CHAPTER 2

VACANT CODES GROUPING

In this chapter, a fast OVSF single code assignment scheme is proposed and extended to multi codes which aims to reduce number of codes searched with optimal/suboptimal code blocking. The code assignment scheme aims to use those vacant codes whose parents are already blocked. This leads to occurrence of more vacant codes in groups, which ultimately leads to less code blocking for higher rate calls. The number of codes searched increases linearly in our scheme compared to most of other novel proposed single code methods like crowded first assignment, where it increases exponentially with increase in user rates. Also, the calculation of vacant codes at one layer will be sufficient to identify the vacant code adjacency for all the layers which reduces complexity. Simulation results are presented to verify the superiority of the scheme. The single code scheme described in section 2.1 reduces code blocking with additional benefit of reduction in number of codes searched prior to assignment of new call. For a particular higher

![Diagram of six layer OVSF code tree with maximum capacity of 32R]

**Figure 2.1: A six layer OVSF code tree with maximum capacity of 32R**

Enter input parameters like arrival rate, call duration, code tree size \( L \) and number of user classes (say \( L' \)).

**Generate a new call of rate \( 2^{l_1}R, i \in [1, L'] \) in layer \( l \).**

- **Existing used capacity of code tree \( +2^{l_1}R < 128R \)?**
  - **Yes**
    - **Reject Call**
  - **No**
    - **At least one vacant code available in layer \( l \)?**
      - **Yes**
        - If there are \( N_i \) vacant codes. List all the vacant codes, \( C_{l,n_i}, i \in [1, L'] \), \( n_i \in [1, 2^{l_i-1}] \). Let \( z \) denotes subset of vacant codes \( N_i \) whose parents need to be checked. Initialize adjacency of these \( z \) vacant codes to zero.
      - **Go to next vacant code**
      - **Go to parent of vacant code \( C_{l,n_i} \) (Initially \( i = 1 \)) in layer \( l \) and check its status. If free?**
        - **No**
          - **Adjacency is zero.**
        - **Yes**
          - Check the parent(s) of \( C_{l,n_i} \) in layer \( l', l \leq l' \leq L \), such that \( C_{l_i,n_i/2^{l-i}} \) is blocked. The identifier \( [x] \) represents smallest integer greater than or equal to \( x \). Increase adjacency by \( 2^{l-i-1} - 1 \). Store adjacency of the vacant code and skip code searches of next vacant codes on right side equal to amount of adjacency. Increment \( i \) by 1.
          - **No**
            - \( i = 9 \)
          - **Yes**
            - Assign new call to code \( C_{l,n_i}, i \in [1, z], \) with minimum adjacency.

**Figure 2.2**: Flowchart of proposed AVC scheme
layer code, all the children codes are either vacant or occupied. This provides the occurrence of vacant codes in groups, which can be easily located and the number of code searches are reduced. This further reduces decision time (or call establishment time), complexity and cost. Although, the ADA scheme also provides the reduction in code searches, but the scheme discussed in this chapter provides the additional benefit of reduction in code blocking [34].

A multi code scheme provides lesser code blocking with higher call establishment delay. In this chapter, the proposed multi code assignment schemes in section 2.2 aims to reduce number of codes searched required before assignment of call and number of code searches required by scheme in section 2.1 are reduced. The scheme uses the status of higher layer codes searched in Orthogonal Variable Spreading Factor (OVSF) code tree to identify lower layer codes status. Two schemes variants of our scheme namely scattered and grouped multi code are used for comparison with other schemes. Simulation results are used to verify the superiority of proposed schemes.

The remainder of the chapter is organized as follows. Section 2.1 to 2.2 gives the description of proposed scheme along with flowchart and examples. Simulation results are given in section 2.3. The chapter is concluded in section 2.4.

2.1 SINGLE CODE ASSIGNMENT: ADJACENT VACANT CODES (AVC)

A fast OVSF code assignment scheme is proposed which aims to reduce number of codes searched with optimal/suboptimal code blocking. The code assignment scheme aims to use those vacant codes whose parents are not free (already blocked). This leads to occurrence of vacant codes in groups (adjacent vacant codes) which ultimately leads to less code blocking for higher rate calls. The number of codes searched increases linearly in our scheme compared to most of other proposed (single code) most popular methods like DCA[25] and CFA[23] (where it increases exponentially with increase in user rates). Also, the calculation of vacant codes at one

<table>
<thead>
<tr>
<th>Layer (l)</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>l</th>
<th>...</th>
<th>L-1</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum adjacency</td>
<td>$2^{l-1}-1$</td>
<td>$2^{l-2}-1$</td>
<td>...</td>
<td>$2^{l-1}-1$</td>
<td>...</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 2.2: Relationship between adjacency of a vacant code in layer \( l \) and its parents in layer \( l', l' > l \).

