Distributed Algorithms for Leader Election on Partially Ordered Keys

Zixue Cheng and Qian-Ping Gu
The University of Aizu
Aizu-Wakamatsu City, Fukushima 965-8580 Japan
email:{z-cheng,qian}@u-aizu.ac.jp

Abstract

Leader election problem is a fundamental problem in distributed computing. The classical leader election problem can be considered as finding the processor with the maximum key in a distributed network in which each processor has one key and a total order is defined on the keys. In this paper, we define a generalized leader election problem which finds all the processors with the maximal keys based on a partial order on the keys. We propose two distributed algorithms for the generalized leader election problem. The first algorithm solves the problem on a network using a spanning tree of the network. The message complexity of the algorithm is $O(mn)$, where m is the number of different keys and n is the number of processors. The second algorithm solves the problem using a coterie of the n processors. The number of messages exchanged on the coterie is $O(\max\{rn, n^{1.5}\})$, where r is the number of the maximal keys.

Key words: Leader election, partial order, distributed network, spanning tree, coterie and quorums, message complexity.

1 Introduction

The leader election problem is a fundamental problem in distributed computing [11, 6, 1, 13]. The problem is to find a processor in a distinguishable computational state from a set of initial processors in the same computational state in a distributed network. The problem can be simplified as finding the maximum key from the n keys held by n processors in the network, each processor has one key and a total order is defined on the keys. The leader election problem has numerous applications in many distributed control problems such as in token-based algorithms; when the token is lost or the owner has failed, the remaining processors elect a leader to issue a new token.

In this paper, we consider a generalized leader election problem. Given n processors of a network, assume that each processor has one key and a partial order is defined on the n keys. The generalized leader election problem is to find all the maximal keys defined by the partial order. Notice that the previous algorithms for the classical leader election problem do not work for the generalized problem defined by a partial order which is not linear.

The generalization is motivated by some distributed applications in computer supported cooperative works and groupware which introduce new distributed problems [3, 18, 7, 15]. Those applications are realized by the cooperations of members/processors interconnected by a computer network. Each of the processors may be characterized by multiple parameters, one parameter states e.g., the ability or experience in a core specialty. The value of a parameter of one processor can be compared with that of the same parameter of another processor but it may not be comparable with the value of a different parameter. Considering the parameters of a processor as the character vector of the processor, the linear order on each parameter then defines a partial order of the character vectors of the processors. Finding the maximal character vectors is an example of the generalized leader election problem and can be applied to the distributed problems such as consensus with partially ordered domain, group decision support systems, and so on [18, 15, 7]. The leader election based on partially ordered keys is also a natural generalization of the classical leader election problem.

We propose two distributed algorithms for the generalized leader election problem. The first algorithm solves the problem of n processors of a network using a spanning tree of the network. The message complexity of the algorithm is $O(mn)$, where m is the number of different keys. The second algorithm solves the problem of n processors using a coterie of the processors. The number of messages exchanged on the coterie is $O(\max\{rn, n^{1.5}\})$, where r is the number of maximal keys.

In the rest of the paper, Section 2 gives the preliminaries. The algorithms based on a spanning tree and a coterie are given in Sections 3 and 4, respectively. Some further research problems are discussed in the final section.

2 Preliminaries

A distributed asynchronous network is a set of processors connected by bidirection communication channels. We consider the networks with arbitrary interconnection topologies. There is no centralized controller, shared memory or global clock in the network. Each processor communicates with others by exchanging messages through the communication channels. Messages can be transmitted independently in both directions on a communication channel and arrive after an unpredictable but finite time delay, without error, and in the FIFO manner. The message complexity of an algorithm is measured in terms of total number of messages that are sent. We assume that the size of a message is $O(\log n)$ bits. We denote the network by an undirected graph $G(V, E)$, where $V = \{p_1, p_2, ..., p_n\}$ is the set of processors (called nodes) in the network and E is the set of communication channels (called edges) between the processors. Each node $p_i \in V$ has one key denoted as k_i.

