This result indicates that knowledge of the topology does not help to reduce the complexity of finding an MST below the bound of Theorem 7.15. An arbitrary spanning tree of a clique can be constructed in $O(N \log N)$ messages as we shall show in the next section; see also [KMZ84].

7.4 The Korach–Kutten–Moran Algorithm

Many results have been obtained for the election problem, not only for the case of ring networks and arbitrary networks, but also for the case of other specialized topologies, such as clique networks, etc. In several cases the best known algorithms have an $O(N \log N)$ message complexity and in some cases this result is matched by an $\Omega(N \log N)$ lower bound. Korach, Kutten, and Moran [KKM90] have shown that there is a close relation between election and the traversal of networks. Their main result is a general construction of an efficient election algorithm for a class of networks, given a traversal algorithm for this class.

The construction yields $O(N \log N)$ election algorithms for many classes of network; because $O(x)$ (linear) traversal is the best possible, no better algorithm can be obtained with this technique. In contrast, many classes of network with sense of direction do allow better election algorithms, as will be seen in Chapter 11. Also, the time complexity of the Korach–Kutten–Moran algorithm equals its message complexity, and in some cases other algorithms with the same message complexity and lower time complexity are known. The interest of the construction is its generality and the relation that is made between traversal as a "module" in the solution of a higher-level problem, election.

7.4.1 The Modular Construction

The Korach–Kutten–Moran algorithm uses the ideas of both the extinction construction (Subsection 7.3.1) and the Peterson/Dolev–Klawe–Rodeh algorithm (Subsection 7.2.2). The resemblance to the extinction construction is that initiators of the election start a network traversal with a token labeled with their identity. If the traversal completes (decides), the initiator of that traversal becomes elected; the algorithm implies that this happens for exactly one traversal. In this subsection the algorithm is described for the case where the channels satisfy the fifo assumption, but by maintaining a little more information in each token and in each process the algorithm can be adapted for the non-fifo case; see [KKM90].

To deal with the situation of more than one initiator, the algorithm operates in *levels* that can be compared to the rounds of the Peterson/Dolev–Klawe–Rodeh algorithm. If at least two traversals are started, the tokens will arrive at a process that has already been visited by another token. If this situation arises the traversal by the arriving token will be aborted. The goal of the algorithm now becomes to bring two tokens together in one process, where they will be killed and a new traversal initiated. Compare this with the Peterson/Dolev *et al.* algorithm, where at most one of each two identities survives a round and continues to the next round. The notion of rounds is replaced in the Korach–Kutten–Moran algorithm by the notion of levels; two tokens will give rise to a new traversal only if they have the same level, and the newly generated token has a level that is one higher. If a token meets with a token of higher level, or arrives at a node already visited by a token of higher level, the arriving token is simply killed without influencing the token at higher level.

The algorithm is given as Algorithm 7.13. In order to bring tokens of the same level together in one process, each token can be in one of three modes: *annexing*, *chasing*, or *waiting*. A token is represented by (q, l), where q is the initiator of the token and l its level. The variable lev_p gives the level of process p and the variable cat_p gives the initiator of the last annexing token forwarded by p (the *currently active traversal* of p). The variable $wait_p$ is *udef* if no token is waiting at p and its value is q if a token (q, lev_p) is waiting at p. The variable $last_p$ is used for tokens in chasing mode: it gives the neighbor to which p forwarded an annexing token of level lev_p, unless a chasing token was already sent after it; in this case $last_p = udef$. The algorithm interacts with the traversal algorithm by a call to the function *trav*: this function returns either the neighbor to which the token must be forwarded, or *decide* if the traversal terminates.

A token (q, l) is initiated in *annexing* mode and in this mode it obeys the traversal algorithm (as in case IV of Algorithm 7.13) until one of the following situations occurs.

