An OVSF Code Assignment Scheme Utilizing Multiple RAKE Combiners for W-CDMA ⋆

Li-Hsing Yen * Ming-Chun Tsou

Dept. of Computer Science and Information Engineering, Chung Hua University, Hsinchu, Taiwan 300, Republic of China

Abstract

Orthogonal variable spreading factor (OVSF) codes have been proposed as the channelization codes used in the wideband CDMA access technology of IMT-2000. OVSF codes have the advantages of supporting variable bit rate services, which is important to emerging multimedia applications. The objective of OVSF code assignment algorithm is to minimize the probability of code request denial due to inappropriate resource allocation. In this paper, we propose an efficient OVSF code assignment scheme that utilizes multiple RAKE combiners in user equipments. Our approach finds in constant time all feasible codewords for any particular request, trying to minimize both rate wastage and code fragments. When working together with an independent code replacement scheme, our approach has the same code request denial rate as previous work but has lower code management overhead. If code replacement is not used, our approach still has a bit of improvement on request denial rate.

Key words: OVSF, IMT-2000, Code Assignment, Radio Resource Management, CDMA

1 Introduction

UMTS/IMT-2000 aims to provide not only voice services that are offered by the second-generation mobile systems, but also higher data rate and variable bit rate services that support differentiated quality-of-service (QoS) for emerging multimedia applications. W-CDMA, selected as the technology for use in the UMTS terrestrial radio access (UTRA), employs direct spread code division multiple access (DS-CDMA) technique, where each user equipment (UE) uses different orthogonal codes on the same frequency band. To support fixed- and mixed data rate services, the 3rd Generation Partnership Project (3GPP) proposed orthogonal variable spreading factor (OVSF) codes as the channelization codes used in W-CDMA. OVSF codes, providing variable data rates by using variable spreading factors (the number of chips for a data bit), can be generated according to an OVSF code tree. OVSF code tree is a binary tree where OVSF codes with spreading factor f are placed at the $(1+\log f)$-th level. Each code in the tree has twice data rate than its child has. Once a code has been allocated to a UE, all its ancestors and descendants in the tree can no longer be allocated to other UEs. With the dynamic nature of code request arrival and code usage time, it becomes an important performance issue how to effectively allocate codes to various data rate requests so as to minimize the code blocking rate (the probability that a request is rejected).

Existing code assignment algorithms toward this direction [1–4] can be classified into two categories: single-code assignment and multi-code assignment.

* This work was supported by the National Science Council, Taiwan, under Contract NSC-91-2213-E-216-002

* Corresponding author
Email address: lhyen@chu.edu.tw (Li-Hsing Yen).

Single-code assignment assumes that each UE has only one RAKE combiner so the system can only allocate one code to it. For multi-code assignment, a UE has more than one RAKE combiner so its request can be honored by allocating multiple codes. Due to the dynamic nature of code usage, code assignment algorithm may suffer from code fragmentation problem, which refers to the circumstance that the system has enough capacity but cannot grant a request simply because the code tree is too fragmented. Code fragmentation may occur to both single-code and multi-code assignments, but it is believed that the problem is less serious in the multi-code case since data rates can be aggregated there. Code fragments can be compacted by using a code replacement algorithm that tries to exchange some allocated codes with unallocated codes of the equal spreading factor in order to obtain codes of lower spreading factors (i.e., higher data rates). However, the number of code replacements required is considered code management overhead (code exchange pays signaling cost) and should be kept minimal whenever possible.

In the paper, we propose a multi-code assignment algorithm that aims to minimize the number of code fragments. The technique of dynamic programming is used to build a codeword table in advance so that when a request is received, all feasible codewords can be found in the corresponding table entries. The one that minimizes the number of code fragments is then selected for allocation. The simulation result shows that our scheme outperforms previous work in the code replacement overhead. Even without the aid of any code replacement scheme, our scheme has an equal to lower code blocking rate than any counterpart has.

