Coterie Templates: A New Quorum Construction Method*

WEE K. NG[†] CHINYA V. RAVISHANKAR

Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, MI 48109-2122
E-mail: {wkn,ravi}@eecs.umich.edu

Abstract

One approach to distributed mutual exclusion algorithms is the use of quorums. Quorum-based algorithms offer the advantage of protocol symmetry, spreading effort and responsibility uniformly across the distributed system. In this paper, we present an $O(\log n)$ algorithm to generate coterie templates *of near-optimal $O(n^{0.63})$ size. Coterie templates are generic quorum structures that exhibit several desirable properties such as fault tolerance, symmetry and low storage cost. In addition, coteries can be* instantiated *from the template to reflect desirable network characteristics.*

1 Introduction

Achieving mutual exclusion is a central problem in parallel and distributed programming. Distributed systems present additional difficulties such as sites having no memory in common, messages lost during communication, sites failing independently, etc. When no coordinator process is present, sites must implement their solution to distributed mutual exclusion cooperatively. Many algorithms and protocols have been proposed for distributed mutual exclusion based on the notions of *quorums* and/or *tokens* [1, 3, 4, 6, 7, 8, 10, 12, 13]. Quorum-based protocols offer a notion of *symmetry*, which serves to distribute algorithm overheads equally across the system. It is necessary to balance the extent to which each site is involved in the enforcement of distributed mutual exclusion, the extent to which each site is aware of the status of other sites, and the extent to which a site is required to inform other sites of its status. These symmetry requirements are satisfied when the size of all quorums are equal and when each site is included in an equal number of quorums.

In this paper, we propose an $O(\log n)$ algorithm that generates symmetric *coterie templates*. A coterie template is a generic quorum structure that exhibits several desirable properties such as fault-tolerance, symmetry and low storage cost. A template may be *instantiated* to give arbitrary quorums, which may then be adopted by a distributed system for enforcing distributed mutual exclusion.

This paper is organized as follows. Section 2 discusses the problem of constructing symmetric and optimum quorums. Section 3 presents an efficient algorithm to construct fault-tolerant, symmetric and near-optimum quorums in the form of templates. It also reviews the algorithm and illustrates some desirable properties of the algorithm and the template it generates. The final section concludes the paper.

2 Coteries and Their Existence

2.1 Problem formulation

Coteries were first proposed by Barbara and Garcia-Molina [7] as a generalization of the notion of quorums by requiring additional properties. In this section, we examine the problem of coterie construction. We first define a problem we call *Symmetric Coterie Construction (SCC)*.

Definition 2.1 (SCC) *Given a finite set $S = \{1, 2, \ldots, n\}$ representing the sites of a network, find $n = |S|$ subsets $Q_i \subseteq S, Q_i \neq \phi$, such that:*

(1) *(Covering)* $\bigcup_{i=1}^{n} Q_i = S$.
(2) *(Minimality)* $Q_i \not\subset Q_j, 1 \leq i, j \leq n, i \neq j$.
(3) *(Equal Size)* $|Q_i| = \kappa, 1 \leq i \leq n$.
(4) *(Equal Effort)* $|\{Q_j | i \in Q_j\}| = \kappa, 1 \leq i \leq n$.
(5) *(Mutual Intersection)* $Q_i \cap Q_j \neq \phi$, $1 \leq i, j \leq n, i \neq j$.

The subsets Q_i's are called *quorums*. Quorums that satisfy the *Minimality*, *Mutual Intersection* and non-emptiness properties form a coterie. The *Equal Size*

*This work was supported in part by the Consortium for International Earth Science Information Networking.

[†]Author's World Wide Web home page is available at "http://www.eecs.umich.edu/~wkn''.

$$\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
2 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
3 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
4 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
6 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
7 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
\end{array}
\qquad
\begin{array}{c|ccccccc|}
 & 3 & 1 & 4 & 2 & 5 & 7 & 6 \\
\hline
1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
2 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
3 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
4 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
6 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
7 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
\end{array}$$

(a) (b)

Figure 2.1: Two symmetric coteries of size 7 under the same template.

and *Equal Effort* properties define the additional notion of *symmetry* in the coterie. Equal-sized quorums ensure that each site sends the same number of messages to request for shared resource. Given the random distribution of quorum accumulation requests, the *Equal Effort* property requires that each site be included in the same number of quorums, ensuring that all sites expend the same effort in enforcing distributed mutual exclusion. In addition it also ensures that the failure of one site has the same impact on the network as any other site. The last property requires any two quorums to intersect, though it is not required that they contain the same number of common sites.