A partial order $\leq$ on the set $S = \{k_1, ..., k_n\}$ is defined so that (1) $k_i \leq k_i$, (2) $k_i \leq k_j$ and $k_j \leq k_i$ imply $k_i = k_j$, and (3) $k_i \leq k_j \leq k_k$ implies $k_i \leq k_k$. For $k_i, k_j \in S$, if $k_i \leq k_j$ or $k_j \leq k_i$ then we say k_i and k_j *comparable*, otherwise *uncomparable* (denoted as $k_i <> k_j$). A key $k_i \in S$ is called *maximal* if $\forall k_j \in S$ with $k_j \neq k_i$, $k_i \not\leq k_j$. For $k_i, k_j \in S$, if $k_i \leq k_j$ and $k_i \neq k_j$ then $k_i < k_j$. In what follows, we also say k_i is covered by k_j or k_i is smaller than k_j if $k_i < k_j$. For $k_i, k_j \in S$ with $k_i = k_j$ and $i \neq j$, k_i and k_j are considered as the same key. We use m to denote the number of different keys of S ($m \leq n$).

The generalized leader election problem considered in this paper is to find all the maximal keys defined by the partial order $\leq$ on S. We propose two algorithms for this problem. The first one uses a spanning tree of the network to exchange messages. The spanning tree has been used for constructing efficient algorithms for many problems in a distributed systems [19, 16, 14]. A spanning tree of $G(V, E)$ can be found, e.g., by the algorithm in [5] in message complexity $O(n \log n + |E|)$. We assume that a spanning tree of $G(V, E)$ has been established and each node knows its neighbors in the spanning tree.

The second algorithm solves the problem on a coterie of the processors in the network. A *coterie* is a class $C = \{Q_i | Q_i \subseteq V\}$ of subsets of nodes that satisfies the following properties: For any Q_i and Q_j with $i \neq j$, $Q_i \cap Q_j \neq \emptyset$ and $Q_i \not\subseteq Q_j$. The subsets Q_i are called *quorums*. Coteries are a logical structure for achieving coordination among processors and have been used in many distributed problems such as mutual exclusion, replica control, and distributed consensus (including leader election) [12, 9, 10]. The construction of a coterie can be found, for example, at [12, 2]. We assume that a coterie of V has been established and each node knows the other nodes in the same quorums.

In both algorithms, initially each node knows only its own key. A set of arbitrary nodes start the algorithms. On the termination of the algorithms, each node knows all the maximal keys. The algorithms are described by the template introduced in [4].

3 Algorithm on spanning tree

Assume that a spanning tree of $G(V, E)$ has been established and each node knows its neighbors in the spanning tree. The algorithm follows a broadcasting strategy to solve the problem: Every node p_i broadcasts its key k_i over the spanning tree. Node p_i finds the maximal keys from the keys it received. To reduce the message complexity, when a key k_i is known to be covered at some node of the tree, that node stops the broadcasting of k_i to its descendants in the tree.

Now, we give a more detailed outline of the algorithm. Each node $p_i \in V$ broadcasts its key k_i over the spanning tree. Each node which received k_i sends a message to p_i to acknowledge the receipt of k_i. More specifically, each node sends its key to its neighbors to start the broadcasting. For each $p_i \in V$, when p_i receives a key k_l from its neighbor p_j, p_i compares k_l with the keys that p_i has received. If k_l is not covered by any other received key and p_i is not the leaf of the spanning tree then p_i records (k_l, p_j) and sends k_l to all its neighbors except p_j from which k_l is received. Otherwise (either k_l is covered by some received key or p_i is a leaf), p_i stops the broadcasting of k_l to p_i's descendants and sends a message (k_l, ack) to p_j. For each recorded (k_l, p_j), when p_i receives (k_l, ack) from all neighbors except p_j, p_i forwards (k_l, ack) to p_j. When p_i receives (k_i, ack) from all neighbors, p_i knows that the broadcasting of its key has been completed. We assume that the information which identifies the index i of key k_i is sent with k_i in the broadcasting.