The rest of the paper is organized as follows. The OVSF code system is described in Section 2. Section 3 details our channelization code assignment

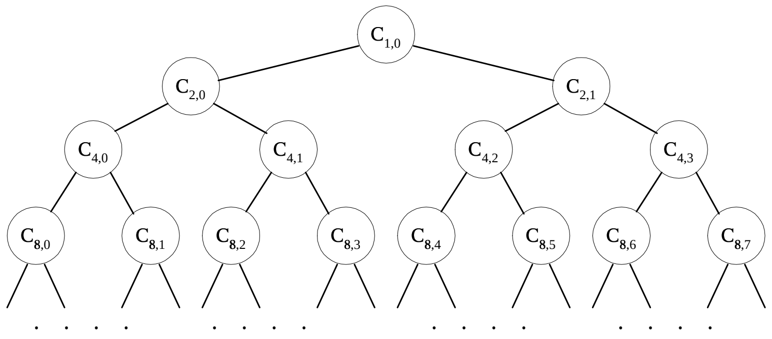

Fig. 1. An OVSF code tree.

scheme. Complexity analysis and numerical results are shown in Section 4. Section 5 concludes our work.

2 Problem Statement

OVSF codes collectively form a tree structure as shown in Fig. 1. Each level in the code tree defines channelization codes of the same spreading factor (SF). Codes of the same SF provide equal data rate, which is the double of that of the next level codes. We denote a single channelization code as $C_{f,n}$, where $f = 2^i$ for some integer i is the SF of the code and n is a sequence number ranging from 0 to $f - 1$. We may omit the sequence number whenever it is irrelevant to our discussion.

In order to keep the orthogonality among codes of different levels, a code can be assigned to a UE if and only if neither code on the path from this code to the root nor code in the subtree of this code is assigned. In other words, once a code has been allocated to UE, all codes on the path from the root of the tree to this node and all codes in the subtree of the code can no longer be allocated. If we remove all such codes and associated incident edges from the code tree once a code has been allocated, the result will be a forest of

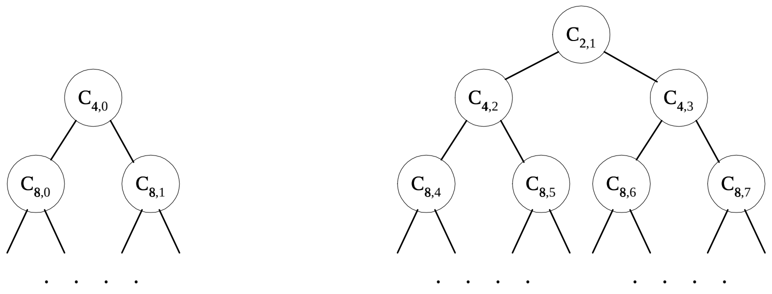

Fig. 2. Code fragments after allocating $C_{4,1}$.

complete binary trees, each represents a code fragment that can be directly allocated or decomposed for further code request. For example, if code $C_{4,1}$ of Figure 1 has been assigned to some UE, codes on the path from the root to $C_{4,1}$, *i.e.*, $C_{1,0}$ and $C_{2,0}$, and all codes in the subtree rooted at $C_{4,1}$ can no longer be allocated. After removing these nodes and associated incident edges, the result will be a forest of two complete binary trees, one representing code fragment $C_{4,0}$ and the other $C_{2,1}$, as shown in Fig. 2.

For practical reasons, the entire code tree is initially divided into four subtrees, each represents a code fragment of SF 4. Let

$$\Omega = \langle S_4, S_8, S_{16}, S_{32}, S_{64}, S_{128}, S_{256} \rangle$$

represent the numbers of various spreading codes available for allocation at any time, where S_f denotes the number of available code fragments of SF f. Initially, $\Omega = \langle 4, 0, 0, 0, 0, 0, 0 \rangle$. Since one code fragment corresponds to one complete binary tree, the number of complete binary trees in the forest is simply $N(\Omega) = S_4 + S_8 + S_{16} + S_{32} + S_{64} + S_{128} + S_{256}$.