Example 2.1 An example of a symmetric coterie for a network of 7 nodes is shown in the *quorum matrix* in Figure 2.1a. An $n \times m$ matrix $\mathbf{X} = \{x_{i,j}\}$ is a quorum matrix where

$$x_{i,j} = \begin{cases} 1 & \text{if node } j \in Q_i \text{, the quorum of node } i \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

In the figure, $S = \{1,2,3,4,5,6,7\}$, $n = 7$, $\alpha = \beta = 3$ and the coterie is the set {{1,2,4}, {2,3,5}, {3,4,6}, {4,5,7}, {5,6,1}, {6,7,2}, {7,1,3}}. Incidentally, $|Q_i \cap Q_j| = 1$ for all $1 \leq i,j \leq n$. However, it is not required that $|Q_i \cap Q_j| = |Q_k \cap Q_\ell|$ in our definition of SCC. ∎

Note that the quorum matrix actually defines a *coterie template.* Suppose we permutate the column labels in Figure 2.1. This results in a different coterie: {{1,2,3}, {1,4,5}, {2,4,7}, {2,5,6}, {3,5,7}, {1,6,7}, {3,4,6}} (Figure 2.1b). Thus, the column labels constitute a permutation π, which produces an *instantiation* of a quorum matrix.

The construction of a coterie from a set of n nodes is closely related to the problem of *block design* in combinatorics. Block designs are arrangements of objects into sets, called *blocks,* such that various conditions on the number of occurrences of objects, or pairs of objects, and sometimes of other things are satisfied. In particular, the problem of coterie construction is *almost* equivalent to the problem of constructing a *balanced incomplete block design, (BIBD)* [11]. A BIBD is specified as a (b, v, r, k, λ)-configuration with $\lambda > 0$, $k < v - 1$, where we are given a set of v varieties and the problem is to design a collection of b blocks, which are sets of varieties such that (1) each block contains k varieties, (2) each variety appears in exactly r blocks, and (3) any two blocks have exactly λ common varieties. A (v, k, λ)-configuration is a restriction to $b = v$ and $r = k$. It is unknown whether a (b, v, r, k, λ)-configuration exists for arbitrary values of b, v, r, k, λ, and whether there are any general algorithms to construct them even when they do exist.

Clearly, the construction of a coterie is similar to the construction of a (v, k, λ)-configuration. Each block corresponds to a quorum and the set of varieties corresponds to S. The only difference lies in criterion 3 above: we do not require λ to be constant for coteries. The similarity with *BIBD* suggests that the coterie construction problem is also hard.

2.2 Previous work

Several attempts have been made in the past to generate optimal symmetric quorums. Maekawa [8] made the insightful observation that symmetric quorums are related to *finite projective planes (FPP)* [2]. Quorums correspond to the lines in the plane, and have equal sizes and intersect in exactly one site. FPPs are related to BIBDs as follows [2]:

Theorem 2.1 *A finite projective plane of order n is equivalent to a (v, k, λ)-configuration with parameters $v = n^2 + n + 1$, $k = n + 1$, and $\lambda = 1$.*

Given a system of n nodes, his algorithm begins by constructing an FPP of order n. It is known [2] that there exists a FPP of order k if k is a power p^m, where p is prime. It is not known whether this is also a sufficient condition. Since FPPs exist only for some n, Maekawa proposes two methods to construct quorums for the non-existent cases.

The first method introduces enough dummy nodes into the system so that its size reaches n', where $n' > n$ is the smallest number larger than n for which an FPP exists. Doing so produces optimal symmetric quorums for a network of n' nodes. However, n' is a lot larger than n, and the quorums are not symmetric.

The second method arranges sites into a square grid so that each site takes as its quorum all sites in the row and column passing through itself. This produces symmetric quorums of size $2\sqrt{n}$. The limitations of this method is that n must be a perfect square, and the quorum size is non-optimal. Again, the quorums are not symmetric.

The association of quorums with finite projective planes provides some theoretical insights into the existence of symmetric quorums. However, it does not produce a general method to generate symmetric and optimal quorums for all n. As FPPs are subsumed as a special case of SCC and because the construction of FPPs is open [5], SCC is an open problem.

3 Template Construction

As fully symmetric coteries (with constant λ) do not exist for all n, we relax the constraint that λ be constant, and present an algorithm QGEN that generates symmetric quorums of size close to the optimum.

3.1 Algorithm QGEN

Algorithm QGEN generates quorum sets for a complete, undirected graph. The method starts with a trivially constructed coterie and reduces the size of the quorums iteratively while preserving the properties of a coterie. This section illustrates the operation of algorithm QGEN on a graph of size $n = 22$ (Figure 3.1). Given a complete graph, we represent its coterie as an $n \times n$ quorum matrix $X = \{x_{i,j}\}$ as defined in Equation 2.1. Upon termination of the algorithm, each quorum is a union of disjoint *runs*, where a run is a set of consecutively numbered nodes within a row. The objective is to find n symmetric quorums for this graph. The following presents some notations that we use in our discussions. As the algorithm is iterative, many of our symbols will be decorated with the iteration number, for which we use the symbol ℓ.

