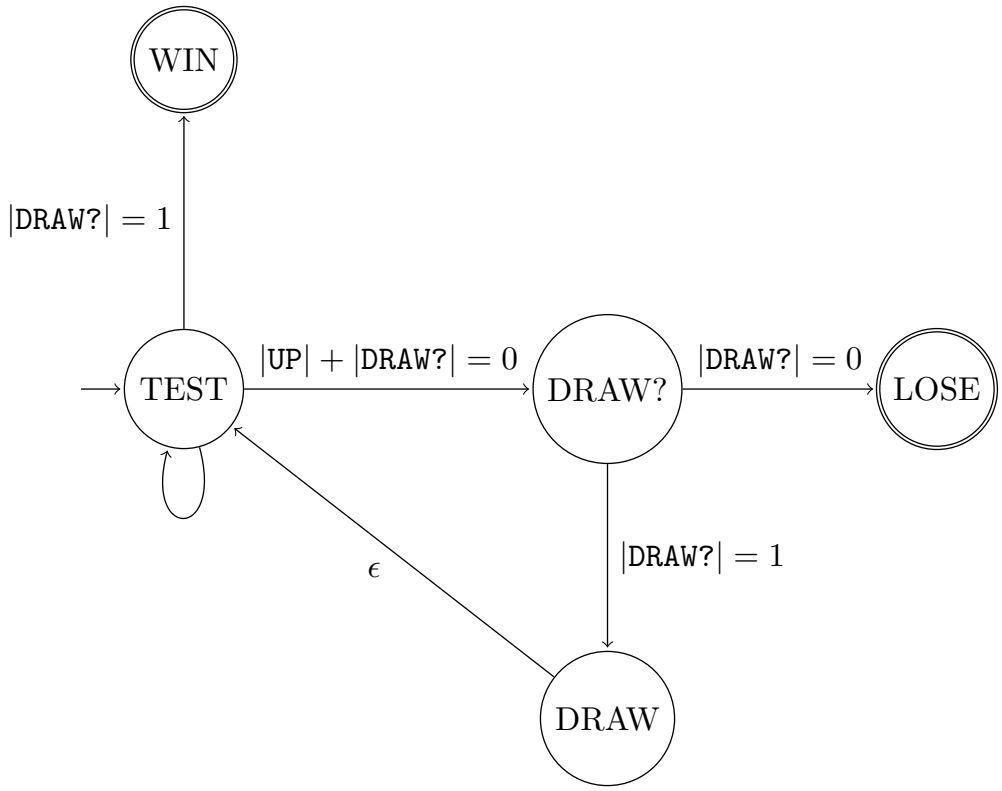


Figure 2.3: Protocol executed by an FHD testing node (FHD).

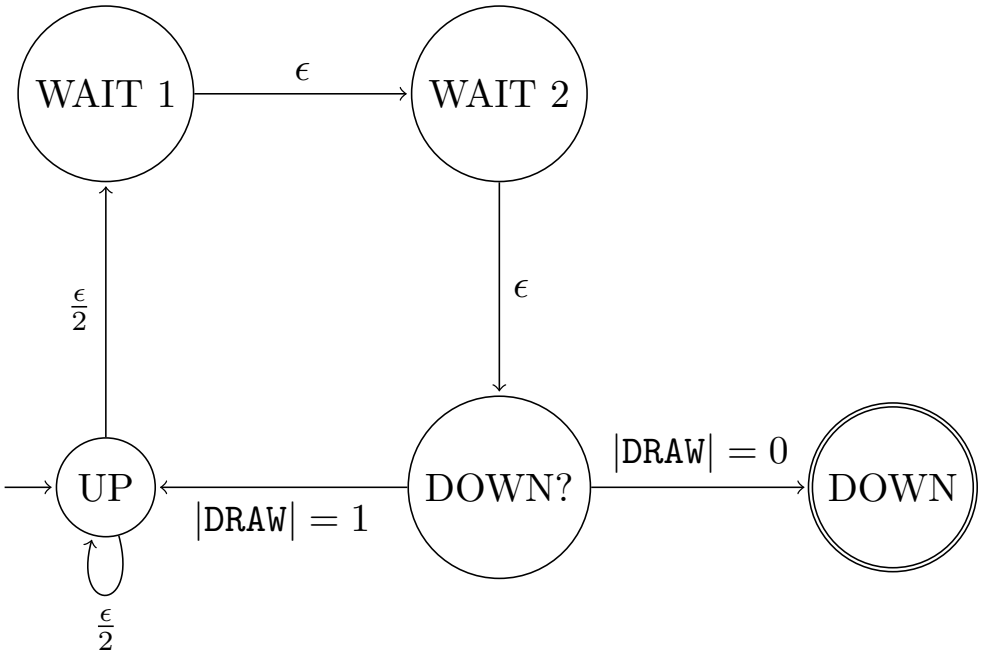


Figure 2.4: Protocol executed by the neighbors of an FHD testing node (`NEIGHBOR FHD`).

and define the Markov Chain M with the state set

$$Q = \{\langle a, b \,|\, a', b' \rangle \mid 0 \leq a, a' \leq \delta_u \text{ and } 0 \leq b, b' \leq \delta_v\}$$

where we allow the short-hands

$$\langle a, b| = \langle a, b \,|\, a', b' \rangle \text{ if } a \neq 0 \text{ and } b \neq 0$$
$$|a', b' \rangle = \langle 0, 0 \,|\, a', b' \rangle \text{ if } a' \neq 0 \text{ and } b' \neq 0 \,.$$

Let $s_1 = \langle a_1, b_1 \,|\, a_1', b_1' \rangle$ and $s_2 = \langle a_2, b_2 \,|\, a_2', b_2' \rangle$ with $s_1, s_2 \in C \times C$ be two states at any given time; then, we denote the probability of going from state s_1 to state s_2 by $p(s_1, s_2)$ and define it as follows:

Draw. If at any point neither u nor v has any neighbor up, we start a new game with the latest configuration. This is expressed as

$$p\big(\, |a, b\rangle \,, \langle a, b| \,\big) = 1$$

Victory. If at any point one node has a strictly positive number of neighbors up x while the other one does not, the game stops. This is expressed as

$$p\big(\, \langle x, 0| \,, s\big) = p\big(\, \langle 0, x| \,, s\big) = 0$$

Reduction step. Let the current configuration be $\langle a, b|$, with $a \neq 0$ and $b \neq 0$. Each node will simply turn each of its neighbors off w.p. $\frac{1}{2}$.

$$p(\langle a', b'|, \langle a, b \,|\, a', b' \rangle) = \frac{1}{2^{a'+b'}} \binom{a'}{a} \binom{b'}{b}$$

Observation 2.3. *The probability of reaching a winning state for u from the state $\langle a, b|$ with $a > 0, b > 0$ is*

$$h(a, b) = \frac{H(a, b)}{2^{a+b} - 2} \tag{2.5}$$

with

$$H(a, b) = \sum_{j=1}^{b} \binom{b}{j} h(a, b-j) + \sum_{i=1}^{a-1} \sum_{j=0}^{b} \binom{a}{i} \binom{b}{j} h(a-i, b-j) \; .$$

Proof. We use the fact that, for any DTMC, the hitting probability h_s^A of reaching a set of states A from state s is[2]

$$\begin{cases} h_s^A = 1 & \text{for } s \in A \\ h_s^A = \sum_{s' \in Q} p_{ss'} h_{s'}^A & \text{for } s \notin A \; . \end{cases}$$

The negative term in the denominator of Equation 2.5 comes from the fact that there is a probability of exactly $\frac{1}{2^{a+b}}$ of staying in state $\langle a, b|$ and a probability of exactly $\frac{1}{2^{a+b}}$ of reaching the state $\langle 0, 0|$ which itself leads back to $\langle a, b|$ w.p. 1, as defined for a draw. Note also that since $\langle a, 0|$ is a winning state for u, $h(a, 0)$ is defined and moreover $h(a, 0) = 1$.