<table>
<thead>
<tr>
<th>Layer Number</th>
<th>Vacant Codes</th>
<th>Adjacency</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l )</td>
<td>( C_{l,n_l^i} ) to ( C_{l,n_l^i+A_{l,n_l^i}} )</td>
<td>( A_{l,n_l^i} )</td>
</tr>
<tr>
<td>( l+1 )</td>
<td>( C_{l+1,\left\lfloor n_l^i/2 \right\rfloor} ) to ( C_{l+1,\left\lfloor n_l^i/2 \right\rfloor + A_{l+1,n_l^i}/2} )</td>
<td>( A_{l+1,n_l^i}/2^m )</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>( l + \log_2 A_{l,n_l^i} + 1 )</td>
<td>( C_{l+m,\left\lfloor n_l^i/2 \right\rfloor} ) to ( C_{l+m,\left\lfloor n_l^i/2 \right\rfloor + A_{l,n_l^i}/2^m} )</td>
<td>( A_{l,n_l^i}/2^m )</td>
</tr>
</tbody>
</table>

where \( m = \log_2 A_{l,n_l^i} + 1 \)

layer will be sufficient to identify the vacant code adjacency for all the layers which reduces complexity and call establishment delay of future calls.

Consider an \( L \) layer CDMA based OVSF code tree. If a new call with rate \( 2^{l-1} R \), where \( 1 \leq l \leq L \) arrives, the proposed scheme (called as adjacent vacant code (AVC) scheme) lists all the vacant codes in layer \( l \). Let layer \( l \) has \( N_l \) vacant codes. The \( i \)th vacant code in the layer \( l \) is represented by \( C_{l,n_l^i} \) and \( 1 \leq i \leq N_l \). For a code \( C_{l,n_l^i} \), define adjacency \( (A_{l,n_l^i}) \) as the number of vacant codes adjacent to \( C_{l,n_l^i} \) whose parents in all layers are same as the parents of \( C_{l,n_l^i} \). The adjacency of all these vacant codes is the same and it is sufficient to check the status of one of these codes. The scheme checks the status of only left code appearing in code tree from this group and for remaining codes adjacency is same. For a vacant code \( C_{l,n_l^i} \), whose left hand code is busy/blocked or its left hand vacant code has at least one same vacant parent as the parent of \( C_{l,n_l^i} \), the AVC scheme checks the parents of code in layers \( l+1 \) to \( L \) till the blocked parent in layer \( l' \) (where \( l' > l \) and \( l' = \min[L+1], L \mid C_{l',n_l^i}/2^{l'-1} \) is blocked) appears. The identifier \( \left\lfloor x \right\rfloor \) represents the smallest integer less than or equal to \( x \). The adjacency of code \( C_{l,n_l^i} \) becomes \( (2^{l'-l-1}) - 1 \). The vacant codes \( C_{l,n_l^i} \) to \( C_{l,n_l^i+A_{l,n_l^i}} \) need not to be checked as all these have same value of adjacency. The values of \( A_{l,n_l^i} \) for a code \( C_{l,n_l^i} \) is equal to \( 2^{l'-l-1} - 1 \).
Table 2.3: Finding number of code searches in Figure 2.1 at the arrival of 2R rate call

<table>
<thead>
<tr>
<th>Code # in layer 2</th>
<th>Whether status checked?</th>
<th>Number of skips</th>
<th>Number of codes searched</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_{2,1}</td>
<td>Y</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>C_{2,2}</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>C_{2,3}</td>
<td>Y</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C_{2,4}</td>
<td>Y</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C_{2,5}</td>
<td>Y</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>C_{2,6}</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>C_{2,7}</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>C_{2,8}</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>C_{2,9}</td>
<td>Y</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C_{2,10}</td>
<td>Y</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C_{2,11}</td>
<td>Y</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>C_{2,12}</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>C_{2,13}</td>
<td>Y</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>C_{2,14}</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>C_{2,15}</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>C_{2,16}</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

NA: Not applicable, Y: Yes, N: No, the code search is skipped as one of its adjacent vacant codes is already considered for status check.

there may be more than $2^{n-l-1} - 1$ consecutive vacant codes. The maximum possible value of $A_{i,n_j}$ depends upon layer number $l$. The relationship between layer number $l$, vacant codes location and maximum adjacency is given in Table 2.1. The lesser is the value of adjacency $A_{i,n_j}$ for a code $C_{i,n_j}$, lesser is the number of adjacent vacant codes for code $C_{i,n_j}$. For a specified adjacency $A_{i,n_j}$ (say), the codes from layer $l$ to $(l+\log_2[A_{i,n_j}+1])$ are vacant and can be used for new calls as given in Table 2.2.