Let R be the basic data rate that a code of SF 256 can offer. Clearly, the data rate offered by a code of SF $f = 2^i$ is $2^{8-i}R$, where $i \in [2, 8]$ is an integer. When a UE requests a data rate rR, any code of SF $f' = 2^j$, where j is an integer in $[2, 8]$, such that $2^{8-j} \geq r$ can be allocated to grant the request. How to select a code to grant the request is the *code placement* problem [5]. When no code can be allocated, a *code blocking* occurs. One reason behind code blocking is due to insufficient system capacity. The other reason is that the code tree becomes too fragmented. *Code fragmentation problem* refers to the circumstance that the system has enough capacity but cannot grant a request simply because the capacity does not belong to a single code. Code fragmentation can be resolved by switching some allocated codes with unallocated ones. For example, a system with $\Omega = \langle 0, 0, 0, 0, 0, 0, 2 \rangle$ cannot grant a request for data rate $2R$, since there is no code of SF 128. However, by switching a allocated code of SF 256 with an available equal-rate code, two available codes of SF 256 can be combined into a single code of SF 128. How to find and replace code fragments to grant a request is the *code replacement* problem [5]. Code placement scheme, optionally with code replacement scheme, aims to reduce code blocking rate, *i.e.*, the probability that a code request cannot be granted.

If we allocate a data rate which exceeds UE's actual demand, some amount of date rate is wasted. Rate wastage is critical to the code blocking rate (it diminishes system capacity) and should be kept minimal whenever possible.

Solving code placement problem involves two steps. First, determine the data rate to be allocated. Second, locate a code in the code tree that corresponds to the data rate. If UE is equipped with only one RAKE combiner, the first step is straightforward: find a code rate $2^i R$ that minimizes rate consumption. As to locate a code of rate $2^i R$ in the code tree, there have been several approaches proposed. Among them, leftmost allocation [6] and crowded-first [5] allocation will be mentioned in our discussion.

If UE is equipped with k RAKE combiners ($k > 1$) and demands data rate rR, we can assign UE a *multi-code*, which is a collection of maximal k codes with aggregated data rate not less than rR. Formally, the set of all feasible multi-codes $\omega(r, k)$ is the set of all vectors $\langle a_6, a_5, a_4, a_3, a_2, a_1, a_0 \rangle$, where $a_i \in Z^+ \cup \{0\}$, that satisfy $a_6 \cdot 2^6 + a_5 \cdot 2^5 + \cdots + a_0 \cdot 2^0 \geq r$ subject to $\sum_{i=1}^{6} a_i \leq k$. In particular, we are interested in a subset of $\omega(r, k)$ that minimizes rate consumption, which is not trivial to find since many combinations of codes need to be considered. As an example, if a UE equipped with three RAKE combiners demands data rate of $6R$, totally three multi-codes minimize rate consumption (actually they waste no rate), as shown below:

- one code of spreading factor 64 (rate $4R$) plus one code of spreading factor 128 (rate $2R$)
- one code of spreading factor 64 (rate $4R$) plus two codes of spreading factor 256 (rate R each)
- three codes of spreading factor 128 (rate $2R$ each)

Generally speaking, rate wastage will be less serious if UEs have multiple RAKE combiners rather than only one, since code fragments can be better utilized with multiple RAKE combiners.

Once the set of minimal-wastage multi-codes is determined, we must select one from the set to be allocated to the UE. A good multi-code assignment algorithm should select a multi-code that minimizes code blocking rate. However, as future requests cannot be known in advance, an optimal on-line code assignment algorithm seems impossible. Existing solutions thus take heuristic approach. The scheme in [2] favors allocating the multi-code that maximizes the number of small-SF codes left. When there is a tie to break, it selects the one that uses the least number of codes. In [4], Shueh et al. proposed a scheme that first chooses a multi-code that represents the request rate in binary-number form. The multi-code is then adjusted in accordance with the number of RAKE combiners that the UE has. This approach may allocate more small-SF codes than necessary, therefore increasing rate wastage.

3 Proposed Scheme

Let $Req(r, k)$ be a code request for data rate rR that is submitted by a UE equipped with k RAKE combiners. The basic idea behind our scheme is that the set of all possible multi-codes that satisfy $Req(r, k)$ can be evaluated off-line and in advance. In case we have two or more multi-code candidates, we select the one that minimizes the number of remaining code fragments.

A multi-code is denoted as $C = \langle a_6, a_5, a_4, a_3, a_2, a_1, a_0\rangle$, where a_i denotes the number of codes with rate 2^i that C uses. The number of codes used by multi-code C is $N(C) = \sum_{i=0}^{6} a_i$. When $N(C) = 1$, the multi-code C may be simply denoted as C_f, where f is the spreading factor of the only code used.