□

Before going further, we need to introduce the following result:

Lemma 2.4. *For all $x, y > 0$*

$$\sum_{i=0}^{x} \sum_{j=0}^{y} \binom{x}{i} \binom{y}{j} \frac{x + y + 1 - 2(i + j)}{x + y + 1 - (i + j)} = 1 \; . \tag{2.6}$$

The proof of Lemma 2.4 is deferred to the end of the section due to its length.

Lemma 2.5. *Given the above Markov Chain M, the probability of hitting a state $\langle a', 0|$, with $a' \neq 0$, from any state $\langle a, b|$ is*

$$h(a, b) = \frac{a}{a + b} \; .$$

Proof. We will prove this result by induction. The result trivially holds for $h(a,0)$ since, as noted above

$$h(a,0) = 1 = \frac{a}{a+0} .$$

From Equation 2.5 it holds that

$$h(a+1,b) = \frac{H(a+1,b)}{2^{a+b+1}-2} \tag{2.7}$$

with

$$H(a+1,b) = \\ \sum_{j=1}^{b} \binom{b}{j} h(a+1,b-j) + \sum_{i=1}^{a}\sum_{j=0}^{b} \binom{a+1}{i}\binom{b}{j} h(a+1-i,b-j) .$$

Assuming that $h(a',b') = \frac{a'}{a'+b'}$ for all a',b' with $0 < a' \leq a+1, 0 < b' \leq b$ (excluded of course that both $a' = a+1$ and $b' = b$) we can write

$$\begin{aligned} H(a+1,b) &= \sum_{j=1}^{b} \binom{b}{j} \frac{a+1}{a+1+b-j} + \sum_{i=1}^{a}\sum_{j=0}^{b} \binom{a+1}{i}\binom{b}{j} \frac{a+1-i}{a+1-i+b-j} \\ &= \sum_{j=0}^{b} \binom{b}{j} \frac{a+1}{a+1+b-j} + \sum_{i=0}^{a}\sum_{j=0}^{b} \binom{a+1}{i}\binom{b}{j} \frac{a+1-i}{a+1-i+b-j} \\ &\quad - \binom{b}{0} \frac{a+1}{a+1+b-0} - \sum_{j=0}^{b} \binom{a+1}{0}\binom{b}{j} \frac{a+1-0}{a+1-0+b-j} \\ &= \sum_{i=0}^{a}\sum_{j=0}^{b} \binom{a+1}{i}\binom{b}{j} \frac{a+1-i}{a+1-i+b-j} - \frac{a+1}{a+1+b} . \end{aligned}$$

We can now rewrite $H(a+1,b)$ as

$$H(a+1,b) = \frac{a+1}{a+b+1}(U-1) \tag{2.8}$$

with

$$\begin{aligned} U &= \sum_{i=0}^{a}\sum_{j=0}^{b} \binom{a+1}{i}\binom{b}{j} \frac{a+1-i}{a+1-i+b-j} \frac{a+b+1}{a+1} \\ &= \sum_{i=0}^{a}\sum_{j=0}^{b} \binom{a}{i}\binom{b}{j} \frac{a+b+1}{a+1-i+b-j} , \end{aligned}$$

where we used the fact that

$$\binom{a+1}{i} = \frac{(a+1)!}{(a+1-i)!i!} = \frac{a+1}{a+1-i}\frac{a!}{(a-i)!i!} = \frac{a+1}{a+1-i}\binom{a}{i} .$$

Using Lemma 2.4 we can now rewrite U as

$$\begin{aligned} U &= \sum_{i=0}^{a}\sum_{j=0}^{b}\binom{a}{i}\binom{b}{j}\frac{2(a+b+1-i-j)-(a+b+1-2(i+j))}{a+1-i+b-j} \\ &= \sum_{i=0}^{a}\sum_{j=0}^{b}\binom{a}{i}\binom{b}{j}\left[2-\frac{a+b+1-2(i+j)}{a+b+1-(i+j)}\right] \end{aligned}$$

and use both Lemma 2.4 and the fact that $\sum_{i=0}^{a}\sum_{j=0}^{b}\binom{a}{i}\binom{b}{j} = 2^{a+b}$ [2] to write

$$U = 2^{a+b+1} - \sum_{i=0}^{a}\sum_{j=0}^{b}\binom{a}{i}\binom{b}{j}\frac{a+b+1-2(i+j)}{a+b+1-(i+j)} = 2^{a+b+1}-1\,. \tag{2.9}$$

Now substituting equation 2.9 in equations 2.7 and 2.8

$$h(a+1,b) = \frac{a+1}{a+b+1}\frac{U-1}{2^{a+b+1}-2} = \frac{a+1}{a+b+1}\frac{2^{a+b+1}-1-1}{2^{a+b+1}-2} = \frac{a+1}{a+b+1}$$

which completes the proof of Lemma 2.5. □

Proof of Theorem 2.2. Since the Markov Chain M represents the protocol's execution, and since the hitting probability for a node to win is (as proved in 2.5)

$$h(a,b) = \frac{a}{a+b}$$

it is clear that the probability for the node with the highest degree to win is

$$\frac{\max\{\delta_u,\delta_v\}}{\delta_u+\delta_v}\,,$$

which concludes the proof.

□

2.2.1 Outlook

There are different ways the Fast Highest Degree method could be improved or modified. Firstly, two nodes competing against one another could enter an infinite loop if they share a neighbor. This happens if a shared neighbor is the last standing neighbor, leading to a draw. Upon restarting the protocol with last configuration, the shared neighbor will be turned on again, and the situation will

keep repeating. One way to alleviate the issue is to restart the protocol with the initial configuration, i.e., all neighbors up. The protocol will be (only slightly) slower in expectation, but will also give a better answer in expectation.

Secondly, there could be more than two nodes competing at the same time. To achieve this, the extra bit of information stored by a *neighbor* node could be added to the master protocol. Several node could then be executing this new protocol, taking over the role of both the master and the slave (testing and neighbor): every node "turns off" with probability $\frac{1}{2}$ (tells its neighbors that it won't be supporting as a neighbor anymore) at every time step, and stops competing when all its neighbors have "turned off".

2.2.2 Proof of Lemma 2.4

Proof of Lemma 2.4. Let's first rewrite the left hand side of equation 2.6 as

$$K = \sum_{i=0}^{x}\sum_{j=0}^{y}\binom{x}{i}\binom{y}{j}\frac{x+y+1-2(i+j)}{x+y+1-(i+j)} = \sum_{i=0}^{x}\sum_{j=0}^{y}\binom{x}{i}\binom{y}{j}\sigma(i+j)$$

with

$$\sigma(k) = \frac{x+y+1-2k}{x+y+1-k}$$

which, along with defining $w = x + y$, allows us to express K as

$$K = \sum_{z=0}^{w}\left(\sum_{i+j=z}\binom{x}{i}\binom{y}{j}\right)\sigma(z) \tag{2.10}$$

$$= \sum_{z=0}^{w}\binom{w}{z}\sigma(z)\,. \tag{2.11}$$

Noting that for odd w

$$\sigma\left(\left\lceil\frac{w}{2}\right\rceil\right) = \frac{w+1-2\frac{w+1}{2}}{w+1-\frac{w+1}{2}} = 0$$

allows us to rewrite equation 2.11 as

$$K = \binom{w}{0}\sigma(0) + \sum_{z=1}^{\lfloor\frac{w}{2}\rfloor}\binom{w}{z}\sigma(z) + \sum_{z=\lceil\frac{w}{2}\rceil+1}^{w}\binom{w}{z}\sigma(z) \tag{2.12}$$