For each leaf node p_i of the spanning tree, when the broadcasting for p_i is completed, p_i starts to check if the broadcasting for every node of V has been completed. If so, then each node of V finds the maximal keys from the received keys and terminates its computation.

The algorithm for each node $p_i \in V$ is given in Figure 1. An arbitrary subset V_0 of nodes initiate the computation. We assume that each node p_i has the following states:

- *idle*, the node has not started the computation.
- *active*, the key of the node is broadcasting over the spanning tree.
- *wait-terminate*, the broadcasting of the key has been completed and the node is waiting for the global terminate message.
- *terminated*, the whole computation has been completed.

For each node p_i, let N_i be the set of neighbors of p_i in the spanning tree and S_i be set of keys that p_i has received. Initially, each node p_i is in *idle* state and $S_i = \{k_i\}$.

Theorem 1 *The algorithm given in Figure 1 solves the generalized leader election problem on a network with an arbitrary interconnection topology in $O(mn)$ message complexity, where m is the number of different keys and n is the number of processors.*

Proof: First, it is noticed that the broadcasting for every node is completed in a finite time. Assume that p_i is a node with key k_i maximal. Then for any key k_l with $k_l \neq k_i$, either $k_l < k_i$ or $k_l <> k_i$. If $k_l \neq k_i$ for all $l \neq i$ then the broadcasting for k_i is completed (the state of p_i becomes *wait-terminate*) only after p_i has received (k_i, ack) from all the leaf nodes of the spanning tree that implies every node in the tree has received k_i. Assume that $k_l = k_i$ for some $l \neq i$. Let $V_i = \{p_l | k_l = k_i\}$. Then the states of all the nodes of V_i become *wait-terminate* only after every node in the tree has received k_i. It is also easy to check that message *terminate* is broadcasted only after the states of all the nodes in the tree have become *wait-terminate*. Therefore, when a node receives the message *terminate*, it has received all the maximal keys. Thus, the algorithm solves the problem correctly.

The message complexity for broadcasting a key k_i which is different from any other key is $O(n)$. For the keys $k_i = k_l$ $(i \neq l)$, let E_i and E_l be the sets of edges of the tree on which k_i and k_l have traveled during the broadcasting, respectively. Then $|E_i \cap E_l| \leq 1$ (see Figure 2). From this, the message complexity for broadcasting the key of the nodes in $V_i = \{p_l | k_l = k_i\}$ is $O(n)$. The message complexity for broadcasting *check-terminate* and *terminate* is $O(n)$. Thus, the message complexity of the algorithm is $O(mn)$, where m is the number of different keys. □