Let $C = \langle a_6, a_5, a_4, a_3, a_2, a_1, a_0\rangle$ and $C' = \langle a'_6, a'_5, a'_4, a'_3, a'_2, a'_1, a'_0\rangle$ be two multi-codes. We define $C + C'$ as $\langle a_6 + a'_6, a_5 + a'_5, a_4 + a'_4, a_3 + a'_3, a_2 + a'_2, a_1 + a'_1, a_0 + a'_0\rangle$. Let $W(r, k)$ be the set of all possible multi-codes that have data rate r and use k codes. We define $C \oplus W(r, k)$ as $\{C + C' | C' \in W(r, k)\}$. $W(r, k)$ can be computed as follows. $W(r, 1) = \{C_f\}$ if $r = 256/f$ and $W(r, 1) = \emptyset$ otherwise. For a general $W(r, k)$, we can examine every integer g such that $0 \leq g \leq \lfloor \log_2 r \rfloor$ to see if any multi-code of $W(r, k)$ may include a code of spreading factor 2^{8-g} (*i.e.* $C_{2^{8-g}}$). If the code is indeed included, all the multi-

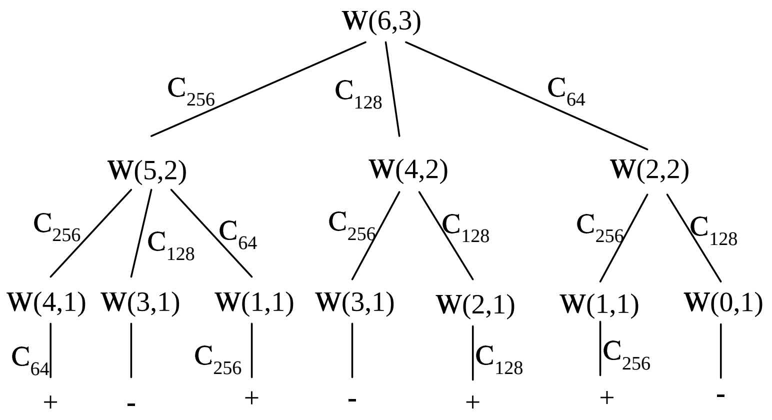

Fig. 3. Process of computing $W(6,3)$.

codes in $W(r,k)$ that include it can be obtained by $C_{2^{8-g}} \oplus W(r-2^g, k-1)$. Therefore we have

$$W(r,k) = \bigcup_{g=0}^{\lfloor \log_2 r \rfloor} C_{2^{8-g}} \oplus W(r-2^g, k-1).$$

In this way, we decompose the problem of solving $W(r,k)$ into smaller subproblems, which can be further decomposed. Fig. 3 shows the process of evaluating $W(6,3)$. Each path from the root to a leaf labeled '+' sign represents a valid multi-code. The multi-code consists of every code that is associated with an edge on the path.

In solving a particular $W(r,k)$, there may be some duplicated $W(r',k')$'s to be evaluated, where $r' < r$ and $k' < k$. We use the technique of dynamic programming [7] to avoid redundant computation. Table 1 shows the result.

Let $M(r,k) = \bigcup_{1 \le j \le k} W(r,j)$. Given a request $Req(r,k)$, all possible multi-codes that can be assigned without rate wastage are in the set $M(r,k)$. If $|M(r,k)| = 0$, we look up for the minimal $r' > r$ such that r' is not greater than the system capacity and $M(r',k)$ is not empty. If no such r' can be found, this request is rejected due to insufficient system capacity. Otherwise, we grant

k \ r	1	2	3	4	5	6	…
1	(0,0,0,0,0,0,1)	(0,0,0,0,0,1,0)	-	(0,0,0,0,1,0,0)	-	-	…
2	-	(0,0,0,0,0,0,2)	(0,0,0,0,0,1,1)	(0,0,0,0,0,2,0)	(0,0,0,0,1,0,1)	(0,0,0,0,1,1,0)	…
3	-	-	(0,0,0,0,0,0,3)	(0,0,0,0,0,1,2)	(0,0,0,0,0,2,1)	(0,0,0,0,1,0,2) (0,0,0,0,0,3,0)	…
4	-	-	-	(0,0,0,0,0,0,4)	(0,0,0,0,0,1,3)	(0,0,0,0,0,2,2)	…
…	…	…	…	…	…	…	…

Table 1

Table for $W(r,k)$.

the request with rate wastage $(r' - r)$.