(which trivially holds for even w). Introducing the variable $z' = w + 1 - z$ the last term of equation 2.12 rewrites to

$$\begin{aligned}
\sum_{z=\lceil \frac{w}{2} \rceil+1}^{w} \binom{w}{z}\sigma(z) &= \sum_{z'=\lfloor \frac{w}{2} \rfloor}^{1} \binom{w}{w+1-z'}\sigma(z) \\
&= \sum_{z'=\lfloor \frac{w}{2} \rfloor}^{1} \frac{w!}{(w-(w+1-z'))!(w+1-z')!}\sigma(z) \\
&= \sum_{z'=\lfloor \frac{w}{2} \rfloor}^{1} \frac{w!}{(z'-1)!(w+1-z')!}\sigma(z) \\
&= \sum_{z'=\lfloor \frac{w}{2} \rfloor}^{1} \frac{z'}{w+1-z'}\frac{w!}{z'!(w-z')!}\sigma(z) \\
&= \sum_{z'=\lfloor \frac{w}{2} \rfloor}^{1} \frac{z'}{w+1-z'}\binom{w}{z'}\sigma(z) \ .
\end{aligned}$$

Furthermore, expressing $\sigma(z)$ in terms of z' gives

$$\begin{aligned}
\sigma(z) &= \frac{w+1-2z}{w+1-z} \\
&= \frac{w+1-2(w+1-z')}{w+1-(w+1-z')} \\
&= -\frac{w+1-2z'}{z'} \ .
\end{aligned}$$

We can again rewrite the last term of equation 2.12 as

$$\begin{aligned}
\sum_{z=\lceil \frac{w}{2} \rceil+1}^{w} \binom{w}{z}\sigma(z) &= \sum_{z'=\lfloor \frac{w}{2} \rfloor}^{1} \frac{z'}{w+1-z'}\binom{w}{z'}(-\frac{w+1-2z'}{z'}) \\
&= -\sum_{z'=\lfloor \frac{w}{2} \rfloor}^{1} \binom{w}{z'}\frac{w+1-2z'}{w+1-z'} \\
&= -\sum_{z'=1}^{\lfloor \frac{w}{2} \rfloor} \binom{w}{z'}\sigma(z') \ .
\end{aligned}$$

We now rewrite equation 2.12 as

$$K = \binom{w}{0}\sigma(0) + \sum_{z=1}^{\lfloor \frac{w}{2} \rfloor} \binom{w}{z}\sigma(z) - \sum_{z'=1}^{\lfloor \frac{w}{2} \rfloor} \binom{w}{z'}\sigma(z') = 1 \ ,$$

which concludes the proof of Lemma 2.4.

□

Chapter 3

Minimal Dominating Set Protocol

In this chapter we present a protocol for approximating a Minimal Dominating Set using a Stone Age network. A dominating set is defined as follows:

Definition 3.1 (Dominating Set)**.** Given an undirected graph $G = (V, E)$, a dominating set is a subset $S \subseteq V$ of its nodes such that for all nodes $v \in V$, either $v \in S$ or a neighbor u of v is in S.

The protocol, which is represented in Fig. 3.1, is based on the *Distributed Greedy Minimal Dominating Set algorithm*, which produces a $(\log \Delta+2)$-approximation of a minimal dominating set. We first present the aforementioned Minimal Dominating Set algorithm, and then introduce the sub-protocols used in our implementation.

3.1 Greedy Minimal Dominating Set Algorithm

In a dominating set algorithm, we can differentiate the following kinds of nodes:

- the *black* nodes, which are part of the dominating set already
- the *gray* nodes, which are not in the dominating set (yet) but have at least one black node in their neighborhood (also referred to as *covered* nodes)
- the *white* nodes, which are not in the dominating set (yet) and have no black node in their neighborhood (also referred to as *uncovered* nodes).

We show how to represent the notions of white, gray and black nodes in the Stone Age Model (see Fig. 3.1):

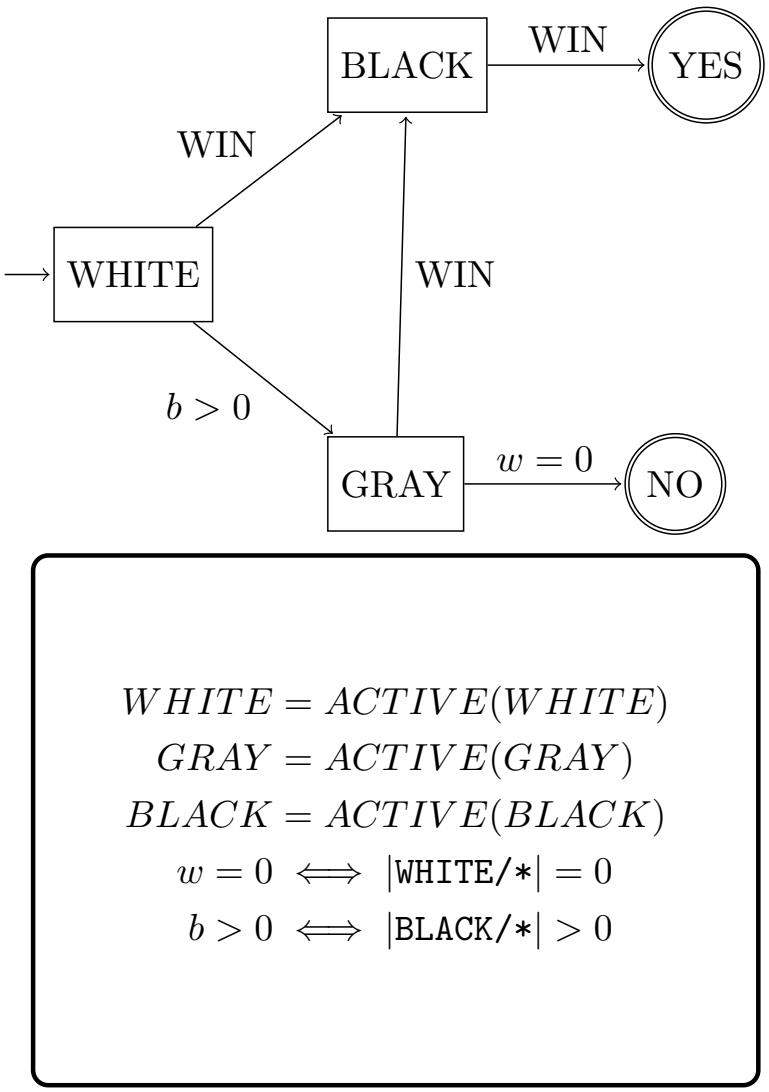


Figure 3.1: Stone Age Greedy Minimal Dominating Set Protocol (`GREEDY MDS`).

Definition 3.2 (Node colours)**.** Let `W`, `G` and `B` be defined as the sets of white, gray and black states, respectively:

$$\begin{aligned} \texttt{W} &= \texttt{WHITE} \\ \texttt{G} &= \texttt{GRAY} \\ \texttt{B} &= \texttt{BLACK} \,. \end{aligned}$$

Let x be a node and $s(x)$ be the state x is in. Then let the color of a node be defined by the color of the states it is in:

$$\begin{aligned} x \text{ is white} &\iff s(x) \in \texttt{W} \\ x \text{ is gray} &\iff s(x) \in \texttt{G} \\ x \text{ is black} &\iff s(x) \in \texttt{B} \,. \end{aligned}$$

Moreover, we call the number of white nodes in a given node's *inclusive* neighborhood the *span* of that node:

Definition 3.3 (Spans)**.** We denote the set of white neighbors of a node u (u possibly included) at time step τ by $W(u, \tau)$. This allows us to define the *span* of a node u at time step τ, $w(u, \tau)$, as the number of u's white neighbors, i.e.

$$w(u, \tau) = |W(u, \tau)|$$

where $v \in W(u, \tau) \iff v \in \hat{N}(u)$ and $s(v, \tau) \in \texttt{W}$. When τ is clear from the context, we denote the span by

$$w(u) = w(u, \tau) \,.$$

The Greedy Minimal Dominating Set Algorithm for a given node v then goes as follows: as long as v has white neighbors,

- compute $w(v)$, the span of v
- send $w(v)$ to all neighbors at distance at most 2
- if $w(v)$ is larger than all other spans within distance 2 then join dominating set.