In some applications, it may need to identify a unique node which holds a particular maximal key. If $m = n$, i.e., all the keys of S are different then the problem can be solved from the algorithm of Figure 1. However, if there are $k_i, k_l \in S$ with $k_i = k_l$ maximal and $i \neq l$ then further work is needed, because some nodes know that the maximal key comes from node p_i while some others know that from p_l. Let k_i be a maximal key and $V_i = \{p_l | k_l = k_i\}$. We can use the following approach to identify a unique node of V_i. A total order $\leq$ is defined on V_i such that $p_i \leq p_j$ if and only if $i \leq j$. Based on the total order, the classical leader election problem on V_i can be defined. On the termination of the algorithm of Figure 1, the nodes of V_i with k_i maximal identify a unique node by an algorithm for the classical leader election problem on the set V_i. The message complexity of identifying a unique node of V_i is $O(n)$ on the spanning tree. Therefore, the message complexity of

```
Algorithm Leader_Election_on_Tree:
  ▷ Variables: $state_i = idle$; $term_i = 0$; $S_i = \{k_i\}$;
      for $1 \leq j \leq n$, $parent_i^j = nil$, $ack_i^j = 0$.
  ▷ Input: $msg_i = nil$.
      Action if $p_i \in V_0$:
          $state_i :=$active; send $k_i$ to all $u \in N_i$.
  ▷ Input: $msg_i = k_l$ from $p_j \in N_i$.
      Action:
          if $state_i = idle$ then
              {$state_i := active$; send $k_i$ to all $u \in N_i$;};
          if $\forall k_m \in S_i$, $k_m < k_l$ or $k_m <> k_l$ then
              {$S_i := S_i \cup \{k_l\}$;
              if $|N_i| = 1$ then send $(k_l, ack)$ to $p_j$
              else {send $k_l$ to all $u \in (N_i \setminus \{p_j\})$;
                  $parent_i^l := p_j$;}};
          else send $(k_l, ack)$ to $p_j$.
  ▷ Input: $msg_i = (k_l, ack)$ from $p_j \in N_i$.
      Action:
          $ack_i^l := ack_i^l + 1$;
          if $l = i$ and $ack_i^i = |N_i|$ then
              $state_i :=$wait-terminate;
          if $l \neq i$ and $ack_i^l = |N_i| - 1$ then
              send $(k_l, ack)$ to $parent_i^l$.
  ▷ Input: $msg_i = nil$.
      Action when $state_i =$wait-terminate and $|N_i| = 1$:
          send check-terminate to $u \in N_i$.
  ▷ Input: $msg_i =$check-terminate from $p_j \in N_i$.
      Action:
          $term_i := term_i + 1$;
          if $term_i = |N_i| - 1$ and $state_i =$wait-terminate then
              send check-terminate to the node of $N_i$
              from which check-terminate is not received;
          if $term_i = |N_i|$ and $state_i =$wait-terminate then
              {send terminate to all $u \in N_i$; find all the
              maximal keys from those of $S_i$ based on $\leq$;
              $state_i := terminated$;}.
  ▷ Input: $msg_i = terminate$ from $p_j \in N_i$.
      Action when $state_i \neq terminated$:
          send terminate to all $u \in N_i \setminus \{p_j\}$; find all
          the maximal keys from those of $S_i$ based on $\leq$;
          $state_i := terminated$.
```

Figure 1. The algorithm for leader election on a spanning tree.

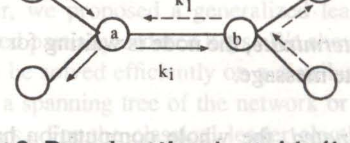

Figure 2. Broadcasting k_i and k_l $(k_i = k_l)$.

identifying a unique node for every maximal key is $O(rn)$, where r is the number of maximal keys. Notice that $r \leq m$.

4 Algorithm on coterie

Assume that a coterie $C = \{Q_1, ..., Q_l\}$ of n nodes has been established. For each node p_i, let $I_i = \{j | p_i \in Q_j\}$ and $C_i = \cup_{j \in I_i} Q_j$. Assume that each node p_i knows the nodes of C_i. An outline of the algorithm is as follows. Each node $p_i \in V$ sends its key k_i to the nodes of C_i. (We assume that the information which identifies the index i of key k_i is sent with k_i to the nodes of C_i.) For each node p_i, when p_i receives the keys from all the nodes of C_i, p_i finds the maximal keys from the received keys. For each received key k_j, if k_j is maximal then p_i sends message *uncovered* to p_j; otherwise, p_i sends message *covered* to p_j. When p_i receives *covered* from some node of C_i, p_i knows that its key k_i is not maximal and p_i enters the *wait-terminate* state. When p_i receives *uncovered* from all the nodes of C_i, p_i knows that its key k_i is maximal and broadcasts k_i to all nodes of V via the coterie. Each node of V sends p_i a message via the coterie to acknowledge the receipt of k_i. When p_i receives the acknowledgements from all the nodes of V, p_i enters the *wait-terminate* state.

