A summary and modification of Higham & Przytycka's algorithm for maximum finding on rings, for the undirected case

Jakub Migdał

1 A starting point

We begin by proposing a simple, round-based leader election scheme on directed rings which we will then extend with two heuristics that lead to an improvement in the number of rounds and sent messages.

```
function DropMessage
    return received.round = last.round and
        ((odd(received.round) and received.id > last.id) or
        (even(received.round) and received.id < last.id))
end function

function Promotion
    return received.round = last.round
end function

function IsLeader
    return received.round = last.round and received.id = last.id
end function
```

Algorithm 1 LE.BASIC

```
repeat
    if DropMessage is false then
        if Promotion is true then
            received.round ← received.round + 1
        end if
        last ← received
        send(received)
    end if
until IsLeader
```

From a high-level perspective the algorithm works as follows: each node sends an envelope containing its ID, and saves it as the last envelope it has sent. The nodes (referred to as processors in the paper, a nomenclature we too shall use) then act as actors comparing received envelopes to the last seen envelope, and eliminating them if the defined conditions hold. This effectively produces a "chain" reaction where each round, the surviving envelopes are gathered into maximal chains of descending (ascending, depending on the round's parity) ID's, from which only the minimum (maximum) of its chain remains.

This perspective immediately establishes the algorithm's correctness based on the following properties:

1. Safety: It will never delete all envelopes

2. Progress: Only one envelope will remain, since a sequence of two alternating rounds must drop a non-zero number of envelopes

3. Correct termination: The algorithm terminates when a processor detects the received envelope is the one it has last sent, meaning it has made a round trip uninterrupted.

It can be shown the algorithm uses no more than $n \log_\phi n \leq 1.44n$ messages, based on the following property:

Lemma 1. *An envelope in round i travels a distance of at least F_{i+1} before advancing to the next round (where F_i is the i'th Fibonacci number)*

The proof follows by induction and considering a few cases which we will omit for the sake of the modified algorithm. With this in mind, and the fact that during each round the envelopes make exactly n hops in total, we can bound the number of envelopes left after round i to be at most $\frac{n}{F_{i+1}}$, giving us the expected number of rounds, and as a consequence, of sent messages.

2 Early promotions

There are cases where an envelope may be promoted before it reaches the next processor that may decide to drop it. Thus a round can be finished with a total number of hops less than n. The paper considers two such possibilities for promotion:

The first promotion may happen due to a 'witness'. Let $< a, i >$ mean an envelope with ID a and round i.

Lemma 2. *An envelope $< b, i >$, where i is even, upon encountering a processor whose last sent envelope was $< a, i-1 >$ with $a < b$, may be promoted early, because the next processor with round i will necessarily have an ID no greater than a.*

Proof. Call w the next processor with round i that $< b, i >$ will encounter, and x the witness for $< b, i >$, whose last sent envelope was $< a, i-1 >$. Moreover, let z be the processor that promoted $< a, i-1 >$ to round i or dropped it. In the first case, $z = w$ and since it promoted $< a, i-1 >$ it shall also promote $< b, i >$. For the second case, $< a, i-1 >$ being dropped necessitates it to have belonged to some chain, the minimum of which, let's call it $< a', i-1 >$, had become promoted to round i. Thus, $< b, i >$ wil necessarily pass through all processors up to this processor, whose last envelope sent will be $< a', i >$, and it will have to promote $< b, i >$. An analogous reasoning follows if i is odd. □

The second type of early promotion is called 'promotion by distance'. The reasoning behind its inclusion in the algorithm is much more heuristic, as it acts more like a 'balancer' of the distance between envelopes in each round. The argument states that, since we have a lower bound of F_{i+1} on the distance travelled by the envelope, the existence of lengthier gaps suggests the processors have eliminated more envelopes than in the worst case. These longer gaps persist throughout the algorithm's run, and need not be alleviated via promotion by witness. Thus, we shall promote envelopes that have travelled a distance greater than F_{i+2} arbitrarily.

```
function PROMOTION'
    return (even(received.round) and last.round = received.round - 1 and received.id > last.id) or
           (odd(received.round) and
              (received.cnt = 0 or
              (last.round = received.round and received.id < last.id)))
end function
```

Algorithm 2 ELECTION

```
repeat
  if not DROPMESSAGE then
    if PROMOTION' then
      received.round ← received.round + 1
      received.cnt ← F_{received.round+2}
    end if
    received.cnt -= 1
    last = received
    SEND(received)
  end if
until ISLEADER
```

It is trivial to prove the correctness of this algorithm as it is not very different from the first one. One non-trivial issue might come up in regards to termination, in the case all envelopes become stuck, never meeting their successor's processor because they're being promoted by distance. Fortunately, at some point the envelope must be promoted with a count number high enough that it will reach the processor without being promoted by distance first. This ensures that for any round with more than one envelope, there exists a round no smaller than it, where at least one envelope must be dropped.