Q_ℓ^i denotes the quorum of node i at the ℓth iteration, $0 \leq i \leq n-1$, $1 \leq \ell \leq \lfloor \log_3 n \rfloor$. We shall sometimes omit the superscript and use Q_ℓ when referring to the quorum of an arbitrary node.

$R_{\ell,r}^i$ denotes the rth run in quorum Q_ℓ^i. Since the number of runs doubles at each iteration, at the ℓth iteration, we will have $Q_\ell^i = \bigcup_{r=1}^{2^\ell} R_{\ell,r}^i$. The matrix initially has a single run in each row, so $Q_0^i = R_{0,1}^i$. As in the above notation, we shall use $R_{\ell,r}$ when no confusion arises.

k_ℓ denotes the size of a quorum at the ℓth iteration, so $k_\ell = |Q_\ell|$. The initial quorum size is defined as follows:

$$k_0 = \begin{cases} x+2 & \text{if } x \bmod 3 = 0 \\ x+1 & \text{if } x \bmod 3 = 1 \\ x & \text{if } x \bmod 3 = 2 \end{cases} \quad (3.1)$$

where $x = \lfloor n/2 \rfloor + 1$. The definition ensures that k_0 is the smallest integer between $\lfloor n/2 \rfloor + 1$ and n such that $k_0 + 1$ is divisible by 3. The reason why 3 is chosen will be explained later.

x_ℓ denotes the size of a run at the ℓth iteration, i.e. $x_\ell = |R_{\ell,r}|$. Initially, $x_0 = k_0$. Since a quorum is a union of runs, $k_\ell = 2^\ell \cdot x_\ell$.

The algorithm begins with an initial non-optimal coterie by associating with each node i a quorum Q_0^i of more than half of the total nodes in the graph. Since $|Q_0^i| \geq n/2$ the mutual intersection property is satisfied, and a coterie is formed. A systematic way of such an initialization is to assign $Q_0^i = \{[i]_n, [i+1]_n, \ldots, [i+k_0-1]_n\}$ where $[j]_n$ denotes j modulo n (Figure 3.1a) where $k_0 = 14$. The first iteration reduces the quorum sizes by partitioning a quorum into disjoint runs, and the new quorum is a union of a subset of these runs. As $|Q_0^i| + 1 = k_0 + 1 = 15$ is divisible by 3, the quorum Q_0^i may be partitioned into 3 runs. For example, $\{0,1,2,3,4\}, \{5,6,7,8\}$ and $\{9,10,11,12,13\}$ are the 3 runs for the quorum of node 0. The algorithm discards the middle run and forms a new quorum $Q_1^0 = R_{1,1}^0 \cup R_{1,2}^0 = \{0,1,2,3,4\} \cup \{9,10,11,12,13\}$. The matrix after this iteration $\mathbf{X}^1$ still forms a coterie (Figure 3.1b). The formal proof of this claim is given in Section 3.2. Note that the size of the middle discarded run is 1 less than the outer 2 runs, so that the size of the original run is $3x - 1$ for some positive integer x. Another iteration is possible since $|R_{1,1}^0| = |R_{1,2}^0|$ is large enough to be partitioned further, giving an even smaller quorum and yielding the matrix $\mathbf{X}^2$ (Figure 3.1c). The process is repeated until the smallest quorum size is reached. In this case, the optimal coterie is found after two iterations. The final quorum size is 8 ($\approx 22^{0.63}$). It is formally presented in Algorithm QGEN (Figure 3.2).

The function QGEN (line 29) accepts the size of the graph as a parameter and returns a quorum array *Quorum*, corresponding to the quorum of node 0. The quorums of other nodes may be determined by right-shifting this quorum, as shown by the quorum matrices in Figure 3.1. There are three routines in Figure 3.2. Function ADJUST increments the size of a run $|R_{\ell,r}|$ if necessary, so that $|R_{\ell,r}| + 1$ is divisible by 3. Procedure PARTITION performs the subdivision process as described earlier. It first computes the run size (line 13) and determine if further partitioning is possible (line 14). A run can be partitioned if and only if its size plus 1 is divisible by 3 and is more than 7 (line 15–20). Otherwise the terminal cases for the iteration are handled explicitly as shown in procedure PARTITION (line 21–27) when the sizes have reached 4, 5, 6 or 7. Function QGEN, the main func-

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	.	.	.	.	.	.	.	.
1	.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	.	.	.	.	.	.	.
2	.	.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	.	.	.	.	.	.
3	.	.	.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	.	.	.	.	.
4	.	.	.	.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	.	.	.	.
5	.	.	.	.	.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	.	.	.
6	.	.	.	.	.	.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	.	.
7	.	.	.	.	.	.	.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	.
8	.	.	.	.	.	.	.	.	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	1	.	.	.	.	.	.	.	.	1	1	1	1	1	1	1	1	1	1	1	1	1
10	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	1	1	1	1	1	1	1
11	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	1	1	1	1	1
13	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	1	1	1	1
14	1	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	1	1	1
15	1	1	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	1
17	1	1	1	1	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1
18	1	1	1	1	1	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1
19	1	1	1	1	1	1	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1
20	1	1	1	1	1	1	1	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1
21	1	1	1	1	1	1	1	1	1	1	1	1	1	.	.	.	.	.	.	.	.	1