Hence, if a vacant code $C_{i,n_j}$ has higher value of adjacency, the bigger portion of code tree around $C_{i,n_j}$ is free, which leads to more higher rate parents free to handle new call(s). The scheme selects the vacant code whose adjacency is least. If a tie occurs for two or more vacant
Table 2.4: Total number of codes searched in layer ‘l’ at the arrival of $2^k R$ call

<table>
<thead>
<tr>
<th>Vacant code</th>
<th>Adjacency ($A_{l,n_l^k}$)</th>
<th>Number of skips</th>
<th>Next location to be searched</th>
<th>Number of codes searched</th>
<th>Total Number of code searched (say $NA_{l,n_l^k}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{l,n_l^k}$ or $C_{l,n_l^r}$</td>
<td>$2^{l_x-1} - 1$</td>
<td>$2^{l_x-1} - 1$</td>
<td>$n_l^k + 2^{l_x-1} + t_x$</td>
<td>$l_x + 1$</td>
<td>$NA_{l,n_l^k} = n_l^k + l_x$</td>
</tr>
<tr>
<td>$C_{l,n_l^r}$</td>
<td>$2^{l_x-1} - 1$</td>
<td>$2^{l_x-1} - 1$</td>
<td>$n_l^r + 2^{l_x-1} + t_x$</td>
<td>$l_x + 1 + t_x$</td>
<td>$NA_{l,n_l^r} = NA_{l,n_l^k} + l_x + 1 + t_x$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$C_{l,n_l^r}$</td>
<td>$2^{l_x-1} - 1$</td>
<td>$2^{l_x-1} - 1$</td>
<td>$n_l^r + 2^{l_x-1} + t_x$</td>
<td>$l_x + 1 + t_x$</td>
<td>$NA_{l,n_l^r} = NA_{l,n_l^r-1} + l_x + 1 + t_x$</td>
</tr>
</tbody>
</table>

codes having same adjacency, the optimum code selection can be done using following two ways.

- Pick any code randomly.
- Use a second level criterion like elapsed time, crowded first capacity, crowded first space etc.

It assigns new call to the most crowded portion of the tree. One of the significant benefits of the AVC scheme is to reduce the vacant code searches at the arrival of new call which makes the code assignment faster than single code schemes like; crowded first code and crowded first space schemes. Adjacency leads to less code searches for a vacant code $C_{l,n_l^k}$ in two ways.
Figure 2.3: Illustration of code searches in AVC and CFA schemes.
(a) Let adjacency is \( k \) for vacant code \( C_{l,n} \), then for all codes between \( C_{l,n} \) and \( C_{l,n+k} \) has same adjacency \( k \) and only one code need to be searched. The next vacant code search will start from \( C_{l,n+k+1} \).

(b) For finding the adjacency of a code \( C_{l,n} \), we go to its parent codes and check whether they are free or not. If we go up \( l'' \), \( 1 \leq l'' \leq (L-1) \) steps above then number of code searches will be \( l'' = (L-I) \), (one for each layer) and adjacency or codes vacant preceding \( C_{l,n} \) checked is \( 2^{l''} - 1 \). Checking of a parent code leads to the search of two children status on code tree and while doing so, code searches will increase linearly (instead of exponentially). For a new \( 2R \) call arrival, the number of codes searched in the AVC scheme is illustrated in Table 2.3 (corresponding to Figure 2.1). As discussed earlier, all the vacant codes need not be checked. For example, the 1st code \( C_{2,1} \) is vacant and status of its parents is checked till the 1st blocked parent appears. The 1st blocked parent is \( C_{4,1} \). The parent checking procedure stops and the adjacency of \( C_{2,1} \) comes out to be 1. Due to the adjacency value 1, code \( C_{2,2} \) is not the candidate for status check, as it is already considered in parent check procedure of \( C_{2,1} \). The next vacant code whose parent should be checked is \( C_{2,5} \). The procedure is repeated for the complete tree as given in Table 2.3. In Table 2.3, column 2 represents whether status of code in layer 2 is checked or not, column 3 represents number of skips before next vacant code to be searched and column 4 represents number of codes searched for finding adjacency. The flow chart of algorithm is given in Figure 2.2. The general outline of the algorithm is given as follows.

1. Input parameters like arrival rate, call duration, code tree size \( L \) and number of user classes (say \( L' \)) are entered.

2. Generate new call with rate \( 2^{l-1}R \).

3. For every new call, capacity check is performed. If capacity is available, go to step 4 otherwise the call is rejected.

4. (a) If at least one vacant code is available, list all the vacant codes (say \( N_I \)) and the subset \( z \) of these \( N_I \) vacant codes whose parent(s) status must be checked.

(b) Go to the first vacant code from the left in code tree.

(c) Find the adjacency and skip the number of vacant codes equal to adjacency.

(d) If all \( z \) vacant codes (and their parents) are checked, find the code with the least value of
adjacency and assign call to this optimum code. Otherwise, go to the next vacant code and return to step 4(c).

(e) Go to step 2.

Assuming layer \( l \) has \( N_f, N_f < 2^{L-l} \) vacant codes and each vacant code in layer \( l \) can be denoted by \( C_{t, n_f} \), \( 1 \leq n_f \leq 2^{L-l} \) and \( 1 \leq i \leq N_f \), the number of codes searched in the AVC scheme can be compared with the codes searched in CFA scheme [23].