A more extensive analysis of the Greedy Minimal Dominating Set Algorithm, along with the proof of the following result can be found in [3]:

Theorem 3.4. *The Greedy Minimal Dominating Set Algorithm produces a* $(\log \Delta + 2)$*-approximation of a minimal dominating set, where* Δ *is the networks' highest degree.*

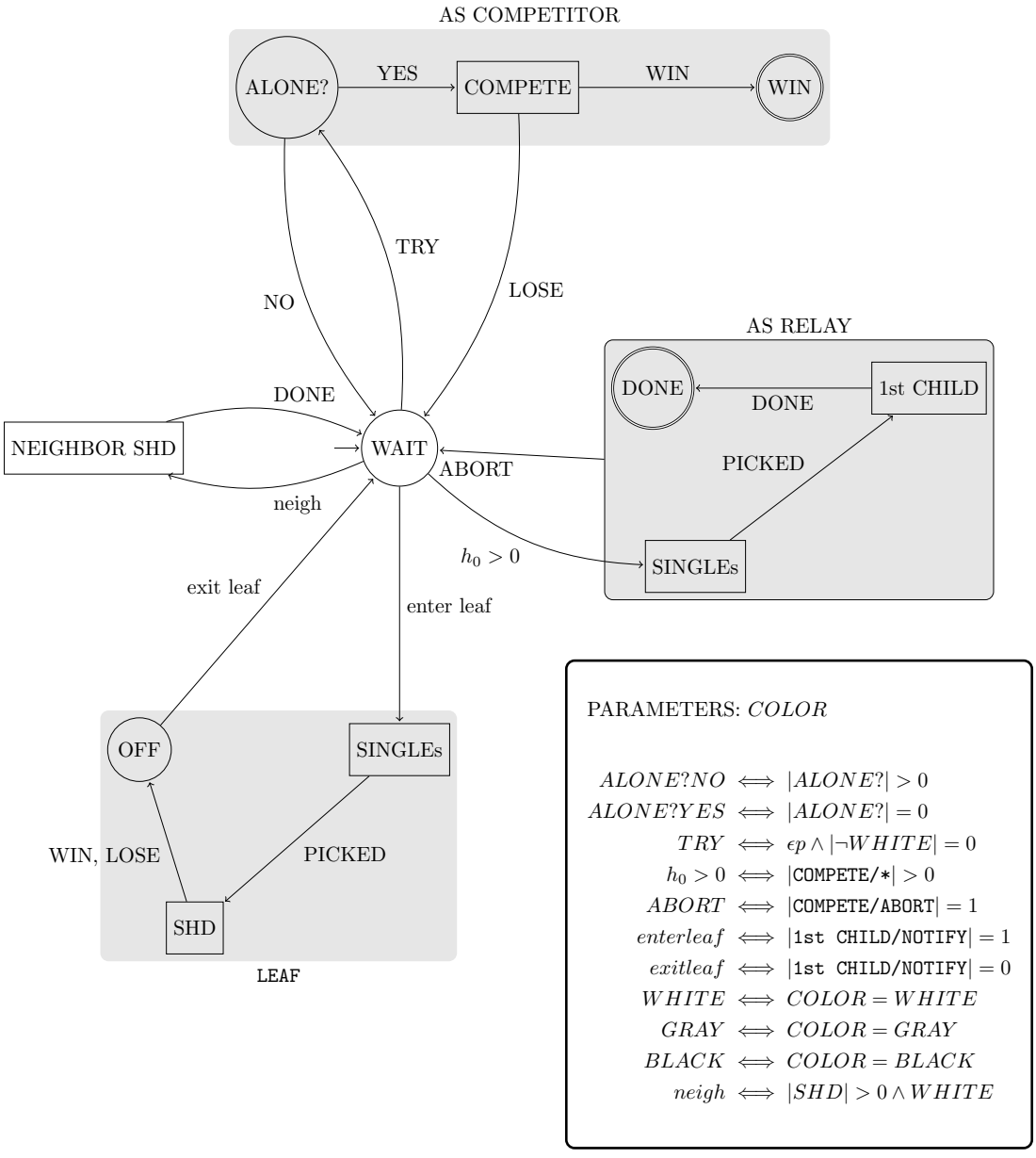


Figure 3.2: Active nodes of the `GREEDY MDS` protocol (`ACTIVE`).

3.2 The GREEDY MDS protocol

It can be seen in Fig. 3.1 that the GREEDY MDS relies exclusively on the ACTIVE protocol, which is presented in Fig. 3.2.

3.2.1 Overview

To explain how the GREEDY MDS protocol works, we will follow what happens to a (white) node becoming black. Every node starts out as white, in state WHITE/WAIT. At some time step it will try to engage in a series of tests to ensure it has the greatest span in its 2-hop neighborhood. To do so, it will single out one of its direct neighbors, and perform a degree test with that neighbor. Then, that neighbor will be used as a relay, and the original node will perform a relayed degree test against every node in the neighbor's 1-hop neighborhood. The process (singling out a direct neighbor, and testing against its inclusive neighborhood) is repeated for every neighbor in the originial node's 1-hop neighborhood. We now give a more detailed explanation about how this is performed by the GREEDY MDS protocol.

3.2.2 Competition

A node that works as competitor will try to become black. It will do so by first verifying that none of its neighbors is also trying (ALONE?). Then it will enter the protocol COMPETE (presented in Fig. 3.3), where it will successively compare its span to all white or gray nodes in its 2-hop neighborhood. For every neighbor v in its direct neighborhood it will perform a *branch test*:

- verify that $w(u) \geq w(v)$
- verify that $w(u) \geq w(x), \forall x \in N(v)$, using v as a relay

The branch test is implemented in the protocol BRANCH TEST found in Fig. 3.4. Such a node v at distance 1 from u will be in in the state group *As Relay* represented in Protocol ACTIVE. The sub-protocol 1st CHILD, shown in Fig. 3.5, is responsible for carrying the distance-1 test (u and v) and responsible for relaying the information between u and the nodes in u's neighborhood during the subsequent distance-2 tests.