(a) $\boldsymbol{X^0}$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	1	1	1	1	1	.	.	.	.	1	1	1	1	1	.	.	.	.	.	.	.	.
1	.	1	1	1	1	1	.	.	.	.	1	1	1	1	1	.	.	.	.	.	.	.
2	.	.	1	1	1	1	1	.	.	.	.	1	1	1	1	1	.	.	.	.	.	.
3	.	.	.	1	1	1	1	1	.	.	.	.	1	1	1	1	1	.	.	.	.	.
4	.	.	.	.	1	1	1	1	1	.	.	.	.	1	1	1	1	1	.	.	.	.
5	.	.	.	.	.	1	1	1	1	1	.	.	.	.	1	1	1	1	1	.	.	.
6	.	.	.	.	.	.	1	1	1	1	1	.	.	.	.	1	1	1	1	1	.	.
7	.	.	.	.	.	.	.	1	1	1	1	1	.	.	.	.	1	1	1	1	1	.
8	.	.	.	.	.	.	.	.	1	1	1	1	1	.	.	.	.	1	1	1	1	1
9	1	.	.	.	.	.	.	.	.	1	1	1	1	1	.	.	.	.	1	1	1	1
10	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	.	.	.	.	1	1	1
11	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	.	.	.	.	1	1
12	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	.	.	.	.	1
13	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	.	.	.	.
14	.	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	.	.	.
15	.	.	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	.	.
16	.	.	.	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1	.
17	.	.	.	.	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1	1
18	1	.	.	.	.	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1	1
19	1	1	.	.	.	.	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1	1
20	1	1	1	.	.	.	.	1	1	1	1	1	.	.	.	.	.	.	.	.	1	1
21	1	1	1	1	.	.	.	.	1	1	1	1	1	.	.	.	.	.	.	.	.	1

(b) $\boldsymbol{X^1}$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	1	1	.	1	1	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.
1	.	1	1	.	1	1	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.
2	.	.	1	1	.	1	1	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.
3	.	.	.	1	1	.	1	1	.	.	.	.	1	1	.	1	1	.	.	.	.	.
4	.	.	.	.	1	1	.	1	1	.	.	.	.	1	1	.	1	1	.	.	.	.
5	.	.	.	.	.	1	1	.	1	1	.	.	.	.	1	1	.	1	1	.	.	.
6	.	.	.	.	.	.	1	1	.	1	1	.	.	.	.	1	1	.	1	1	.	.
7	.	.	.	.	.	.	.	1	1	.	1	1	.	.	.	.	1	1	.	1	1	.
8	.	.	.	.	.	.	.	.	1	1	.	1	1	.	.	.	.	1	1	.	1	1
9	1	.	.	.	.	.	.	.	.	1	1	.	1	1	.	.	.	.	1	1	.	1
10	1	1	.	.	.	.	.	.	.	.	1	1	.	1	1	.	.	.	.	1	1	.
11	.	1	1	.	.	.	.	.	.	.	.	1	1	.	1	1	.	.	.	.	1	1
12	1	.	1	1	.	.	.	.	.	.	.	.	1	1	.	1	1	.	.	.	.	1
13	1	1	.	1	1	.	.	.	.	.	.	.	.	1	1	.	1	1	.	.	.	.
14	.	1	1	.	1	1	.	.	.	.	.	.	.	.	1	1	.	1	1	.	.	.
15	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	1	1	.	1	1	.	.
16	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	1	1	.	1	1	.
17	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	1	1	.	1	1
18	1	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	1	1	.	1
19	1	1	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	1	1	.
20	.	1	1	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	1	1
21	1	.	1	1	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	1

(c) $\boldsymbol{X^2}$

Figure 3.1: Quorum matrices after the initial, first and second iterations respectively. The zero entries have been replaced with dots for clarity.

tion, initializes the Quorum array (line 32–37) and calls procedure PARTITION (line 38) to modify the array so that the final array corresponds to the first row of a quorum matrix.

In this algorithm, we partition a quorum at each iteration into 3 partitions and discard the middle partition. In fact, we may partition a quorum into p runs, where p is a prime number $p \geq 3$, and remove the $\lfloor p/2 \rfloor$ interleaving partitions alternately. This is the generalized version of QGEN. However we show in Section 3.3 that the smallest quorum is generated with $p = 3$.

3.2 Proof of correctness

The correctness of the above construction is established by the following theorem.