3 Complexity analysis

We will denote $host_i(a)$ to be the processor that promoted the envelope from round $i-1$ to i, and, if applicable, $destroyer_i(a)$ the processor that dropped it in round i.
We shall denote the distance between two processors p and q with $\delta(p,q)$.
We will ocassionally refer to envelopes by their IDs, since they do not change over the algorithm's run.
Envelope b is the *immediate successor in round i of envelope a when the first round i envelope encountered after envelope b in round i, travelling in the direction of the ring, is envelope a. Contrary to the paper, we will use lemmas only after proving them, instead of in order of utility.*

Lemma 3. *A witness for round $k+1$ that promotes an envelope to round $k+2$ was a host for round k*

Proof. An envelope with label c and round $k+1$ is promoted to round $k+2$ at its first encounter with a processor that has round number k and label, say d, less than c. The processor that promoted envelope $d$to round k, $host_k(d)$, has label d and round number k and all other processors with label d and round number k must follow $host_k(d)$. Envelope c could not have reached $host_k(d)$ in round k since otherwise it would have been destroydd by $host_k(d)$, so it encounters $host_k(d)$ in round $k+1$. □

Lemma 4.

$$\delta(host_k(b), host_{k+2}(b) \geq F_{k+2}$$

Proof. Consider b's travel in odd round k. If b was promoted to round $k+1$ after travelling F_{k+2} links then the observation is immediate. Otherwise, b was promoted by some processor, say $host_k(c)$ and $b < c$, and, by the induction hypothesis, $d(host_k(b), host_k(c)) > F_{k+1}$. Since b reaches round $k+2$, in round $k+1$, b travels from $host_{k+1}(b)(= host_k(c))$ to some witness w for round $k+1$ with label $d < b$ that promotes b to round $k+2$. By the inequalities, $c \neq d$, and hence w must be some processor not in the interval $[host_k(c), host_{k+1}(c))$. Hence $\delta(host_{k+1}(b), host_{k+2}(b)) \geq \delta(host_k(c), host_{k+1}(c)) \geq F_k$, by the induction hypothesis. The combined distance is therefore at least $F_k + F_{k+1} = F_{k+2}$. □

Lemma 5. *If envelope a reaches round $i+1$ with i being odd, then $\delta(host_i(a), host_{i+1}(a)) \geq F_{i+1}$ If a is destroyed in odd round i, then $\delta(host_i(a), destroyer_i(a)) \geq F_i$*

Proof. The proof is by induction on the odd round numbers. The basis, round 1, holds trivialy because each envelope travels $1 = F_1 = F_2$ link. So suppose that the lemma holds for round k where $k > 1$ and k is odd. Let a and b be labels of two envelopes in round $k+2$ where envelope b is the immediate successor in round $k+2$ of envelope a. According to the algorithm, envelope a travels F_{k+4} links in round $k+2$ unless it reaches $host_{k+2}(b)$ before travelling this distance. Therefore, we need to estimate $\delta(host_{k+2}(a), host_{k+2}(b))$.
Suppose a is eliminated in round $k+2$. Since $k+2$ is odd, $a > b$. Since a reached round $k+2$, a must have been promoted by a witness for round $k+1$. By lemma 4, the witness for round $k+1$ that could promote a and most closely precedes $host_{k+2}(b)$ is $host_k(b)$. Thus, $\delta(host_{k+2}(a), host_{k+2}(b)) > \delta(host_k(b), host_{k+2}(b)) > F_{k+2}$ by lemma 4. Therefore, the lemma holds for round $k+2$ in this case.
Suppose a survives round $k+2$. If a survives because it travels a distance of F_{k+4} without encountering a processor with round number $k+2$ then the lemma holds trivially for round $k+2$. Otherwise, a travels to $host_{k+2}(b)$ and is promoted because $a < b$. Now, a was promoted from round $k+1$ to round $k+2$ by some witness for round $k+1$ that had label smaller than a. Since $b > a$, that witness cannot be $host_k(b)$. By lemma 3, the witness for round $k+1$ that promotes a must be $host_k(g)$ for some envelope with label g that is between envelope a and envelope b in round $k+1$. By the induction hypothesis, $\delta(host_k(g), host_{k+1}(g)) > F_{k+1}$. Then,

$$\begin{aligned}
\delta(host_{k+2}(a), host_{k+2}(b)) &= \delta(host_k(g), host_{k+2}(b)) \\
&= \delta(host_k(g), host_{k+1}(g)) + \delta(host_{k+1}(g), host_{k+2}(b)) \\
&\geq \delta(host_k(g), host_{k+1}(g)) + \delta(host_k(b), host_{k+2}(b)) \\
&\geq F_{k+1} + F_{k+2} = F_{k+3}
\end{aligned}$$

So the lemma holds for round $k+2$ in this case as well. □

We now bound the savings due to early promotion. Let a and b be the labels of two envelopes in round i where envelope b is the immediate successor in round i of envelope a. We say that envelope a saves k links in round i if, in round i, the distance envelope a travels is $\delta(host_i(a), host_i(b)) - k$.