If the set of minimal-wastage multi-codes has only one element ($|M(r,k)| = 1$ or $|M(r',k)| = 1$), the only multi-code is chosen. If the set has two or more multi-codes, we select the one that minimizes the number of code fragments left. For example, assume that $\Omega = \langle 0,0,0,0,1,3,1\rangle$ and a request $Req(6,3)$ arrives. From Table 1 we know that

$$M(6,3) = \{\langle 0,0,0,0,1,1,0\rangle, \langle 0,0,0,0,1,0,2\rangle, \langle 0,0,0,0,0,3,0\rangle\}.$$

Table 2 shows all possible allocations. Clearly, multi-code $\langle 0,0,0,0,0,3,0\rangle$ results in the least number of code fragments and thus will be selected for allocation.

If there is a tie to break, the one that uses least codes will be selected. As an example, consider $Req(6,3)$ and $\Omega = \langle 0,0,0,1,2,1,0\rangle$. Table 3 shows all possible allocation results. As all three multi-codes result in two code fragments left, the one that uses least codes, i.e., $\langle 0,0,0,0,1,1,0\rangle$, will be selected.

Once the multi-code is selected, how to locate all code fragments of the multi-code in the code tree becomes straightforward. We may allocate these codes one by one, using any approach that is proposed for single-code allocation. If

Multi-code	Ω'	$N(\Omega')$
$\langle 0,0,0,0,1,1,0\rangle$	$\langle 0,0,0,0,0,2,1\rangle$	3
$\langle 0,0,0,0,1,0,2\rangle$	$\langle 0,0,0,0,0,2,1\rangle$	3
$\langle 0,0,0,0,0,3,0\rangle$	$\langle 0,0,0,0,1,0,1\rangle$	2

Table 2

Possible code allocations for $Req(6,3)$ with $\Omega = \langle 0,0,0,0,1,3,1\rangle$. Ω' denotes the contents of Ω after code allocation.

Multi-code	Ω'	$N(\Omega')$
$\langle 0,0,0,0,1,1,0\rangle$	$\langle 0,0,0,1,1,0,0\rangle$	2
$\langle 0,0,0,0,1,0,2\rangle$	$\langle 0,0,0,1,1,0,0\rangle$	2
$\langle 0,0,0,0,0,3,0\rangle$	$\langle 0,0,0,1,1,0,0\rangle$	2

Table 3

Possible code allocations for $Req(6,3)$ with $\Omega = \langle 0,0,0,1,2,1,0\rangle$. Ω' denotes the contents of Ω after code allocation.

any one of the code fragments cannot be located due to code fragmentation problem, a code replacement algorithm is invoked to resolve the problem.

4 Performance Evaluation

4.1 Complexity Analysis

We shall analyze the number of codewords that will be examined for a particular request. This stands for the time complexity of our approach. Experimental results are presented in the next section.

Let us start with $W(r,k)$. Each element of $W(r,k)$ is a partition of integer r into parts which are powers of two, with the additional constraint that the number of parts is k. In the literature, *binary partition function*, denoted by $b(r)$, counts the number of partitions of r into powers of two without the restriction on the part size. Readily,

$$b(r) = \lim_{m\to\infty} \sum_{k=1}^{m} |W(r,k)|.$$

Both the function and its evaluation have been well investigated. In fact, the binary partition function can be expressed as a recurrence relation

$$\begin{cases} b(2n+1) & = b(2n) \\ b(2n) & = b(2n-1) + b(n) \end{cases}$$

for any natural number n [8]. However, it appears that the asymptotic rate of growth is not known exactly [9]. Churchhouse [10] gives the asymptotic upper bound $b(r) \sim O(r^{0.5\cdot\log_2 r})$. Unfortunately, this is a poor approximation for small values of r [11], as the sequence $\{b(r)\}_r$ exhibits oscillatory behavior in some of its lower order terms [12]. Since we are primarily concerned with small r's, these theoretical results, though interesting, do not help much. Besides, the binary partition function does not consider part size constraint, making it only an approximation solution to our problem.