3.2.3 Testing a branch

We now show that a node u winning a branch test (exiting with BRANCH TEST/WIN) using v indeed means that u has a span greater that any node in v's inclusive

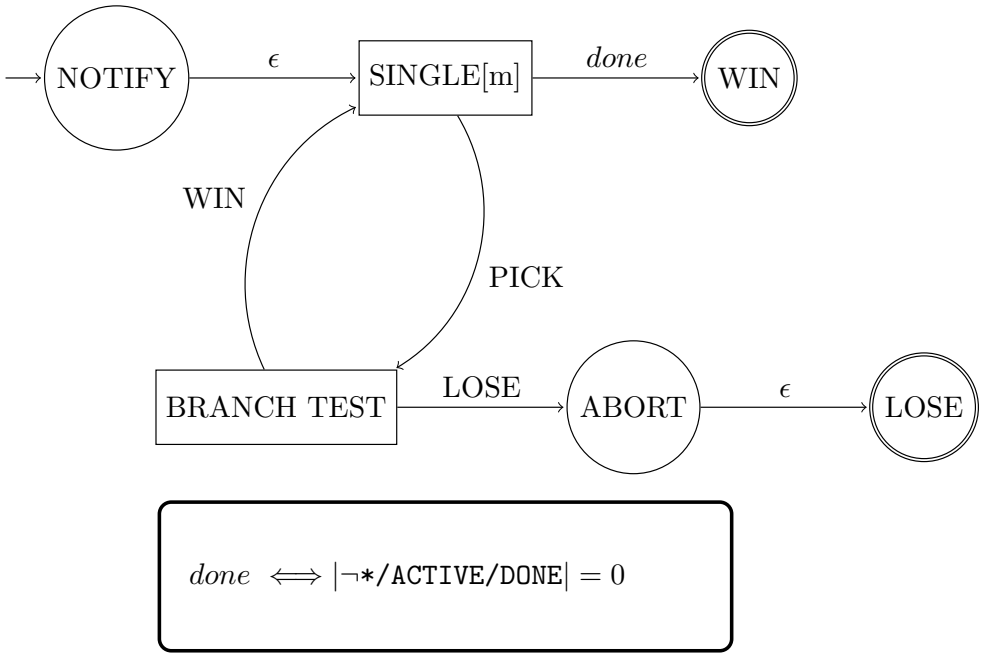


Figure 3.3: (COMPETE).

neighborhood. Let two such nodes u and v be at distance 1 from each other, let τ_s be a time step such that

$$u \xrightarrow{\tau_s} \texttt{BRANCH TEST}, v \xrightarrow{\tau_s} \texttt{FIRST CHILD}$$

where $\texttt{U} = \tilde{U}\texttt{/BRANCH TEST}, \texttt{V} = \tilde{V}\texttt{/1st CHILD}$, $\tilde{U}$ and $\tilde{V}$ any states, and let τ_f be the *first* time step after τ_s at which u and v leave the protocol.

Lemma 3.5 (Branch protocol)**.** *Either u has the greatest span within its 2-hop neighborhood and exits with* `WIN`*, or it exits with* `LOSE`*, i.e.*

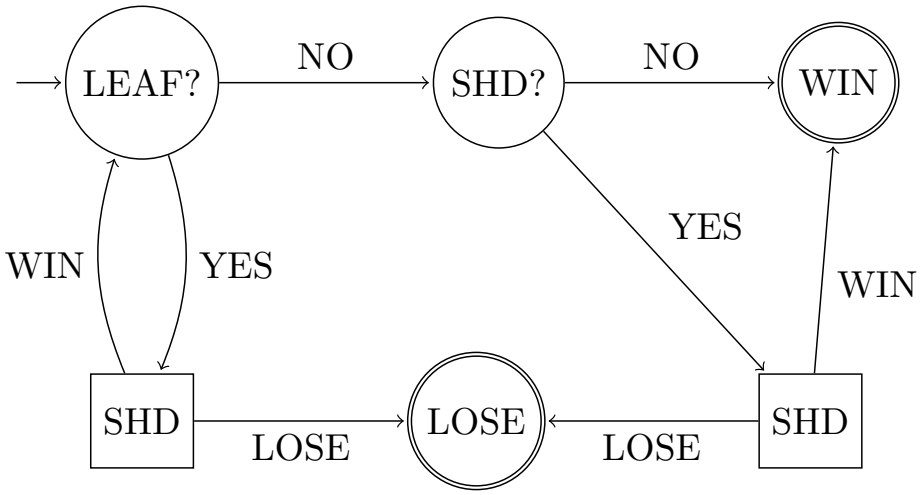
$$u \xrightarrow{\tau_f} \texttt{U/WIN} \iff w(u) \geq w(x), \forall x \in \hat{N}(v)$$

and

$$u \xrightarrow{\tau_f} \texttt{U/LOSE} \iff w(u) < w(x), \text{ for some } x \in \hat{N}(v)$$

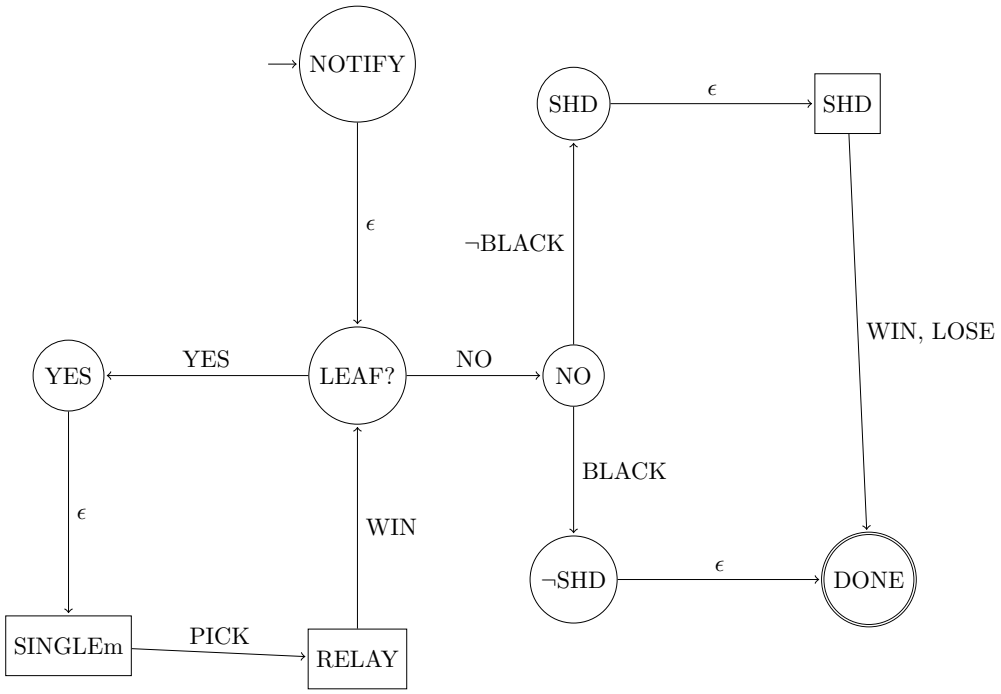
hold, and moreover $\tau_s < \tau_f < \infty$.

Proof. First note that, by definition, u has a span greater than or equal to that of any black node. Let $N_v = \{x \in N(v) \mid x \notin S_B \wedge s(x) \neq \texttt{ACTIVE/LEAF/OFF}\}$ be the set of all the non-black neighbors of v that have yet to be SHD-tested, at a given time step, with $n_v = |N_v|$. Exactly one time step after v enters `V/NOTIFY`, every node in N_v will enter `ACTIVE/LEAF/SINGLEs`, v will enter `V/LEAF?`, and u will still be in state `U/LEAF?`. Note that if this configuration happens at any time



$$
\begin{aligned}
LEAF?YES &\iff |\texttt{*/1st CHILD/YES}| = 1 \\
LEAF?NO &\iff |\texttt{*/1st CHILD/NO}| = 1 \\
SHD?YES &\iff |\texttt{*/1st CHILD/SHD}| = 1 \\
SHD?NO &\iff |\texttt{*/1st CHILD/}\neg\texttt{SHD}| = 1
\end{aligned}
$$

Figure 3.4: (`BRANCH TEST`).