If there are $k_j, k_l \in S$ with $k_j = k_l$ maximal and $j \neq l$ then the maximal key k_j may be broadcasted by nodes p_j and p_l. This is not efficient in the sense of message complexity. We use the following approach to reduce the number of messages. For each node $p_i \in V$ and each key k_j received by p_i, define $V_{ij} = \{p_l | p_l \in C_i, k_l = k_j\}$. When node p_i finds k_j is maximal among the received keys, p_i selects only one node of V_{ij} for broadcasting k_j. More precisely, p_i sends *uncovered* to the node of V_{ij} with the largest index and *covered* to all the other nodes of V_{ij}.

For each node $p_i \in V$, when p_i enters the *wait-terminate* state, p_i starts to check if all the nodes of V are in the *wait-terminate* state. If so, p_i terminates its computation.

Figure 3 gives the algorithm for each node p_i. To simplify the description of the algorithm, when we say node p_i sends a message to all nodes of C_i, we mean that the message is sent to all the nodes, including p_i itself, of C_i. An arbitrary subset V_0 of nodes initiate the computation. We assume that each node p_i have the following states:

- *idle*, the node has not started the computation.
- *active*, the node is finding the maximal keys.
- *wait-terminate*, the node is waiting for the global terminate message.
- *terminated*, the whole computation has been completed.

```
Algorithm Leader_Election_on_Coterie:
  ▷ Variables: $state_i = idle$; $S_i = \{k_i\}$; $Max_i = \emptyset$;
      $max_i = 0$; $term1_i$; $term2_i = 0$; $ack1_i = 0$;
      for $1 \leq j \leq |C_i|$, $ack2_i^j = 0$.
  ▷ Input: $msg_i = nil$.
    Action if $p_i \in V_0$:
      $state_i$ :=active; send $k_i$ to all $u \in C_i$.
  ▷ Input: $msg_i = k_j$ from $p_j \in C_i$.
    Action:
      $S_i := S_i \cup \{k_j\}$;
      if $state_i = idle$ then
        {$state_i := active$; send $k_i$ to all $u \in C_i$;};
      if $|S_i| = |C_i|$ then
        {find the maximal keys from $S_i$; $\forall k_j \in S_i$,
        if $k_j$ is maximal and $j = \max\{l | p_l \in V_{ij}\}$ then
          send $p_j$ uncovered
        else send $p_j$ covered;}.
  ▷ Input: $msg_i = covered$ from $p_j \in C_i$.
    Action when $state_i = active$:
      $state_i$ :=wait-terminate;
      send check-terminate to all $u \in C_i$;
  ▷ Input: $msg_i = uncovered$ from $p_j \in C_i$.
    Action when $state_i = active$:
      $max_i := max_i + 1$;
      if $max_i = |C_i|$ then send $(k_i, max)$ to all $u \in C_i$.
  ▷ Input: $msg_i = (k_j, max)$ from $p_j \in C_i$.
    Action:
      send $(k_j, max, f)$ to all $u \in C_i$.
  ▷ Input: $msg_i = (k_l, max, f)$ from $p_j \in C_i$.
    Action:
      $Max_i := Max_i \cup \{k_l\}$; send $(k_l, ack)$ to $p_j$.
  ▷ Input: $msg_i = (k_l, ack)$ from $p_j \in C_i$.
    Action:
      $ack2_i^l := ack2_i^l + 1$;
      if $ack2_i^l = |C_i|$ then send $ack$ to $p_l$.
  ▷ Input: $msg_i = ack$ from $p_j \in C_i$.
    Action:
      $ack1_i := ack1_i + 1$;
      if $ack1_i = |C_i|$ then
        {$state_i$ :=wait-terminate;
        send check-terminate to all $u \in C_i$;}.
  ▷ Input: $msg_i$ =check-terminate from $p_j \in C_i$.
    Action:
      $term1_i := term1_i + 1$;
      if $term1_i = |C_i|$ then send terminate to all $u \in C_i$;
  ▷ Input: $msg_i = terminate$ from $p_j \in C_i$.
    Action:
      $term2_i := term2_i + 1$;
      if $term2_i = |C_i|$ then
        {$state_i := terminated$;
        terminates the computation;}.
```

Figure 3. The algorithm for leader election on a coterie.