Theorem 3.1 *Quorums generated by QGEN form a symmetric coterie.*

Proof: We must show that the five properties of a symmetric coterie exist in the quorums generated by QGEN. The *Non-emptiness* property is trivially true. At each iteration, all quorums are of the same size, therefore the *Minimality* property holds. Additionally, they are reduced by the same number of nodes, therefore the *Equal Size* property also holds. The number of quorums that each node is included in determines the *Equal Effort* property. Due to the symmetry of the reduction process, all nodes are excluded from the same number of quorums at each iteration. Thus the *Equal Effort* property holds. We next show the *Mutual Intersection* property holds by induction on the number of iterations $0 \leq \ell < \lfloor \log_3 n \rfloor$. At iteration $\ell = 0$, each quorum contains $Q_0^i = \{[i]_n, [i+1]_n, \ldots, [i+k_0-1]_n\}$ where $[j]_n$ denotes j modulo n. For any $0 \leq i < j \leq n-1$, $Q_0^i \cap Q_0^j = \{[j]_n, [j+1]_n, \ldots, [i+k_0-1]_n\} \cup \{[i]_n, [i+1]_n, \ldots, [j+k_0-1]_n\} \neq \phi$. Hence iteration $\ell = 0$ holds.

We claim that if quorums at the ℓth iteration, $1 \leq \ell \leq \lfloor \log_3 n \rfloor$, mutually intersect, then so do the quorums at the $(\ell+1)$th iteration. Consider the ways in which two runs R_i, R_j of quorums Q_i, Q_j respectively intersect. At the next iteration, both runs are partitioned into three segments A, B and C as shown in Figure 3.3. Figure 3.3a shows the three ways in which R_j still intersect with R_i as long as it is still within the boundaries of R_i. When R_j is just out of the right-hand boundary of R_i (Figure 3.3b), the fact that the middle segment that is discarded is always one element less that the remaining segments ensures that R_j intersect with the next consecutive run of R_i.

```
boolean $Quorum$ : array[$0 \ldots n-1$];

function ADJUST($r$);
begin
    case ($r$ mod 3) of
        0 : return $r + 2$;
        1 : return $r + 1$;
        2 : return $r$;
    endcase;
end;

procedure PARTITION($s, r$);
begin
    $size \leftarrow r - s + 1$;
    if ($size > 7$) then
        $size \leftarrow$ ADJUST($size$);
        $x \leftarrow (size + 1)/3$;
        for $i = x$ to $(2x - 2)$ do
            $Quorum[i] \leftarrow 0$ ;
        endfor;
        PARTITION($s, s + x - 1$);
        PARTITION($s + 2x - 1, r$) ;
    else
        case ($size$) of
        4, 5 : $Quorum[s + 2] \leftarrow 0$;
        6, 7 : $Quorum[s + 3] \leftarrow 0$;
               $Quorum[s + 4] \leftarrow 0$;
        endcase ;
    endif;
end;

function QGEN($n$);
begin
    $k_0 \leftarrow$ ADJUST($\lfloor n/2 \rfloor + 1$);
    for $i = 0$ to $n - 1$ do
        if ($i < k_0$) then
            $Quorum[i] \leftarrow 1$ ;
        else
            $Quorum[i] \leftarrow 0$ ;
    endfor;
    PARTITION($0, k_0 - 1$) ;
    return $Quorum$;
end;
```

Figure 3.2: Algorithm QGEN.

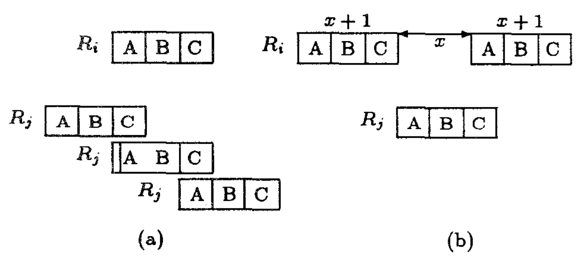

Figure 3.3: Runs of different quorums preserve intersection after partitioning.

Again R_j is within the boundary of this run. Hence intersection preserves through the next iteration.

However there are four special terminal cases when the pre-final run size is less than 8, i.e. 4, 5, 6 and 7. We prove that subdivision in these cases also preserves the intersection property. Again we shall show that Q_j intersects with Q_i for all $j \neq i$ after the final partition. In particular, we show how two runs of Q_j's still intersect with the two runs of Q_i. Figure 3.4 illustrates the final partitions when the pre-final run sizes are 4, 5, 6 and 7. Consider a run size of 4. We have divided the set of Q_j's into two: those whose two runs still intersect with Q_i, i.e. Q_j for $j \in \{[i+k]_n \mid 1 \leq k \leq 10\}$, and those whose runs do not. For the latter case, we may consider the two runs and the middle partition as a single large run, in which case the size is more than 7. By the induction above, the intersection property holds. In the figure, those nodes of Q_j that do not intersect with the two runs of Q_i are left out. Those that do are shown in bold. It is obvious by inspection that Q_i and Q_j still intersect after the final partitioning for all $j \neq i$. The same argument holds for cases 5, 6 and 7. Hence we may conclude that the algorithm generates a symmetric coterie. ∎