Corollary 1. *Every envelope that reaches round $i+1$ where i is even saves at least F_i links in round i.*

Proof. Let a be the label of an envelope that remains alive after an even round i, and let b be the label of the envelope in round i that is the immediate successor in round i of envelope a. Then $a > b$ because a survives an even round. According to algorithm ELECT, if envelope a has not already been promoted to round $i+1$ before reaching $\text{host}_{i-1}(b)$, it will achieve early promotion by witness at $\text{host}_i(b)$. But, by Lemma 5, $\delta(\text{host}_{i+1}(b), \text{host}_i(b)) \geq F_i$. □

Theorem 3.1. *Algorithm ELECT sends fewer than $1.271n \log n + O(n)$ messages on rings of size n.*

Proof. To bound the number of messages, we bound (1) the total number of rounds, and (2) the number of messages in any block of two consecutive rounds consisting of an even round followed by an odd round.

By Lemma 5, if round number i is odd, then the distance between any two hosts in round i is at least F_i. Thus, in round i, where i is odd, there can be at most n/F_i remaining envelopes.

Denote by $F^{-1}(x)$ the least integer j such that $F_j \geq x$. It follows that algorithm ELECT uses at most $F^{-1}(n) + O(1)$ rounds for rings of size n.

To estimate the number of messages sent in a block, consider an even round i followed by an odd round $i+1$. Assume that there are x envelopes in round $i+1$. Then, by corollary 1, the total number of links traveled by envelopes in round i is at most $n - xF_i$. Clearly, the total number of links traveled by envelopes in round $i+1$ is at most n. Since, in odd round $i+1$, each envelope travels at most F_{i+1+2} before promotion, this number is also at most xF_{i+3}. Thus, the total number of messages in round i is at most $\min(xF_{i+3}, n)$, and the total number of messages in the block is bounded above by

$\min(n + x(F_{i+3} - F_i)), 2n - xF_i)$. Since $n + x(F_{i+3} - F_i) = 2n - xF_i$ for $x = \frac{n}{F_{i+3}}$, this bound is at most $2n - n\frac{F_i}{F_{i+3}}$

Recall that ϕ denotes $\frac{1+\sqrt{5}}{2}$ and let $\hat{\phi}$ denote $\frac{1-\sqrt{5}}{2}$. Observe that for even i:

$$\begin{aligned}\frac{F_i}{F_{i+3}} = \frac{\frac{1}{\sqrt{5}}(\phi^i - \hat{\phi}^i)}{\frac{1}{\sqrt{5}}(\phi^{i+3} - \hat{\phi}^{i+3})} = \frac{\phi^i - \phi^{-i}}{\phi^{i+3} + \phi^{-i+3}} &= \frac{(\phi^i - \phi^{-i}) \cdot (\phi^{i+3} - \phi^{-i+3})}{(\phi^{i+3} + \phi^{-i+3}) \cdot (\phi^{i+3} - \phi^{-i+3})} \\ &> \frac{\phi^{2i+3} - \phi^3 - \phi^{-3}}{\phi^{2i+6}} \\ &> \phi^{-3} - \phi^{-2i}\end{aligned}$$

Therefore, the number of messages in a block starting with even round i is at most:

$$2n - n\frac{F_i}{F_{i+3}} < n(2 - \phi^{-3} + \phi - 2i) = n\left(2 - \frac{8}{(1+5)^3 + \phi^{-2i}}\right) = n(4 - \sqrt{5} + \phi^{-2i})$$

Hence there are at most

$$\begin{aligned}\sum_{\textbf{even}(i)\textbf{and}2 \le i \le F^{-1}(n)} n(4 - \sqrt{5} + \phi^{-2i}) + O(n) &= \frac{4 - \sqrt{5}}{2} n \log_\phi n + O(n) \\ &< \frac{1.764}{2}(1.441) n \log n + O(n) \\ &< 1.271 n \log n + O(n)\end{aligned}$$

messages sent by any computation of ELECT on a ring of size n. □

4 Undirected rings

There is no immediate obvious way to modify the algorithm to work for undirected, unoriented rings. A stupidly simple solution came to me later than I'd like to admit, but there is nothing unwarranting running the algorithm twice, having each node initiate two processors in both directions. This will result in two independently elected leaders, which can then broadcast their presence to the ring, and decide between who should stay as the single leader, for example one with a greater ID. One may call it extra work, or, more optimistically, free redundancy in case one of the elections fails in some way.