Figure 4 is a stacked bar diagram showing the number of multi-codes in $W(r,k)$ for $r = 1$ to 128 and $k = 1$ to 5. There are total 635 multi-codes. The oscillatory behavior is quite obvious. Given a request $Req(r,k)$, where $1 \le r \le 128$ and

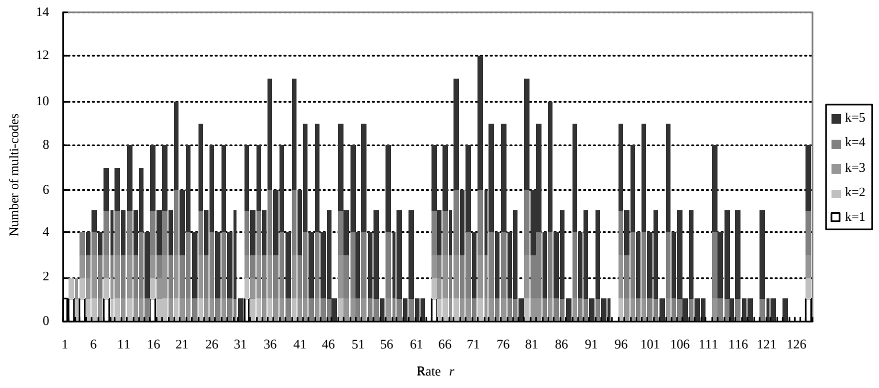

Fig. 4. Number of multi-codes in $W(r,k)$ for $r = 1$ to 128 and $k = 1$ to 5.

$1 \le k \le 5$, the number of multi-codes to be examined by our scheme is at most 12 (occurring on $r = 72$ and $k = 5$). From a practical perspective, the time complexity of our scheme is not much.

4.2 Numerical Results

We evaluated the performance of our multi-code assignment algorithm by simulation. The performances of [2] (referred to as OCCA) and [4] (referred to as FED) are also evaluated for comparison. We assume that the depth of the OVSF code tree is 9 and the initial set of multi-codes of the system is $\langle 0,0,4,0,0,0,0,0,0 \rangle$. Code requests are randomly generated by a Poisson process with mean arrival rate λ. The time interval for which a multi-code is used is assumed to be an exponentially distributed random variable with mean $1/\mu$. Each request $Req(r,k)$ is randomly generated with r being uniformly distributed between 1 and M, where M is 64 or 128. The value of k depends on r. When $1 \le r \le 64$, k is a random variable uniformly distributed between 1 and 4. When $65 \le r \le 128$, k is uniformly distributed in [2, 4]. Total 50000 requests are generated for each round. All algorithms use the same trace of each round as their input.

We measured the code blocking rate caused by each algorithm. The measured code blocking rate is essentially the sum of two components: one is due to insufficient system capacity and the other is due to code fragmentation. In the following figures, our method is labeled with OursA. OursB refers to a variation of our method that considers only multi-codes consisting of exactly k codes when handling $Req(r, k)$ (while OursA considers all multi-codes that use k *or less* codes). Once a multi-code has been chosen, leftmost [6] or crowded-first [5] code placement algorithm is used to allocate one by one all associated codes.

Figs. 5 and 6 show the results of $M = 64$ with leftmost and crowded-first code placement algorithms, respectively. In both scenarios, OursA as well as FED performs better than either OCCA or OursB. The performances of OCCA and OursB are similar. With $M = 128$, the performances of these four methods are all about the same, though OursA still slightly outperforms others. We also found that the performance of any method is higher with crowded-first code placement algorithm than with leftmost one. This is consistent with what previous report claimed [5].