3.3 Analysis of algorithm

3.3.1 Size complexity

At the ℓth iteration, the size of a quorum, $k_\ell = 2^\ell \cdot x_\ell$ where $x_\ell = (x_{\ell-1} + 1)/3$, $x_0 = k_0$. Expanding the recurrence relation gives $k_\ell = (2/3)^\ell \left(2k_0 + 3^\ell - 1\right)/2$. Since the subdivision process is iterated $\lfloor \log_3 k_0 \rfloor$ times and $k_0 \approx n/2$,

$$\begin{aligned}
k_{\lfloor \log_3 k_0 \rfloor} &= (2/3)^{\log_3 k_0} \left(2k_0 + 3^{\log_3 k_0} - 1\right)/2 \\
&= k_0^{\log_3 (2/3)} \left(3k_0 - 1\right)/2 \\
&= 1/2 \left(3k_0^{\log_3 2} - k_0^{\log_3 (2/3)}\right) \\
&\approx 1/2 \left(3 (n/2)^{0.63} - (n/2)^{-0.37}\right) \\
&\approx 0.97n^{0.63} - \left(0.65/n^{0.37}\right) \\
&< 0.97n^{0.63} \qquad (3.2)
\end{aligned}$$

Therefore the final quorum size is $O(n^{0.63})$. In any generalized version of this algorithm where $p \geq 3$ is the unit of partitioning, it can be similarly shown that the optimal quorum size generated by the algorithm is $O(n^{\log_p \lfloor p/2 \rfloor})$. Since $\log_p \lfloor p/2 \rfloor$ is minimum at $p = 3$, our algorithm achieves the smallest quorum size of $O(n^{0.63})$.

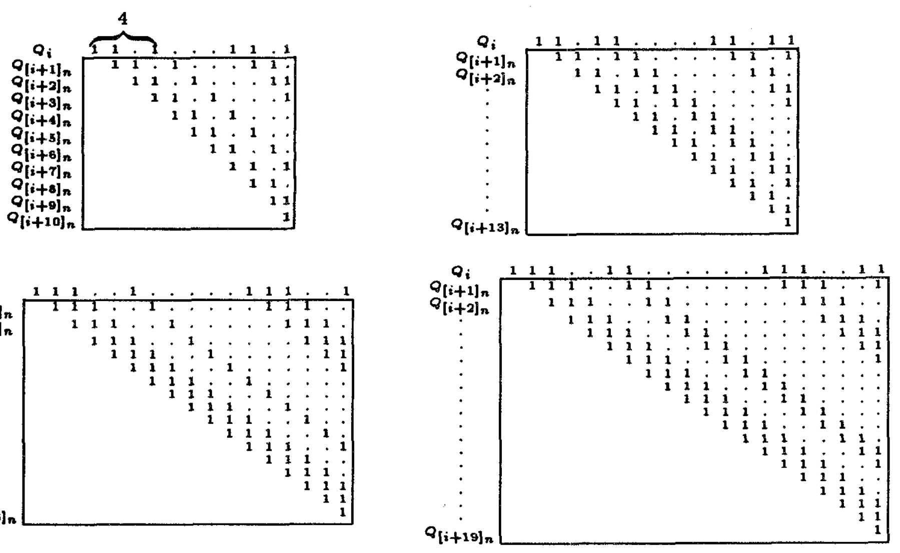
Figure 3.4: Final partitions for the special terminal cases when the pre-final run sizes are 4, 5, 6 or 7.

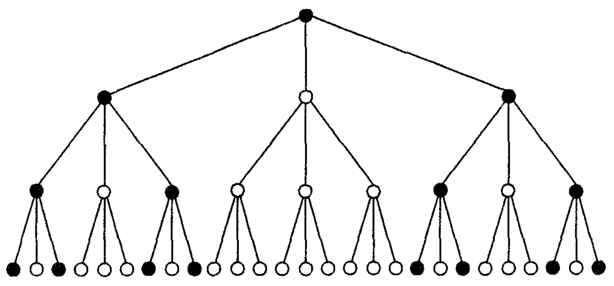
Figure 3.5: Ternary tree as the search space of QGEN.