<table>
<thead>
<tr>
<th>Status with Adjacency</th>
<th>Codes #</th>
<th>Code searches in AVC scheme</th>
<th>Number of skips in AVC scheme</th>
<th>Code search in CFA scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Busy/Blocked Adjacency:0</td>
<td>( C_{2,1}, C_{2,5}, C_{2,7}, C_{2,9}, C_{2,11}, C_{2,13}, C_{2,14}, C_{2,16}, C_{2,21}, C_{2,22}, C_{2,26}, C_{2,28}, C_{2,30} ) to ( C_{2,61} )</td>
<td>1 for each code = 31</td>
<td>0</td>
<td>1 each = 31</td>
</tr>
<tr>
<td>Vacant Adjacency:0</td>
<td>( C_{2,2}, C_{2,6}, C_{2,8}, C_{2,10}, C_{2,12} )</td>
<td>2 each = 10</td>
<td>0</td>
<td>5 each = 5x5 = 25</td>
</tr>
<tr>
<td>Vacant Adjacency:1</td>
<td>( C_{2,25}, C_{2,26}, C_{2,27}, C_{2,28}, C_{2,30} )</td>
<td>3 (for ( C_{2,25} ) only)</td>
<td>1</td>
<td>10 each = 20</td>
</tr>
<tr>
<td>Vacant Adjacency:3</td>
<td>( C_{2,41} ) to ( C_{2,40} )</td>
<td>4 (for ( C_{2,41} ) only)</td>
<td>3</td>
<td>19 each = 76</td>
</tr>
<tr>
<td></td>
<td>( C_{2,31} ) to ( C_{2,40} )</td>
<td>4 (for ( C_{2,31} ) only)</td>
<td>3</td>
<td>19 each = 76</td>
</tr>
<tr>
<td></td>
<td>( C_{2,41} ) to ( C_{2,44} )</td>
<td>4 (for ( C_{2,41} ) only)</td>
<td>3</td>
<td>19 each = 76</td>
</tr>
<tr>
<td></td>
<td>( C_{2,49} ), ( C_{2,52} )</td>
<td>4 (for ( C_{2,49} ) only)</td>
<td>3</td>
<td>19 each = 76</td>
</tr>
</tbody>
</table>

Total number of searches = 72

Table 2.5: Comparison of number of code searches in AVC and CFA scheme for tree in Figure 2.2

2.1.1 CODE SEARCHES FOR AVC SCHEME

For a new call arrival with rate \( z^{L-1} R \), let \( C_{t, n_f} \) be the first vacant code. The next vacant code whose status need to be checked is at the location \( n_f + 2^{k-1} \) \((k_1 \) is the steps/layers above \( l \) where first blocked parent code is found for a vacant code \( C_{t, n_f} \). Let the \( k^{th} \) vacant code whose status has
to be checked (which may be different from $C_{i,n_{j}^{k}}$) is denoted by $C_{i,n_{j}^{k+1}}, x_{k} \in [1,N_{f}], x_{k} > k$.

Consider $n_{i}^{1}$ and $l_{1}$ is $n_{i}^{x_{1}}$ and $l_{x_{1}}$ respectively. If $t_{x_{1}}$ denotes number of busy or blocked codes checked, next vacant code $C_{i,n_{j}^{x_{1}}}$ searched will be at location $n_{i}^{x_{1}} + 2^{l_{x_{1}}-1} + t_{x_{1}}$. This result can be generalized to find location of $k^{th}$ vacant code $C_{i,n_{j}^{k}}$ and is given by $n_{i}^{x_{k-1}} + 2^{l_{x_{k-1}}-1} + t_{x_{k-1}}$. The

![Tree Diagram](image)

**Figure 2.4:** Illustration of parent and children code adjacency.

procedure is repeated $z$ times, where $z$ denotes subset of $N_{f}$ vacant codes which need to be checked for adjacency and the last vacant code checked is $C_{i,n_{j}^{z}}$, as given in Table 2.4.

For the first code $C_{i,n_{j}^{1}}$ or $C_{i,n_{j}^{x_{1}}}$, number of code searches is $(l_{x_{1}} + 1)$. For $k^{th}$ vacant code $C_{i,n_{j}^{k}}$, number of code searches is $l_{x_{k}} + 1 + t_{x_{k-1}}$. The procedure is repeated till last code $C_{i,n_{j}^{z}}$. Number of code searched for a vacant code and total number of code searched till that vacant code is also shown in Table 2.4.