We also conducted experiments to investigate the impacts of code replacement algorithm on code blocking rate. Table 4 shows the result of using DCA scheme [3] as the code replacement algorithm. With the aid of code replacement algorithm, code fragmentation can be totally eliminated and code blocking now occurs only for insufficient system capacity. Therefore, all four methods result in the same code blocking rate. However, each method requires different number of code replacements. The number of code replacements is considered

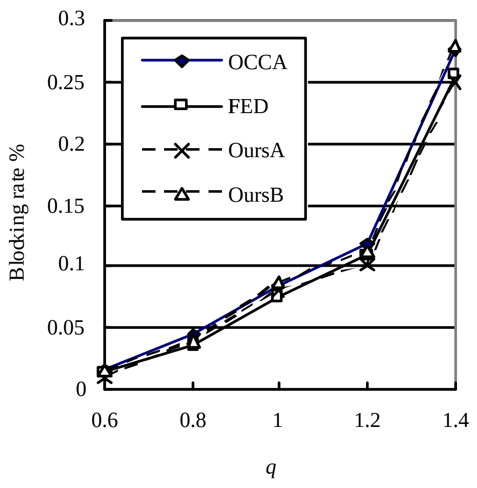

Fig. 5. Code blocking rate with leftmost code placement ($M = 64$, $q = \lambda/\mu$).

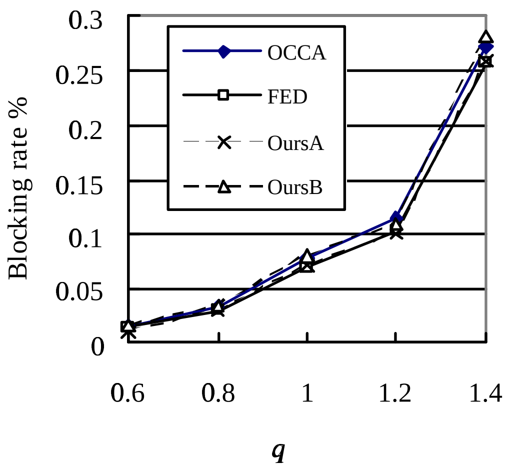

Fig. 6. Code blocking rate with crowded-first code placement ($M = 64$, $q = \lambda/\mu$).

the overhead of code management. Figs. 7 to 10 show the number of code replacements in different settings. We can see that OursA performs the best, which is followed by FED, OCCA, and then OursB.

As OursB considers only multi-codes in $W(r, k)$ rather than those in $M(r, k) = \bigcup_{1 \leq j \leq k} W(r, j)$ when handling request $Req(r, k)$, it does not perform well compared with the counterparts. The merit of OursB is that it examines less multi-codes than OursA does. We are interested in the amount of multi-code examinations that OursB can save. Table 5 shows the ratio of average number

	λ/μ				
	0.6	0.8	1.0	1.2	1.4
$M = 64$	0.012%	0.026%	0.062%	0.086%	0.218%
$M = 128$	2.56%	3.88%	5.46%	7.44%	8.65%

Table 4

Code blocking rate with code replacement.

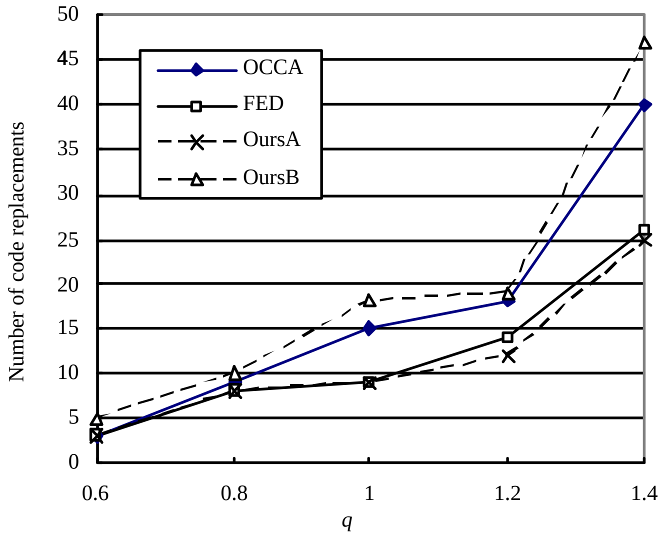

Fig. 7. Number of code replacements with leftmost code placement ($M = 64$, $q = \lambda/\mu$).

of multi-codes examined by OursB to that examined by OursA. We can see that about two third examinations are saved by OursB.