$PARAMETERS$:$BLACK$

$$LEAF?YES \iff |h_2SINGLEs| > 0$$
$$LEAF?NO \iff |h_2SINGLEs| = 0$$
$$WIN \iff \texttt{|*/BRANCH TEST/LEAF?|} = 1$$

Figure 3.5: (`1st CHILD`).

step τ, then at time step $\tau + 2$ v will enter state `V/SINGLEm`, u will be in state `U/SHD` (since time step $\tau + 1$) and every $x \in N_v$ will be in a non-accepting state of `ACTIVE/LEAF/SINGLEs` again. Then one neighbor from N_v will be singled out and SHD-tested against u in finite time, from Lemmas 1.4, 2.1 and 1.5.

Let u enter `SHD` at time step τ_s. When u is SHD-tested, two outputs are possible: WIN and $LOSE$. Assume first that u exits `U/SHD` with WIN, at time step $\tau_r > \tau_s$. Then u will re-enter state `U/LEAF?`, v will re-enter state `V/LEAF?`, and all nodes in $N_v(\tau_r)$ will still be in state `ACTIVE/LEAF/SINGLEs`, with $n_v(\tau_r) = n_v(\tau_s) - 1$. This is the same configuration as that of τ_s, with one node less in N_v. Now assume that at a given time step u exits `SHD` with $LOSE$. Then u will also exit `U` with $LOSE$. Since the process is repeated until $n_v = 0$, u can exit `U` with WIN only if it has a span greater than that of any node in $N(v) \setminus v$.

□

Using Lemma 3.5, it is now easy to show that if a node u exits `COMPETE` with `WIN`, it has the greatest span within its 2-hop neighborhood. The `COMPETE` protocol carries a `BRANCH TEST` with every node in u's neighborhood. Note that u might be tested against the same node several times; unfortunately, due to the nature of the Stone Age model it is not clear if this can be avoided.

Lemma 3.6 (Competing protocol)**.** *A node that exits $COMPETE$ with WIN has a span greater or equal to any other node in its 2-hop neighborhood.*

Proof. Let u enter `COMPETE`. One step later, all its neighbors will be in state `1st CHILD/NOTIFY`. By Lemma 1.4, as long as there are "first children" (nodes executing the protocol `1st CHILD`), a "first child" v will be picked. By Lemma 3.5, only if u has a span greater than or equal to that of v or any of v's neighbors can it not exit with `LOSE`. This will repeat until either u has no first child neighbors left, meaning it has a span greater than that of any node in $\hat{N}_2(u)$, or u exits with `LOSE`. □

3.2.4 Forming a Dominating Set

It is now clear that only a node which has the greatest span in its 2-hop neighborhood *can* enter the dominating set, i.e. become black. We now show that not only can it, but eventually will. The reason is that every node will try to compete until it is either black, or until it and all its neighbors are covered.

Lemma 3.7 (Nodes becoming black)**.** *Let u be a gray or white node such that $v \in S_W \cup S_G, \forall v \in \hat{N}_2(u)$, for any given time interval $[\tau_0, \tau_f]$. Then, $\forall \tau \in [\tau_0, \tau_f]$, the following holds:*

$$p_B = P(u \to \texttt{ACTIVE(BLACK)}) > 0 \iff w(u) \geq w(v), \forall v \in \hat{N}_2(u) \ .$$

Proof. Let u be a node G such that $s(x) = \texttt{ACTIVE/WAIT}, \forall x \in N(u) \cup \{u\}$. Then p_C, the probability of u entering `ACTIVE/COMPETE` at a given time step τ_C, is the probability of u being the only node in $\hat{N}(u)$ to enter `ACTIVE/ALONE?` at time $\tau_C - 1$ with

$$p_C = p(1-p)^{\delta_u} > 0 \ .$$

Let p_W be the probability of u exiting `ACTIVE/COMPETE` with WIN. Then, from Lemma 3.6, it holds that

$$p_B = p_C p_W > 0 \iff p_W > 0 \iff w(u) \geq w(v), \forall v \in \hat{N}_2(u) \ .$$

□

Lemma 3.8. *If the protocol* `GREEDY` *terminates, the nodes in state* `GREEDY/YES` *form a dominating set, and this dominating set is a* $(\log \Delta + 2)$*-approximation of a minimal dominating set.*

Proof. It follows from Lemma 3.6 that any node exiting `ACTIVE` with `WIN` has the greatest span in its 2-hop neighborhood. Also, the only way for a node to reach one of the two accepting states `GREEDY/YES` and `GREEDY/NO` is to either become black or gray first. Lemma 3.8 then follows from Theorem 3.4. □

Lemma 3.9. *The protocol* `GREEDY` *terminates in finite time w.p. 1.*

Proof. Let the protocol start at τ_0. Then it holds for any node that

$$\tau_0 : v - \mathtt{S}_W$$

since $\mathtt{S}_W$`/WAIT` is the initial state. If a white node $u \in \mathtt{S}_W$ has the greatest span in its 2-hop neighborhood, and thus

$$w(u) \geq w(x), \forall x \in \hat{N}_2(u) \ ,$$

then by Lemma 3.7 it will eventually become black. If on the other hand

$$w(u) < w(x), \text{ for some } x \in \hat{N}_2(u) \ ,$$

then since Δ (the graph's maximal degree) is finite, there exists at least one node u' in the whole graph for which the previous argument applies. That node u' will eventually become black, and thus allow another white node $u'' \in N_2(u')$ with span equal to or less than u''s previous span to become black as well. Thus u itself will either become black itself, or a node at distance one will become black,

and u will become gray. Thus a white node will eventually become either gray or black.

If there is a gray node v_G with

$$w(v_G) \geq w(x), \forall x \in \hat{N}_2(v_G) \ ,$$

then again by Lemma 3.7 it will eventually become black. If it doesn't, then, similarly to the previous paragraph, eventually nodes in $N_1(v_G)$ will become black or gray, and v_G will become `NO`. Thus a gray node will eventually become either black or `NO`.

Once a node u_B has become black (and since it is then not white) it will never have to be accounted for as `NEIGHBOR`. Thus it will not interact with any competing node $v_C \notin N_2(u_B)$. When there is no node $v \in N_2(u_B)$ with $w(v) > w(u_B)$, which from above will happen eventually, then u_G will enter `YES`.

□

Chapter 4

Computational Power of a Path Network

In this chapter we research what a "difficult" problem is in the Stone Age model. To start out we take the example of the Local Model, where a problem can be called "difficult" if it needs linear time to be solved. The reason is that linear is an upper bound for the execution time in the Local Model (every node can solve any local problem in a single time step, what takes time is only the information sharing between the nodes). A solution to a Stone Age problem that can be solved can actually take much longer than linear time to be found. To show that, we will first prove an equivalence between a specific the Stone Age model topology (path network) and the so called Linear Bounded Automaton model (a Turing Machine with limited tape size).

Some NP problems can actually be solved using a Linear Bounded Automaton (or LBA). The Satisfiability problem, for instance, is proven to be an NP problem; yet it can be solved using a limited tape size, making it a perfect candidate. By showing that a Stone Age network can be equivalent to an LBA we show that a Stone Age network can solve an NP problem. This is also interesting in regard to the Local Model. As mentioned earlier, every problem can be solved in linear time in the Local Model: now, unless $P = NP$, this means that some problems *cannot* be solved in linear time in the Stone Age model, making them even more "difficult" than in the Local Model.

The network topology that we use in order to show that the Stone Age model can be as powerful as an LBA is that of a *path network*, as depicted in Fig. 4.1.

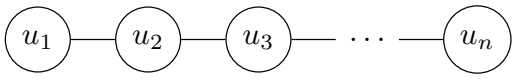


Figure 4.1: A path network. It is a tree with two terminal nodes of degree 1, while all others have degree 2.