Total number of code searched in AVC scheme is given below:

$$N_{AVC} = n_{i}^{x_{1}} - 1 + l_{x_{1}} + l_{x_{2}} + l_{x_{2}} + t_{x_{1}} + t_{x_{1}} + t_{x_{2}} + + l_{x_{z}} + l_{x_{z}} + t_{x_{z-1}} + t_{x_{z}}$$  \( 1 \)

$$N_{AVC} = n_{i}^{x_{1}} - 1 + \sum_{j=1}^{z}(l_{x_{j}} + 1 + t_{x_{j}})$$  \( 2.2 \)

$$N_{AVC} = n_{i}^{x_{1}} - 1 + z + \sum_{i=1}^{z}(l_{x_{i}} + t_{x_{i}})$$  \( 2.3 \)

30
2.1.2 CODE SEARCHES FOR CFA SCHEME

Considering the definition of \( C_{\text{t,n}^I} \), the number of codes searched for \( C_{\text{t,n}^I} \) using CFA [23] scheme is \( NC_{\text{t,n}^I} = l_{x_1} + 2^{l+l_h} - 1 \). For the \( i \)th vacant code, number of code searched is

\[ NC_{\text{t,n}^I} = l_{x_i} + 2^{l+l_h} - 1 \].

For all the \( N_I \) vacant codes, where \( N_I \leq 2^{L-l} \), the total number of codes searched becomes

\[ NC_{\text{CFA}} = 2^{L-l} + l_{x_1} + (2^{l+l_h} - 1) + l_{x_2} + (2^{l+l_h} - 1) + \ldots + l_{x_{N_I}} + (2^{l+l_h} - 1) \]  \hspace{1cm} (2.4)

\[ NC_{\text{CFA}} = 2^{L-I} + \sum_{i=1}^{N_I} (l_{x_i} + 2^{l+l_h} - 1) \]  \hspace{1cm} (2.5)

\[ NC_{\text{CFA}} = 2^{L-l} - N_I + \sum_{i=1}^{N_I} (l_{x_i} + 2^{l+l_h} - 1) \]  \hspace{1cm} (2.6)

Our proposed scheme always needs less number of code searches as compare to CFA [23], the comparison is carried and is given in Appendix A. In comparison to CFA, where code search is increasing exponentially with every parent search, the AVC scheme leads to a linear code searches which decreases the number of code searched as we go in upper layers from \( l \) to \( L - l \). For illustration, consider example in Figure 2.3. Consider a new call arrival with the rate \( 2R \). CFA scheme searches most crowded portion of the tree. For that it searches the parent codes and their children. It selects a vacant code whose parent/parents have maximum number of busy codes. The comparison of CFA and AVC scheme is illustrated in Table 2.5 in context with Figure 2.3. The adjacencies of all the vacant codes in layer 2 (corresponding to \( 2R \) arrival) is listed in column 4. The higher value of adjacency (number of skips) corresponds to large free area. The code with the minimum adjacency is the optimum code and hence one of the codes with adjacency 0 is used for \( 2R \) call.

Periodic code searches can be avoided to a great extent if the current code status remains valid for next \( k \) calls. Adjacencies of codes of layer \( l \) provide information of adjacency of children codes (layers below \( l \) and parent codes (layers above \( l \). Let the vacant code \( C_{\text{t,n}^I} \) has adjacency \( P_l \). If the layer \( l - 1 \), codes \( C_{\text{t,2n}^l} \) to \( C_{\text{t,2n}^{l+1}} \) has adjacency

\[ A_l = 2P_l + 1 \]  \hspace{1cm} (2.7)

Layer \( l - 2 \), codes to has adjacency, \( A_2 = 2A_l + 1 \).
Similarly, for vacant codes in $l_1$ steps below the layer \( l \) i.e layer \( l-l_1 \) codes to has the adjacency

\[
A_{l_1} = 2A_{l_1-1} + 1 \tag{2.8}
\]

For a vacant code $C_{l,n_1}$, parent codes have adjacency given by

Layer \( l+1 \), code $C_{l+1,\lfloor n_1/2 \rfloor}$ has adjacency

\[
P_{l+1} = \left\lfloor \frac{P_l}{2} \right\rfloor \tag{2.9}
\]

Layer \( l+2 \), code $C_{l+1,\lfloor n_1/2^2 \rfloor}$ has adjacency $P_{l+1} = \left\lfloor \frac{P_l}{2^2} \right\rfloor$. In a similar way, for a vacant code in \( l_2 \) steps above layer \( l \), i.e layer \( l+l_2 \) code has adjacency

\[
P_l = \left\lfloor \frac{P_l}{2^{l_2}} \right\rfloor \tag{2.10}
\]

For \( L \)th layer adjacency will be $P_L = \left\lfloor \frac{P_l}{2^{L-l}} \right\rfloor$.