3.3.2 Time complexity

The algorithm QGEN constructs a ternary tree during computation, as shown in Figure 3.5, where each node represents a run. Each node is one of the three subruns of its parent node (except the root), and in turn, produces three subruns. The middle run is always discarded, as represented by white circles in the figure. The length of the middle run is always one less than the length of the other two sibling runs. The black circles are runs that the algorithm partitions further. As the algorithm is recursively called with each node as the parameter, its running time is proportional to the size of the tree (less the discarded nodes). Since the height of the tree is $\lfloor \log_3 k_0 \rfloor$, the time complexity of the version presented in Figure 3.2 is approximately $2^{\lfloor \log_3 k_0 \rfloor} = n^{\log_3 2}$.

However the symmetry of QGEN permits the $O(\log n)$ version shown in Figure 3.6 and by the arrows in Figure 3.5. In the new version, a run is denoted by an interval $\langle s, r \rangle$ which indicates the consecutive numbers $s, s+1, \ldots, r-1, r$ representing the run. A quorum is a set of such runs, as declared at line 4. For instance the quorum in Figure 3.1 is given by $\{\langle 0,1\rangle, \langle 3,4\rangle, \langle 9,10\rangle, \langle 12,13\rangle\}$. A new variable is used: *TerminalCase*, which indicates those final run sizes: 4,5,6 and 7, that requires special treatment. At line 19, procedure PARTITION only traverses the leftmost branch of the ternary tree and computes the final leftmost run. It stores the run at lines 24, 26 or 28. This requires $O(\log n)$ time, proportional to the height of the tree. In the older version (see Figure 3.2), procedure PARTITION is called upon both the left and right branches, discarding the middle. The final leftmost run, now contained in *Quorum*, is then used to re-construct the entire quorum in $O(\log n)$ time, as shown in lines 36–49 after procedure PARTITION returns in function QGEN. Therefore the time complexity of the implementation is $O(\log n)$.

3.4 Upper bound for common nodes

The optimum quorum size of $O(n^{0.5})$ (see Section 2) assumes that λ, the number of common nodes between any two quorums, is 1. Algorithm QGEN is an approximation algorithm where the constraint that λ be constant is relaxed. We shall derive a bound for

```
integer $FinalStartRun$;                                  {smallest run size}
integer $TerminalCase$;                                   {final run size cases}
integer $RunSize$;                                        {final run size}
integer $StartRun$ : array$[1 \ldots \lfloor \log_3 k_0 \rfloor]$;             {starting run indicator}
boolean $Quorum$ : set of $run$;

function ADJUST($r$);                                     {adjust run size so}
begin                                                     {that partitioning is possible}
    case ($r$ mod 3) of
        0 : return $r + 2$;
        1 : return $r + 1$;
        2 : return $r$;
    endcase;
end;

procedure PARTITION($s, r, k$);                           {partition one run only}
begin
    $size \leftarrow r - s + 1$;
    if ($size > 7$) then
        $size \leftarrow$ ADJUST($size$);
        $x \leftarrow (size + 1)/3$;
        $StartRun[k] \leftarrow 2x - 1$;
        PARTITION($s, s + x - 1, k - 1$);
    else                                                  {special terminal cases for}
        $TerminalCase \leftarrow size$;                   {small run size}
        case ($size$) of
            4, 5 : $Quorum \leftarrow \{\langle 0, 1 \rangle\}$;
            6, 7 : $Quorum \leftarrow \{\langle 0, 1, 2 \rangle\}$;
            else: $Quorum \leftarrow \{\langle 0, size - 1 \rangle\}$;
                $RunSize \leftarrow size$;
        endcase;
    endif;
end;

function QGEN();
begin
    $k_0 \leftarrow$ ADJUST($\lfloor n/2 \rfloor + 1$);                {initial quorum size}
    PARTITION($0, k_0 - 1, \lfloor \log_3 k_0 \rfloor$);
    case ($TerminalCase$) of
        4:   $Quorum \leftarrow Quorum \cup \{\langle 3, 3 \rangle\}$;
        5:   $Quorum \leftarrow Quorum \cup \{\langle 3, 4 \rangle\}$;
        6:   $Quorum \leftarrow Quorum \cup \{\langle 5, 5 \rangle\}$;
        7:   $Quorum \leftarrow Quorum \cup \{\langle 5, 6 \rangle\}$;
        else: $Quorum \leftarrow Quorum \cup \{\langle StartRun[1], RunSize - 1 + StartRun[1] \rangle\}$;
    endcase;
    for $k = 2$ to $\lfloor \log_3 k_0 \rfloor$ do
        for each $\langle a, b \rangle \in Quorum$ do                 {reconstruct quorum}
            $Quorum \leftarrow Quorum \cup \{\langle a + StartRun[k], b + StartRun[k] \rangle\}$;
        endfor;
    endfor;
    return $Quorum$;
end;
```

Figure 3.6: An $O(n^{0.63} \log_3 n)$ implementation of QGEN.

the maximum value of λ in quorums generated by the algorithm.