We can refine the definition of a Stone Age network for the special case of a path network as follows:

Definition 4.1. A Stone Age Path Network, or SAPN, is defined by

- a node state space S_S
- a transition function $\delta_S : S_S \times S_S \times S_S \to S_S$, which produces a new state based on respectively the node's current state, and node's left neighbor's state and the node's right neighbor's state, and for which it holds that

  $$\delta_S(s, s_L, s_R) = \delta_S(s, s_R, s_L)$$

  as a node cannot differentiate between right and left neighbors
- an initial state $\Gamma_S = (s_1, s_2, \ldots, s_n)$ with $\Gamma_S \in S_S^n$ where s_i is the initial state of node u_i.

Note that we allow for each node to have a different initial state.

4.1 Linear Bounded Automaton Equivalence

The Linear Bounded Automaton (or LBA) is a special kind of Turing Machine. Its alphabet contains two special symbols, used as left and right marker of the tape. The LBA may not write over those special symbols, nor move to the left of the left marker or right of the right marker. It can be simply represented as a Turing Machine with a limited tape, as seen on Fig. 4.2.

We will use the following definition for the Linear Bounded Automaton:

Definition 4.2. A Linear Bounded Automaton, or LBA, is a Turing Machine with

- a tape alphabet $\Sigma'_L = \Sigma_L \cup \{D_L, D_R\}$, where D_L and D_R are the left and right delimiters, respectively,
- a state space S_L
- a transition function $\delta : S_L \times \Sigma'_L \to S_L \times \Sigma'_L \times \{L, R\}$ as

  $$\delta(s, \sigma) = (s', \sigma', X)$$

  where s is the current state, σ is the symbol on the current tape square, s' is the new state, σ' the tape symbol to write and X is either L or R, representing respectively moving to the left or right
- two extra conditions on the transition function δ_L

  $$\delta_L(s, D_L) = (s', D_L, R)$$

  and

  $$\delta_L(s, D_R) = (s', D_R, L)$$

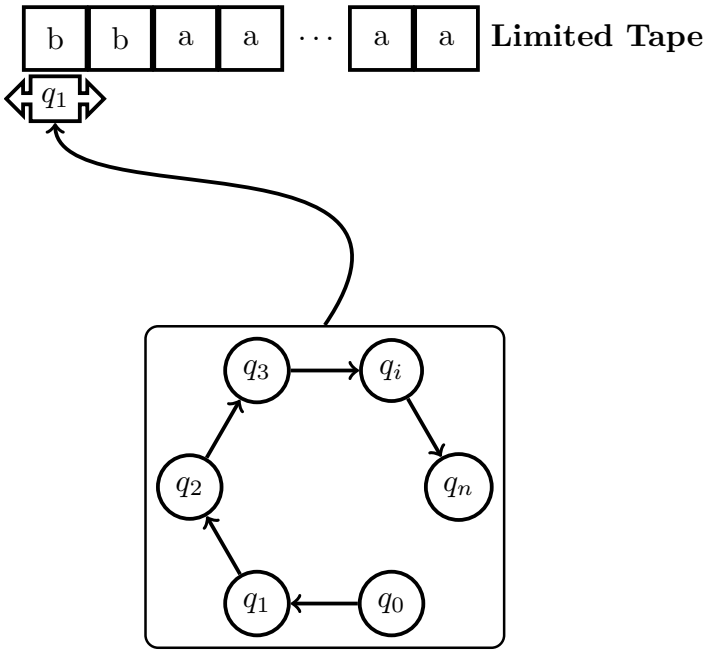


Figure 4.2: The Linear Bounded Automaton is a Turing Machine with a limited tape size.

The two extra conditions on the transition function ensure that the tape will not overwrite a delimiter or move past it.

As the LBA and the SAPN have a very similar structure, it is not surprising that, as we will show shortly, they have equivalent computational power.

4.1.1 Simulating LBA Execution with Stone Age Path Network

We will represent each square of the LBA's tape by a node on the Stone Age Path Network (SAPN). We label each node node of the SAPN from u_1 to u_n, where the node u_i should represent the i^{th} square of the LBA's tape. Note that there are two main differences between an LBA and a SAPN:

- the LBA starts with some input encoded on its tape
- the LBA knows which square is left and which is right from the current square.

To work with those differences we introduce the following assumptions on the way the SAPN is given its input:

- the node u_i holds the input information of the i^{th} square of the LBA's tape

- the node u_1, representing the LBA's leftmost square, is marked.

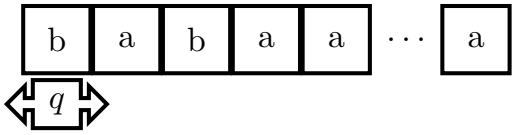


Figure 4.3: Input on an LBA tape.

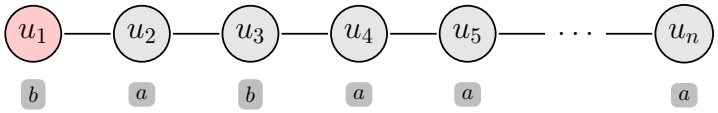


Figure 4.4: Input in the SAPN.

There is one important point that needs to be clarified. The tape of the LBA decides to go either right or left; unfortunately, the SAPN cannot tell which one of its neighbors is right or left. To alleviate the issue, we will *color* the nodes in the SAPN using three distinct *colors*, for instance 0, 1 and 2. We will start from the (marked) leftmost node, coloring it as 0. Then, every following node to the right will be colored according to the following sequence: $0, 1, 2, 0, 1, 2, \ldots$. That way it is always possible to infer from which side a message is coming, based on the senders color; a message from a 1 node will always come from the left of a 2 node, and always come from the right of a 0 node. In other words, if the current node's coloring is c, and it receives a message from a node with coloring c' then

$$\text{if } c = c' +_3 1 \text{ then the message arrived from the left, otherwise from the right}$$

where we used the symbol $+_3$ to represent an addition modulo 3.

We will be considering only one node per time step, representing the LBA's head; using the coloring we can then always differentiate which node is left and which node is right from the current "head" node.

Given an LBA $Q_L = (\Sigma_L, S_L, \delta_L)$, we present the SAPN $Q_S = (S_S, \Sigma_S, \delta_S)$ simulating Q_S as

$$S_S = \Sigma_S^{(A)} \times \Sigma_S^{(C)} \times \Sigma_L \times S_L$$

with

$$\Sigma_S^{(A)} = \{w, L, R\}$$

and

$$\Sigma_S^{(C)} = \{0, 1, 2\} \cup \{u\} \, .$$

First, $\Sigma_S^{(A)}$ allows a distinction between a node that currently has the role of the head and a node that is merely representing a square on the tape. A node representing the tape head goes either *left* (L) or *right* (R). A node that does not represent the head is waiting (w). Then, $\Sigma_S^{(C)}$ represents the coloring $(0, 1, 2)$ or lack thereof (u). Finally, Σ_L and S_L represent the LBA's information.

Transition function δ_S Let's first consider nodes in state $s = (a = w, c, \sigma_L, s_L)$ with neighbors in states $s' = (a', c', \sigma'_L, s'_L)$ and $s'' = (a'', c'', \sigma''_L, s''_L)$, i.e. nodes that do not currently represent the tape head. We define the transitions as

1. It the node is uncolored, and a neighbor of color c' is in state R, then change to color $c = c' +_3 1$, process the current "tape square" and move, according to the LBA's instructions. Formally, if $c' \neq u$, $c = u$ and $a' = r$ then
$$\delta_S(s) = (d, c' +_3 1, \sigma'_L, s'_L)$$
with
$$(s'_L, \sigma'_L, d) = \delta_L(s_L, \sigma_L)$$

2. If the node is either the right neighbor of a node passing the head right, or the left neighbor of a node passing the head left, then process the current "tape square" and move, according to the LBA's instructions. Formally, if either $c = c' +_3 1$ and $a' = r$ or $c = c' -_3 1$ and $a' = l$ then
$$\delta_S(s) = (d, c, \sigma'_L, s'_L)$$
with
$$(s'_L, \sigma'_L, d) = \delta_L(s_L, \sigma_L)$$

3. otherwise $\delta_S(s) = s$.