For illustration, consider the Figure 2.4. For a call of rate $4R$, vacant codes are $C_{3,1}$ to $C_{3,7}$. Adjacency for these codes is 3 from $C_{3,1}$ to $C_{3,4}$, 1 for $C_{3,5}$, $C_{3,6}$ and 0 for $C_{3,7}$ respectively. The new call will be assigned to $C_{3,7}$ as adjacency is minimum for $C_{3,7}$. As we know the adjacency of code in layer 3, apparently we know adjacency of their parent and children codes also. From Figure 2.4, it is clear that the adjacency of $C_{2,1}$ to $C_{2,8}$ has adjacency 7 i.e. $(2 \times 3 + 1)$. $C_{2,9}$ to $C_{2,12}$ has adjacency 3 i.e $(2 \times 1 + 1)$. In a similar way, we can find adjacency of parent codes of a vacant code for e.g $C_{2,1}$ has adjacency 7, its parent code $C_{3,1}$ has adjacency $\left\lfloor 7/2 \right\rfloor = 3$.

### 2.1.3 SIMULATION RESULTS

#### A. SIMULATION PARAMETERS

- Arrival rate ‘\( \lambda \)’ is Poisson distributed with mean value 0-4 calls/ units of time.
- Call duration ‘\( 1/\mu \)’ (is the service rate) is exponentially distributed with mean value of 3 units of time.
- The maximum capacity of the code tree is $128R$ (\( R \) is 7.5kbps).
- There are 5 classes of users with rates $R$, $2R$, $4R$, $8R$, $16R$.
- Simulation is done for 10000 users and result is average of 10 simulations.
Figure 2.5: Comparison of number of code searches for distribution (a) [10, 15, 25, 25, 25], (b) [25, 25, 25, 15, 10]
Figure 2.6: Comparison of number of code searches for distribution (a) [20,20,20,20,20], (b) [35,30,15,10,10].

AVC: Adjacent vacant code, CFA [22]: Crowded first assignment, FSP [27]: Fixed set partitioning, LCA [27]: Last code assignment, RFCB [33]: Recursive fewer codes blocked, DCA [75]: Dynamic code assignment, NCPH [90]: Next Code Precedence High, ADA [34]: Adaptive code assignment.
Figure 2.7: Comparison of code blocking probability for distribution (a) [10,15,25,25,25], (b) [25,25,25,15,10]

**Figure 2.8:** Comparison of Code Blocking Probability for distribution (a) [20,20,20,20,20], (b) [35,30,15,10,10]
An event driven simulation is considered where each code in a layer is a server. The number of
we consider call duration of all the servers available for layer $l$ is $2^{L-l}$ which is equal to number of
codes in layer $l$. Let $\lambda_i, i \in [1,5]$ be the load of the $i^{th}$ class users. Also, for 5 class system the
average arrival rate and average traffic is $\lambda = \sum_{i=1}^{5} \lambda_i$ and $\rho = \sum_{i=1}^{5} \lambda_i$ respectively. In this
simulation, we consider call duration of all the calls equal i.e. $1/\mu_i = 1/\mu$. Therefore, the average
traffic load is $\rho = 1/\mu \times \sum_{i=1}^{5} \lambda_i = \lambda / \mu$. If we define $[P_1, P_2, P_3, P_4, P_5]$ as probability distribution
matrix, where $P_i, i \in [1,5]$ is the capacity portion used by the $i^{th}$ class users. As mentioned earlier,
the code blocking is the major limiting factor in OVSF based networks. The average code
blocking for a 5 class system is defined as

$$P_B = \frac{\sum_{i=1}^{5} (\lambda_i P_{B_i}/\lambda)}{\lambda}$$  \hspace{1cm} (2.11)

where $P_{B_i}$ is the code blocking of $i^{th}$ class and is given by

$$P_B = \frac{\rho_i^{G_i} / G_i!}{\sum_{n=1}^{G_i} \rho_i^n / n!}$$  \hspace{1cm} (2.12)

where $\rho_i = \lambda_i / \mu_i$ is the traffic load for $i^{th}$ class.

B. RESULTS

Four traffic distributions are used for performance results given by

- [10, 15, 25, 25, 25], high rates dominating scenario.
- [25, 25, 25, 15, 10], low rates dominating scenario I.
- [20, 20, 20, 20, 20], uniform distribution scenario.
- [35, 30, 15, 10, 10], low rates dominating scenario II.

Two performance metrics namely number of code searches and blocking probability of the
proposed AVC scheme is compared with fixed set partitioning (FSP), crowded first assignment
[23](CFA), left code assignment [23](LCA), recursive fewer code blocking [33](RFBC), adaptive
code assignment [34](ADA), next code precedence high [90] (NCPH) and dynamic code
algorithm [25] (DCA) schemes discussed earlier.
(i) **CODE SEARCHES**

The comparison of the code searches is shown in Figure 2.5 and Figure 2.6 for four arrival rate distributions. If \( N_i \) denotes the number of codes searches required for \( i^{th} \) class, the total number of codes searched is given by

\[
N = \sum_{i=1}^{5} N_i
\]  

(2.13)

where \( N_i \) is the total number of codes searched for \( i^{th} \) class of user. Results shows that AVC scheme leads to less number of code searches compared to CFA, RFCB, ADA, NCPH and DCA schemes. Although, CFA and DCA are two most used single code assignment schemes, the AVC scheme requires less code searches compared to both of them which can be very beneficial especially for real time calls. On the other hand, the number of code searches required by AVC scheme are comparable to LCA and more than FSP schemes but the large code blocking probability in LCA and FSP schemes (as discussed in the next subsection) make these schemes less attractive.