Theorem 3.2 *The number of common nodes between any two quorums produced by QGEN is no more than a third of the quorum size.*

Proof: Q^i, the quorum of node i, has the maximum number of nodes in common with Q^{i-1} and Q^{i+1}. We shall consider Q^i and Q^{i+1} only since the other case is identical. If the size of each run in Q^i is x, then Q^i and Q^{i+1} intersect in exactly $x-1$ nodes at each run. Therefore the maximum number of common nodes is the difference between the size of a quorum and the number of runs, i.e. $\approx 3/2k_0^{\log_3 2} - 2^{\log_3 k_0} = 3/2(n/2)^{\log_3 2} - (n/2)^{\log_3 2} = 1/2(n/2)^{\log_3 2} \approx 0.32n^{\log_3 2}$. Since the quorum size is approximately $n^{\log_3 2}$ as given by Equation 3.2 and since we have ignored the $0.65/n^{0.37}$ term, the maximum number of common nodes is no more than a third of the quorum size. ∎

3.5 Global knowledge and storage cost

Sites must maintain information about the membership of their quorums. This requires at most $\log_2 n$ bits for the quorums generated by QGEN where the ith bit is 1 if and only if site i is in the quorum. Thus the storage cost is low. Also, the symmetry of the quorums generated by QGEN permits the storage of global coterie information at no extra cost. To know the quorums of other sites, each site need only know the site numbers of these other sites. The quorums can then be computed easily and need not be stored explicitly. For two sites i and j, if $Q^i = \{i, i+1, \ldots, i+m\}$, then $Q^j = \{[i+k]_n, [i+1+k]_n, \ldots, [i+m+k]_n\}$ if $i < j$ and $Q^j = \{[i-k]_n, [i+1-k]_n, \ldots, [i+m-k]_n\}$ if $i > j$ where $k = |i-j|$.

4 Conclusions

Achieving distributed mutual exclusion is a fundamental problem in distributed systems. All process interactions in distributed systems involve coordination and synchronization to some degree. Many protocols and algorithms have been proposed in the past to achieve mutual exclusion in distributed system. However, no algorithms exist to date that construct symmetric and optimum quorums. In this paper, we have defined a generalized form of the problem of quorum construction, SCC. Our contribution is the proposal of an efficient algorithm QGEN to generate symmetric and near-optimum coteries in the form of generic coterie templates. Coterie template not only allows multiple quorums to be instantiated, it also permits these quorums to reflect important network characteristics.

References

[1] D. Agrawal, A. Abbadi. An Efficient and Fault-Tolerant Solution for Distributed Mutual Exclusion. *ACM Transactions on Computer Systems*, Vol. 9, No. 1, pp. 1–20, Feb. 1991.

[2] A. Albert, R. Sandler. *An Introduction to Finite Projective Planes*, Holt, Rinehart & Winston, New York, 1968.

[3] D. Barbara, H. Garcia-Molina. Mutual Exclusion in Partitioned Distributed Systems. *Distributed Computing*, No. 1, pp. 119–132, 1986.

[4] O. Carvalho, G. Roucairol. On Mutual Exclusion in Computer Networks. *Communications of the ACM*, Vol. 26, No. 2, pp. 146–147, 1983.

[5] C. J. Colbourn, P. C. van Oorschot. Applications of Combinatorial Designs in Computer Science. *ACM Computing Surveys*, Vol. 21, No. 2, pp. 223–50, 1989.

[6] Y. I. Chang, M. Singhal, M. T. Liu. A Fault Tolerant Algorithm for Distributed Mutual Exclusion. *Proceedings of the 9th Symposium on Reliable Distributed Systems*, pp. 146–154, Oct. 1990.

[7] H. Garcia-Molina, K. Barbara. How to Assign Votes in a Distributed System. *Journal of the ACM*, Vol. 32, No. 4, pp. 841–860, Oct. 1985.

[8] M. Maekawa. A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems. *ACM Transactions on Computer Systems*, Vol. 3, No. 2, pp. 145–159, 1985.

[9] M. Raynal. A Simple Taxonomy for Distributed Mutual Exclusion Algorithms. Technical Report 560 INRIA, 78153 Le Chesnay Cedex, France, 1990.

[10] G. Ricart, A. Agrawala. An Optimal Algorithm for Mutual Exclusion in Computer Networks. *Communications of the ACM*, Vol. 24, No. 1, pp. 9–17, Jan. 1981.

[11] H. J. Ryser. *Combinatorial Mathematics*, Carus Mathematical Monographs, No. 14, Mathematical Association of America, Washington, D.C., 1963.

[12] B. A. Sanders. The Information Structure of Distributed Mutual Exclusions Algorithms. *ACM Transactions on Computer Systems*, Vol. 5, No. 3, pp. 284–299, Aug. 1987.

[13] M. Singhal. A Heuristically-Aided Algorithm for Mutual Exclusion in Distributed Systems. *IEEE Transactions on Computers*, Vol. 38, No. 5, pp. 651–662, May 1989.