Theorem 2 *The algorithm of Figure 3 solves the generalized leader election problem. The number of messages exchanged on the coterie is $O(\max\{rn, n^{1.5}\})$, where r is the number of maximal keys.*

Proof: We first show the correctness of the algorithm. Let p_i and p_j be any two nodes of V. The definition of the coterie guarantees that C_i and C_j has a common node p_k. Since both p_i and p_j send their keys k_i and k_j to p_k, these two keys are compared there. Therefore, the key k_i is compared with all the other keys. After this, if k_i is not covered by any key then it is maximal otherwise it is not. Let k_j be a maximal key. If for all k_l with $l \neq j$, $k_l \neq k_j$ then node p_j receives only *uncovered* and k_j is broadcasted to all nodes of V. If there are keys $k_{j_1}, ..., k_{j_l}$ with $k_{j_i} = k_j$ and $j_i \neq j$ $(1 \leq i \leq l)$ then one node (the node with the largest index) of p_j and $p_{j_1}, ..., p_{j_l}$ receives only *uncovered* and k_j is broadcasted to all nodes of V. Obviously, all the maximal keys are found and broadcasted to every node of V in a finite time.

For each node p_i with key k_i maximal, p_i enters state *wait-terminate* only after all the nodes of V have received k_i. p_i sends message *terminate* only after all the nodes of C_i have been in state *wait-terminate*. Therefore, p_i terminates only after all nodes of V have received all the maximal keys. Obviously, the algorithm terminates in a finite time.

The number of messages for determining if k_i is maximal for all $p_i \in V$ is $O(\sum_{i=1}^{n} |C_i|)$. The number of messages for broadcasting one maximal key k_i is $O(|C_i| + \sum_{p_j \in C_i} |C_j|)$. Assume that $k_1, ..., k_r$ are the maximal keys. Then the number of messages for broadcasting all the maximal keys are $O(\sum_{i=1}^{r}(|C_i| + \sum_{p_j \in C_i} |C_j|))$. The number of messages for termination is $O(\sum_{i=1}^{n} |C_i|)$.

To show the number of messages of the theorem, we use the coterie introduced by Agrawal and Jalote [2]. The coterie is constructed as follows. Assume that $n = m(m-1)/2$ for some integer m. Create a complete graph K_m with vertices $\{1, 2, ..., m\}$ and n edges $\{(i, j) | i, j \in \{1, 2, ..., m\}, i \neq j\}$. Do a one to one mapping from the set of n nodes of V to the n edges of K_m. For each vertex i of K_m, let E_i be the set of edges incident to i. A quorum Q_i is defined as the set of nodes which are mapped to the edges in E_i. The coterie is defined as $C = \{Q_i | i \text{ is a vertex of } K_m\}$. For example, let $V = \{p_1, p_2, ..., p_6\}$. The one to one mapping between V and the set of edges of K_4 is

$$(p_1, (1,2)), (p_2, (2,3)), (p_3, (3,4)),$$
$$(p_4, (4,1)), (p_5, (1,3)), (p_6, (2,4)).$$

The coterie C is:

$$\{\{p_1, p_4, p_5\}, \{p_1, p_2, p_6\}, \{p_2, p_3, p_5\}, \{p_3, p_4, p_6\}\}.$$

For the nodes of $V = \{p_1, p_2, p_3, p_4, p_5, p_6\}$,

$$C_1 = \{p_1, p_4, p_6, p_2, p_5\}, C_2 = \{p_2, p_6, p_3, p_1, p_5\},$$
$$C_3 = \{p_3, p_6, p_2, p_5, p_4\}, C_4 = \{p_4, p_5, p_3, p_1, p_6\},$$
$$C_5 = \{p_5, p_1, p_2, p_3, p_4\}, C_6 = \{p_6, p_2, p_3, p_1, p_4\}.$$