(1) The traversal algorithm terminates: q becomes leader in this case (see Case IV in Algorithm 7.13).
(2) The token arrives at node p of level $lev_p > l$: the token is killed in this case. (This case is implicit in Algorithm 7.13; the conditions in that algorithm all require $l > lev_p$ or $l = lev_p$.)
(3) The token arrives in a node where a token of level l is waiting: the two tokens are killed in this case and a new traversal is started from that node (see Case II in Algorithm 7.13).
(4) The token arrives at a node with level l that has been most recently

```
var lev_p          : integer     init −1 ;
    cat_p, wait_p  : P           init udef ;
    last_p         : Neigh_p     init udef ;

begin if p is initiator then
        begin lev_p := lev_p + 1 ; last_p := trav(p, lev_p) ;
              cat_p := p ; send ⟨ annex, p, lev_p ⟩ to last_p
        end ;
      while ... (* Termination condition, see text *) do
        begin receive token (q, l) ;
              if token is annexing then t := A else t := C ;
              if l > lev_p then (* Case I *)
                begin lev_p := l ; cat_p := q ;
                      wait_p := udef ; last_p := trav(q, l) ;
                      send ⟨ annex, q, l ⟩ to last_p
                end
              else if l = lev_p and wait_p ≠ udef then (* Case II *)
                begin wait_p := udef ; lev_p := lev_p + 1 ;
                      last_p := trav(p, lev_p) ; cat_p := p ;
                      send ⟨ annex, p, lev_p ⟩ to last_p
                end
              else if l = lev_p and last_p = udef then (* Case III *)
                wait_p := q
              else if l = lev_p and t = A and q = cat_p then (* Case IV *)
                begin last_p := trav(q, l) ;
                      if last_p = decide
                        then p announces itself leader
                        else send ⟨ annex, q, l ⟩ to last_p
                end
              else if l = lev_p and ((t = A and
                   q > cat_p) or t = C) then (* Case V *)
                begin send ⟨ chase, q, l ⟩ to last_p ; last_p := udef end
              else if l = lev_p then (* Case VI *)
                wait_p := q
        end
end
```

Algorithm 7.13 The Korach–Kutten–Moran algorithm.

visited by a token of identity $cat_p > q$ (see Case VI) or a *chasing* token (see Case III): the token becomes *waiting* in that node.

(5) The token arrives at a node of level l that has been most recently visited by an *annexing* token with identity $cat_p < q$: the token becomes *chasing* in this case and is sent via the same channel as the previous token (see Case V).

A *chasing* token (q, l) is forwarded in each node via the channel through which the most recently passed token was sent, until one of the following situations occurs.

(1) The token arrives in a process of level $lev_p > l$: the token is killed in this case.
(2) The token arrives in a process with a *waiting* token of level l: the two tokens are removed and a new traversal is started by this process (see Case II).
(3) The token arrives at a process of level l where the most recently passed token was *chasing*: the token becomes *waiting* (see Case III).

A *waiting* token resides in a process until one of the following situations occurs.

(1) A token of higher level arrives at the same process: the waiting token is killed (see Case I).
(2) A token of equal level arrives: the two tokens are removed and a traversal of higher level is started (see Case II).

In Algorithm 7.13 the variables and token information used by the traversal algorithm are ignored. Observe that if p receives a token of level higher than lev_p, this is an annexing token of which p is not the initiator. If a traversal terminates in p, p becomes leader and floods a message to all processes, causing them to terminate.

Correctness and complexity. In order to demonstrate the correctness of the Korach–Kutten–Moran algorithm it will be shown that the number of tokens generated at each level decreases until, at some level, only one token is generated, of which the initiator will be elected.

Lemma 7.22 *If $k > 1$ tokens are generated at level l, at least one and at most $k/2$ tokens are generated at level $l + 1$.*

Proof. At most $k/2$ tokens are generated at level $l + 1$ because when such a token is generated, two tokens of level l are killed at the same time.

Assume that there is a level l such that $k > 1$ tokens are generated at level l, but no token is generated at level $l + 1$. Let q be the process with *maximal* identity that generates a token at level l. The token (q, l) does not complete the traversal, because it will be received by a process p that has already forwarded a different token of level l. When this happens for the first time, (q, l) becomes chasing or, if p has already forwarded a chasing

token, (q, l) becomes waiting. In either case, there are chasing tokens at level l.