(ii) **CODE BLOCKING**

The code blocking comparison is given in Figure 2.7 and Figure 2.8 for four distributions. The results show that the code blocking performance in the proposed AVC scheme is comparable to CFA, RFCB and is superior to FSP, LCA, ADA and NCPH schemes. The DCA scheme gives almost zero code blocking (hence not plotted) at the expense of higher reassignments which lead to increased complexity. Though the blocking probability is comparable to CFA and RFCB but reduction in code searches (as discussed in the previous subsection) makes the proposed scheme faster, less complex and cheaper.

From above results, it is clear that the AVC scheme gives better aggregate results in terms of number of code searches and blocking probability compared to other novel schemes. As mentioned earlier, the number of code searches is less due to linear increase in code checking with respect to layer number. In all other schemes, this increase is exponential. Therefore, the AVC scheme can be useful especially for real time calls.

**2.2 MULTI CODE ASSIGNMENT**

**2.2.1 SINGLE CODE ASSIGNMENT: MODIFIED**

38
For a vacant code, \( C_{l,n_l}, 1 \leq n_l \leq 2^{L-l} \), define vacant code adjacency as the number of vacant codes adjacent to \( C_{l,n_l} \) having same vacant ancestor code as for \( C_{l,n_l} \) in any layer between \( l+1 \) to \( L \). Whenever the system finds a first vacant code it checks for the consecutive other vacant codes in its vicinity on right side of the vacant code. There are two ways to find adjacency when

<table>
<thead>
<tr>
<th>Rate</th>
<th>Vacant Codes</th>
<th>Adjacency</th>
<th>Code Searches</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R )</td>
<td>( C_{1,1}, C_{1,10}, C_{1,13}, C_{1,50} )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( 2R )</td>
<td>( C_{1,3-6}, C_{1,11-12}, C_{1,15-16}, C_{1,19-20}, C_{1,23-26}, C_{1,51-52} )</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>( 4R )</td>
<td>( C_{1,5-8}, C_{1,29-32}, C_{1,41-44}, C_{1,57-60} )</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( 8R )</td>
<td>( C_{1,33-40} )</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>( 2R )</td>
<td>( C_{2,3}, C_{2,6}, C_{2,8}, C_{2,10}, C_{2,12}, C_{2,26} )</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>( 4R )</td>
<td>( C_{2,3-4}, C_{2,15-16}, C_{2,21-22}, C_{2,29-30} )</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>( 8R )</td>
<td>( C_{2,17-20} )</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( 4R )</td>
<td>( C_{3,1}, C_{3,8}, C_{3,11}, C_{3,15} )</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>( 8R )</td>
<td>( C_{3,9-10} )</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>( 8R )</td>
<td>( C_{4,5} )</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>
a new call with rate $2^{l-1}R$, $1 \leq l \leq L$ arrives:

1. After locating a vacant code, search adjacent vacant codes one by one, adjacency will be of the form $1, 3, 5, 7, \ldots, 2^{l-1} - 1$, $1 \leq l \leq L$.

2. Search the parent code(s) of vacant code. If they are vacant, then adjacency increases as:

$$1, 3, 5, 7, \ldots, 2^{l-1} - 1,$$

where $1 \leq l \leq L$. If parent is blocked, then the adjacency is 0.

Both the methods lead to the same amount of code blocking. However, later lead to less number of code searches. The searching of a parent code requires search of its two children codes. This reduces number of code searches significantly. For lower requested rates, the second method is preferred. A new call of the rate $2^{l-1}R$ will be assigned to a vacant code which has minimum adjacency i.e. to that portion of the code tree which has minimum number of vacant codes. For illustration, consider a 7 layer code tree shown in Figure 2.9. Table 2.6 lists all vacant codes in layers 1, 2, 3, 4, 5, along with their respective adjacency and the number of code searches required.

Our scheme further reduces the code searches at the lower layers using adjacency of higher layer vacant codes. Also, if a code is blocked or busy, its parent code is not searched as it will be vacant codes. Also, if a code is blocked or busy, its parent code is not searched as it will be blocked or busy too. If the adjacency of a layer $l$ is $A_l$, the adjacency of layer $(l-1)$

$$A_{l-1} = 2 \times A_l + 1$$

(2.14)

### 2.2.2 MULTICODE EXTENSION

As discussed earlier, multi code assignment can lead to significantly less code blocking as compared to single code assignment. The multi code extension of our scheme will initially search a vacant code for a call of the rate $2^{l-1}R$, $1 \leq l \leq L$ as in section 2.2.1 using single code scheme. If the system does not have a vacant code of rate $2^{l-1}R$, the code will be blocked using single code scheme. The multi code assignment is used which divides the incoming call rate. Let the system has $m$ rakes. The multi code scheme can have various refinements.
A. MAXIMUM FRACTIONS SCHEME

Find \( \max(i) \mid \sum_{j=1}^{l} 2^{i-1} R = 2^{l-1} R, i \leq m, 1 \leq l_j \leq l \). The maximum fraction algorithm converts the rate into maximum fraction, so that all the rakes are utilized for handling the call. For each of the fractions, the code assignment scheme given in section 2.2.1 is used. Therefore, one call may use up to \( m \) fractions to handle incoming call. The scheme is complex and costly but leads to very small code/call blocking as the tree fragmentation is reduced due to lower rate fractions.