Using the coterie of [2], $|C_i| = O(\sqrt{n})$ for $1 \leq i \leq n$. From this, the number of messages for termination and determining if k_i is maximal for all $p_i \in V$ is

$$O(\sum_{i=1}^{n} |C_i|) = O(n^{1.5}).$$

The number of messages for broadcasting r maximal keys are

$$O(\sum_{i=1}^{r}(|C_i| + \sum_{p_j \in C_i} |C_j|)) = O(rn).$$

Thus, the number of messages of the algorithm on the coterie is $O(\max\{rn, n^{1.5}\})$, where r is the number of maximal keys. □

Now, we give an example on the above algorithm. For $V = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, let C be the Agrawal-Jalote's coterie on V, and $C_i = \cup_{j \in I_i} Q_j$ $(1 \leq i \leq 6)$, as defined in the proof of Theorem 2. The keys of processors of V are given in Figure 4. Initially, p_3 starts the computation. After each p_i has received all the keys from the processors in C_i,

$$S_1 = \{(2,3), (1,0), (1,1), (1,4), (2,3)\},$$
$$S_2 = \{(1,4), (1,1), (2,2), (2,3), (2,3)\},$$
$$S_3 = \{(2,2), (1,1), (1,4), (2,3), (1,0)\},$$
$$S_4 = \{(1,0), (2,3), (2,2), (2,3), (1,1)\},$$
$$S_5 = \{(2,3), (2,3), (1,4), (2,2), (1,0)\},$$
$$S_6 = \{(1,1), (1,4), (2,2), (2,3), (1,0)\}.$$

Processor p_1 sends *uncovered* to p_2, p_5 and *covered* to p_1, p_4, p_6; processor p_2 sends *uncovered* to p_2, p_5 and *covered* to p_1, p_3, p_6; processor p_3 sends *uncovered* to p_2, p_5 and *covered* to p_3, p_4, p_6; processor p_4 sends *uncovered* to p_5 and *covered* to p_1, p_3, p_4, p_6; processor p_5 sends *uncovered* to p_2, p_5 and *covered* to p_1, p_3, p_4; and processor p_6 sends *uncovered* to p_1, p_2 and *covered* to p_3, p_4, p_6. After this, p_2 and p_5 receive only *uncovered* and each of p_1, p_3, p_4, p_6 receives at least one *covered*. Finally, key $(1,4)$ of p_2 and key $(2,3)$ of p_5 are broadcasted to all processors.

5 Concluding remarks

In this paper, we proposed a generalized leader election problem based partially ordered keys. We showed that this problem can be solved efficiently on a distributed network either using a spanning tree of the network or a coterie of the processors. For the classical leader election problem based on totally ordered keys, the problem can be solved in message complexity $O(n \log n)$ on a complete connected

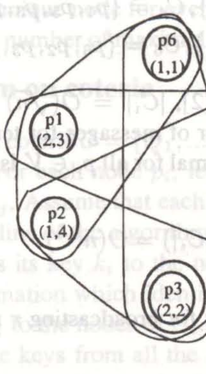

Figure 4. The keys in the coterie.

network [8] or a logical structure called k-dimensional arrays of the nodes [17]. The transitive property of the linear order is critical to achieve the message complexity bound of $O(n \log n)$. For the partial order $\leq$, two keys may be uncomparable and furthermore the uncomparable relation $<>$ is not transitive (for example, we do not know the relation between k_i and k_l if we do not compare them, even though the information $k_i <> k_j$ and $k_j <> k_l$ are known). Due to the property of the relation $<>$, the algorithms of [8, 17] do not work on the generalized leader election problem. Let r be the number of the maximal keys and m be the number of different keys of the n processors in a network. If $r = m$ then the message complexity $O(mn)$ of our first algorithm is optimal. However, whether $O(mn)$ can be reduced further is open for $r < m$. For solving the problem on a logical structure of n processors such as the coterie or k-dimensional array, when $r \geq n^{0.5}$, the number of messages $O(\max\{rn, n^{1.5}\})$ of our second algorithm is optimal. Whether $O(n^{1.5})$ can be reduced further for $r < n^{0.5}$ is open as well.