Let (r, l) be the token with *minimal* identity after which a chasing token is sent. Token (r, l) is itself not chasing, because a token only chases tokens with smaller identity. We may therefore assume that (r, l) became waiting when it first arrived at a process p' that had already forwarded a different token of level l. Let p'' be the *last* process on the path followed by (r, l) that received, after it forwarded (r, l), an annexing token that turned into a token chasing r. This chasing token either meets (r, l) in p', or abandons the chase if a waiting token was found before the token reached p'. In both cases a token at level $l + 1$ is generated, a contradiction. □

Theorem 7.23 *The Korach–Kutten–Moran algorithm (Algorithm 7.13) is an election algorithm.*

Proof. Assume that at least one process initiates the algorithm. By the previous lemma, the number of tokens generated in each level decreases, and there will be a level, l say, at which exactly one token, say (q, l) is generated. No token of level $l' < l$ completes the traversal, hence none of these tokens causes a process to become elected. The unique token at level l only encounters processes at levels smaller than l, or with $cat = q$ (if it arrives at a process it has already visited), and is forwarded in both cases. Hence the traversal of this token completes and q is elected. □

A function f is said to be *convex* if $f(a) + f(b) \leq f(a + b)$. The complexity of the algorithm is analyzed here under the assumption that an $f(x)$-traversal algorithm (see Section 6.3) is used, where f is a convex function.

Theorem 7.24 *If an $f(x)$-traversal algorithm is used, where f is convex, the KKM election algorithm uses at most $(1 + \log k)[f(N) + N]$ messages if initiated by k processes.*

Proof. If the algorithm is initiated by k processes, at most $2^{-l}k$ tokens are generated at level l, which implies that the highest level is at most $\lfloor \log k \rfloor$.

Each process sends annexing tokens of at most one identity in each level. If in some level l there are tokens with identities $p_1, \ldots, p_j$ and there are N_i processes that have forwarded the annexing token (p_i, l), then it follows that $\sum_{i=1}^{j} N_i \leq N$. Because the traversal algorithm is an $f(x)$-traversal algorithm, annexing token (p_i, l) has been sent at most $f(N_i)$ times, which brings the total number of messages carrying annexing tokens of level l to at most $\sum_{i=1}^{j} f(N_i)$, which is at most $f(N)$ because f is convex. Each process sends at most one chasing token at each level, which gives at most N chasing tokens per level.

So there are at most $(1 + \log k)$ different levels, and at most $f(N) + N$ messages are sent in each level, which proves the result. □

Attiya's construction. A different construction of election algorithm from traversal algorithms was given by Attiya [Att87]. In this construction, the traversal by one token, with identity q say, is not aborted as soon as the token arrives at a process r already visited by another token, of process p say. Instead, the annexing token waits at r but sends out a "hunter" token to chase the token of p and then return to r if p can be successfully defeated. If the hunter returns it is not necessary to start a new traversal, but the current traversal by q's token is continued, thus potentially saving on message complexity. To allow the hunter to return to process r it must be assumed that the network is bidirectional. If an $f(x)$-traversal algorithm is used, the resulting election algorithm has a message complexity of approximately $3.\sum_{i=1}^{N} f(N/i)$.

7.4.2 Applications of the KKM Algorithm

If an $f(x)$-traversal algorithm for a class of networks exists, this class is said to be $f(x)$-traversable. Because many classes of network are $O(x)$-traversable, the construction gives election algorithms with message complexity $O(N \log N)$, and these are the best that can be obtained from the result. In some cases, when sense of direction exists, better algorithms are possible (see Chapter 11).

Election in rings. Rings are x-traversable, hence the KKM algorithm elects a leader in a ring using $2N \log N$ messages. We already know from Theorem 7.13 that this is the best possible.

Election in cliques. Cliques are $2x$-traversable, also without sense of direction, hence the KKM algorithm elects a leader in a clique using $3N \log N$ messages according to Theorem 7.24. A more careful analysis of the algorithm reveals that the complexity is actually $2N \log N + \mathrm{O}(N)$ in this case. Each token is chased using at most three chasing messages, so the total number of chase messages in a computation is bounded by $3.\sum_{i=0}^{\log N+1} 2^{-i}N = \mathrm{O}(N)$. No election algorithm for cliques with a complexity better than $2N \log N + \mathrm{O}(N)$ is known to date. A lower bound of $\Omega(N \log N)$ was proved by Korach, Moran, and Zaks [KMZ84].