For illustration, consider Figure 2.9 for a call of the rate 16\( R \) and 4 rakes, the algorithm will divide it into two 8\( R \) rates and check total rakes used or not. So, it will further divide one of the 8\( R \) into two 4\( R \)s. Still numbers of fractions are not equal to rakes. The algorithm will further divide one of the 4\( R \) into two 2\( R \) rates and assign vacant codes to them.

B. MINIMUM FRACTIONS SCHEME

Find \( \min(i) \mid \sum_{j=1}^{l} 2^{i-1} R = 2^{l-1} R, i \leq m, 1 \leq l_j \leq l \). The scheme converts rate into least number of fractions. Each fraction is handled by different rake. The algorithm is simple and cost effective, but may lead to higher code blocking as the code fragmentation increases.

For illustration, consider Figure 2.9 for a call of the rate 16\( R \) and system of 4 rakes, the algorithm will divide it into two 8\( R \) rates and will search two vacant 8\( R \) codes. Code tree has only one vacant 8\( R \) code. One of the 8\( R \) will be further divided into two 4\( R \) fractions and will be assigned to vacant codes. When a call rate is divided into fractions, there are two ways of finding suitable code for rate fractions.

C. SCATTERED MULTICODE SCHEME

An incoming call is divided into maximum fractions or minimum fractions as defined above. Let \( r \) be the number of rate fractions. Scattered multi code scheme will treat each rate fraction as a distinct call, and a vacant code for every rate fraction is assigned as discussed in single code scheme. This will lead to higher number of code searches with lesser code blocking. This scheme is best suited for maximum fraction division.
AVCS: Adjacent vacant code scattered, AVCG: Adjacent vacant code grouped, CFAS [22]: Crowded first assignment scattered, CFAG [22]: Crowded first assignment grouped, LCAS [22]: Left code assignment scattered, LCAG [22]: Left code assignment grouped.

Figure 2.10: Comparison of number of code searches (a) and code blocking probability (b) for distribution: [20,20,20,20].
Figure 2.11: Comparison of number of code searches (a) and code blocking probability (b) for distribution: (a) [10,10,10,30,40].
D. GROUPED MULTI CODE SCHEME
An incoming call is divided into maximum fractions or minimum fractions as defined above. Let \( r \) be the number of rate fractions. Grouped multi code scheme will assign vacant code to all rate fractions as close as possible. This will lead to lesser code fragmentation when codes are released.

2.2.3 RESULTS
The probability distribution matrix \([p_1, p_2, p_3, p_4, p_5]\), where \( p_i, \ i \in [1, 5] \) is the code tree capacity portion used by the \( i \)th class users. Three distribution scenarios are analyzed and are given by

- \([20, 20, 20, 20, 20]\), uniform distribution.
- \([10, 10, 10, 30, 40]\), high rates calls dominating.

The scattered and grouped multi code schemes are compared with CFA and LCA scattered and grouped multi code schemes. The various schemes are denoted as adjacency scattered (AVCS), adjacency grouped (AVCG), CFA scattered (CFAS), CFA grouped (CFAG), LCA scattered (LCAS), and LCA grouped (LCAG). The comparison of number of code searches and code blocking probability for various schemes is given in Figure 2.10 and Figure 2.11. The AVCG scheme requires least number of searches. Further, the comparison of blocking probability for various schemes is also shows clearly that the AVCS version provides least code blocking. Therefore, the adjacency scheme can be incorporated in WCDMA networks.

2.3 CONCLUSION
3G and beyond wireless networks are designed to handle multimedia rates better. Call processing delay and jitter are the significant QoS parameters for most of the real time calls. The chapter proposed a fast, single code, and multi code assignment scheme. The scheme can be even better when the system favors real time calls. The number of codes searched in the AVC scheme is significantly lesser than popular crowded first scheme. The online calculation of codes searched reduces the cost, complexity, and buffer size at the transmitter. The code blocking is also comparable or superior to existing single code schemes. For multi codes, code searches increases with number of rakes, which increases call establishment delay. The adjacency of higher layers can reduce searches at lower layer significantly in our scheme is used to locate
optimum codes in multi code assignment. The AVC scheme can be combined with existing FSP to increase speed of call assignment process further.