References

[1] Y. Afek, and E. Gafni, "Time and message bounds for election in synchronous and asynchronous complete networks," *SIAM J. Computing*, Vol. 20, No. 2, pp. 376-394, 1991.

[2] G. Agrawal and P. Jalote. An efficient protocol for voting in distributed systems. In *Proc. of the 12th IEEE International Conference on Distributed Computing Systems*, pages 640–647, 1992.

[3] M. Barborak, M. Malek, and A. Dahbura, "The consensus problem in fault-tolerant computing," *ACM Computing Surveys*, Vol. 25, No. 2, 1993.

[4] V.C. Barbosa, *An Introduction to Distributed Algorithms*, The MIT Press, 1996

[5] R.G. Gallager, P.A. Humblet, and P.M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Trans. on Programming Language Systems*, Vol.5 pp. 66-77, 1983.

[6] G. Singh, "Leader election in the presence of link failures," *IEEE Trans. on Parallel and Distributed Systems*, Vol 7. No.3, pp. 231-236, 1996.

[7] T. Ito and T. Shintani, "On a persuasion mechanism among agents for group choice design support systems," *IEICE Trans. D-II, Vol. J80-D-II*, No. 9, pp. 1-9, 1997.

[8] E. Korach, S. Moran, and S. Zaks, "Tight lower and upper bounds for some distributed algorithms for a complete network of processors," *Proc. of ACM-PODC'84*, pp. 199-207, 1984.

[9] A. Kumar, "Hierarchical quorum consensus: a new method for managing replicated data," *IEEE trans. on Computers*, Vol. 40, no. 9, pp. 996-1104, 1991.

[10] T.V. Lakshman and A.K. Agrawala, "Efficient decentralized consensus protocols," *IEEE Trans. Softw. Eng.*, Vol. SE-12, no. 5, pp. 600-607, 1986.

[11] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[12] M. Maekawa, "A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems," *ACM Trans. Comput. Syst.*, Vol. 3, no. 2, pp. 145-159, 1995.

[13] T. Masuzawa, N. Nishikawa, K. Hagihara, and N. Tokura, "Optimal fault-tolerant distributed algorithms for election in complete networks with a global sense of direction," *Proc. Third Int'l Workshop on Distributed Algorithms*, 1989.

[14] K. Raymond, "A tree-based algorithm for distributed mutual exclusion," *ACM Trans. on Computer Systems*, Vol. 7, No. 1, 1989, pages 61-77, 1989.

[15] I. Shimojo, T. Tachikawa, and M. Takizawa, "Distributed consensus protocols with partially ordered domain," *Technical Report of IEICE, 97-DPS-83-7*, pp. 37-42, 1997.

[16] M.M. Wu and M.C. Loui, "An efficient distributed algorithm for maximum matching in general graphs," *Algorithmica* 5 pp. 383-406, 1990.

[17] S. Yuan and A.K. Agrawala, "A class of optimal decentralized commit protocols," *Proc. of the 8th International Conference on Distributed Computing Systems*, pp. 234-241, 1988.

[18] C. Yahata and M. Takizawa, "General protocols for consensus in distributed systems," *Proc. of DEXA (Lecture Notes in Computer Science, No. 978, Springer-verlag)*, pp. 227-236, 1995.

[19] S. Zaks, "Optimal distributed algorithms for sorting and ranking," *IEEE Trans. Computers*, Vol. C-34, pp. 376-379, 1985.