At any time, if a node just passed the head further, it goes back to a waiting state:
$$\delta_S((R, c, \sigma_L, s_L)) = (w, c, \sigma_L, s_L)$$
and
$$\delta_S((L, c, \sigma_L, s_L)) = (w, c, \sigma_L, s_L) \ .$$

We also assume that every node u_i initially holds the LBA's i^{th} square value, and that the leftmost node enters the state R on the first time step, in order to start the process.

Analysis

Note that at any time step only one node represents the tape head. It follows that the total number of steps $f_{SA}(n)$ performed by the SAPN to reach a configuration is exactly the number of steps $f_{LBA}(n)$ of the LBA, or simply

$$f_{SA}(n) = f_{LBA}(n) \ .$$

4.1.2 Simulating Stone Age Path Network Execution with LBA

We are now interested in building an LBA which simulates the execution of a given SAPN. Since each node of the SAPN can potentially change state at each time step depending on its neighbors state and own current state, the LBA has traverse its own tape for every time step in the SAPN.

Given an SAPN $Q_S = (S_S, \delta_S, \Gamma_S)$, we present the LBA $Q_L = (\Sigma_L, S_L, \delta_L)$ simulating Q_S with

$$S_L = \Sigma'_S \times \Sigma'_S \times \Sigma_L^{(A)}$$

with

$$\Sigma'_S = \Sigma_S \cup \{X\}$$

and

$$\Sigma_L^{(A)} = \{n, rf, mlf, lf, p, c\}$$

(representing respectively a *new* node, fetching the right neighbor's state, moving back to fetch the left neighbor's state, fetching the left neighbor's state, processing the node, and eventually copying the new node's state on the old state) with tape alphabet

$$\Sigma_L = \Sigma_S \times \Sigma_S$$

(accounting for both the new and previous state) and with transition function δ_L. For a state $s \in S_L$ with $s = (a, rn, ln)$ and a current symbol $\sigma \in \Sigma'_L$ with $\sigma = (\sigma^{(p)}, \sigma^{(n)})$ (if $\sigma \in \Sigma_L$) we propose the following transition function δ_L:

- If we work on a new node, and the current square is the right delimiter, we change to copy mode. Formally, if $a = n$ and $\sigma = D_R$ then $\delta_L(s, \sigma) = (s', D_R, L)$ with $s' = (a' = c, rn, ln)$
- If we work on a new node, and the current square is *not* the right delimiter, we move to fetch the right neighbor's state. Formally, if $a = n$ and $\sigma \neq D_R$ then $\delta_L(s, \sigma) = (s', \sigma, R)$ with $s' = (a' = rf, rn, ln)$
- If we are fetching the right neighbor's state, read it, and move to fetch the left neighbor's state. Formally, if $a = rf$ then $\delta_L(s, \sigma) = (s', \sigma, L)$ with $s' = (a' = mlf, rn', ln)$ and $rn' = \sigma^{(p)}$

- If we are merely moving over a node to reach its left neighbor, do not do anything special. Formally, if $a = mlf$ then $\delta_L(s, \sigma) = (s', \sigma, L)$ with $s' = (a' = lf, rn, ln)$
- If we are fetching the left neighbor's state, and we actually read the left delimiter D_L, we infer that the node has only one neighbor. Formally, if $a = lf$ and $\sigma = D_L$ then $\delta_L(s, \sigma) = (s', \sigma, R)$ with $s' = (a' = p, rn, ln')$ and $ln' = X$
- If we are fetching the left neighbor's state, and the current tape square is not the left delimiter, read it, and move back to the node. Formally, if $a = lf$ and $\sigma \neq D_L$ then $\delta_L(s, \sigma) = (s', \sigma, R)$ with $s' = (a' = p, rn, ln')$ and $ln' = \sigma^{(p)}$
- If we have gather all the information of the left and right neighbor, we process the node. Formally, if $a = p$ then $\delta_L(s, \sigma) = (s', \sigma', R)$ with $s' = (a' = n, rn, ln)$ and $\sigma' = (\sigma^{(p)}, \delta_S(\sigma^{(p)}, ln, rn))$
- If we are copying the new results over the previous results and the current tape square is the left delimiter, we start processing the tape from the beginning again. Formally, if $a = c$ and $\sigma = D_L$ then $\delta_L(s, \sigma) = (s', \sigma, R)$ with $s' = (a' = n, rn, ln)$
- If we are copying the new results over the previous results and the current tape square is *not* the left delimiter, we copy the current result (new node state) over the previous one, and continue to the left. Formally, if $a = c$ and $\sigma \neq D_L$ then $\delta_L(s, \sigma) = (s, \sigma', L)$ with $\sigma' = (\sigma^{(n)}, \sigma^{(n)})$.

Since the LBA will be in each action state $a \in \Sigma_L^{(A)}$ exactly once per square, the total number of steps $f_{LBA}(n)$ performed by the LBA to reach a configuration is

$$f_{LBA}(n) = 6nf_{SA}(n) \ .$$

4.2 Super-polynomial Execution Time and Local Model

Based on the results of Section 4.1 we will now discuss properties of the Stone Age model in terms of time complexity, including relations to the Local Model. Specifically, we will show the two following results:

Theorem 4.3. *Unless $P = NP$ there are specific network topologies such that the Stone Age model can reach execution times exponential in the number of nodes in the network*

and

Theorem 4.4. *Unless $P = NP$ there is a network topology T and a problem P such that the Local Model can solve any instance of P on T in linear time, whereas the Stone Age model can only solve a general instance of P on T in super-polynomial time.*

We mean *exponential*, *linear* and *super-polynomial* time in terms of the number of nodes in the given network.

4.2.1 Satisfiability Problem

The Satisfiability Problem (SAT) is to test whether a given Boolean formula expressed with the variables $x = (x_1, x_2, ..., x_n)$ is satisfiable (evaluates to 1) for some x. It was proven [4] to be an NP problem, yet solvable using an amount of space linear in n, which makes it an NP problem that can be solved with an LBA.

As we showed in Section 4.1, for every problem solvable by an LBA there exists an SAPN that can solve it too. Let us call such an SAPN M_S. From Section 4.1.1, we know that we can then in turn create an LBA M_L which can solve the problem in time $f_{M_L} = 6nf_{M_S}$. Because SAT is an NP problem, then unless $P = NP$ M_L will need a super-polynomial amount of time to solve an instance of it. Since f_{M_L} must be super-polynomial, so must f_{M_S}, which proves Theorem 4.3.

Observation 4.5. *A path network of nodes operating under Local Model assumptions can solve any problem in $2n$ steps.*

Observation 4.5 comes from definition of the Local Model, which allows any node to solve even NP problems within one time step. On a path network, the information of every node can simply be gathered on one node (n steps at most from leftmost node to rightmost node), used by that node to solve the problem (including assigning where to dispatch the results), and sent back to the relevant nodes (n steps at most). Observation 4.5, in conjunction with Theorem 4.3, makes Theorem 4.4 trivially hold (note that this would not hold for a congested model where the size of messages is limited).

Bibliography

[1] Wattenhofer, R.: Stone age distributed computing. In: ETH Zurich

[2] Norris, J.: Markov chains. In: Cambridge University Press. (September 2004)

[3] Kuhn, F.: Lecture notes on network algorithms. In: University of Lugano

[4] Sipser, M.: Introduction to the theory of computation, 3rd edition. In: Cengage Learning. (July 2012)