5 Conclusions

The objective of OVSF multi-code assignment algorithm is to minimize code blocking rate. The causes of code blocking are due to insufficient system capac-

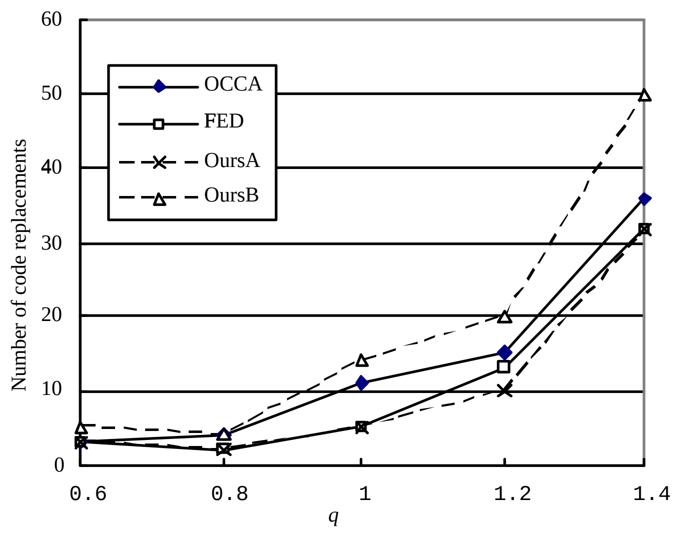

Fig. 8. Number of code replacements with crowded-first code placement ($M = 64$, $q = \lambda/\mu$).

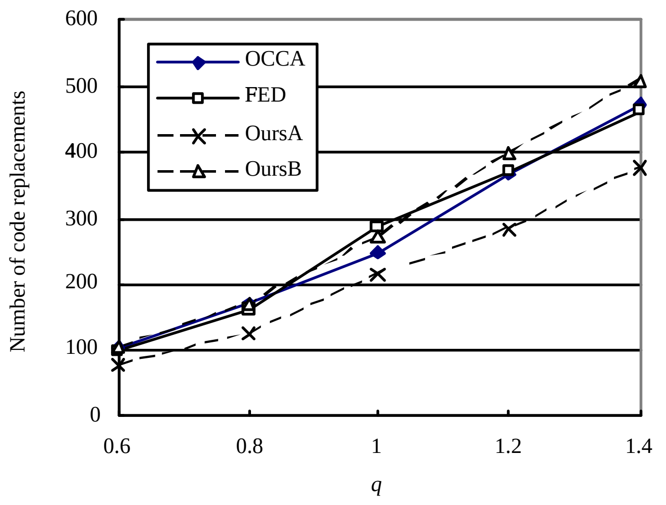

Fig. 9. Number of code replacements with leftmost code placement ($M = 128$, $q = \lambda/\mu$).

ity and code fragmentation. To preserve system capacity, rate wastage must be kept minimized. To eliminate code fragmentation, we can invoke a code replacement procedure. If UEs have multiple RAKE combiners, rate wastage can be reduced while the frequency of invoking code replacement can be lowered.

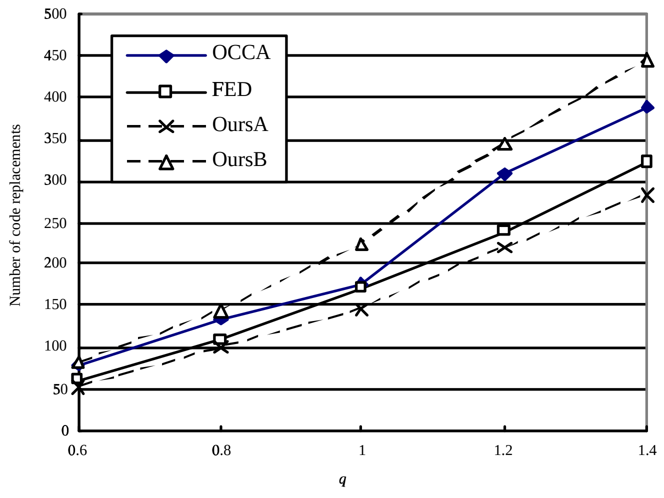

Fig. 10. Number of code replacements with crowded-first code placement ($M = 128$, $q = \lambda/\mu$).

	λ/μ				
	0.6	0.8	1.0	1.2	1.4
$M = 64$	36.29%	36.28%	36.28%	36.27%	36.29%
$M = 128$	31.39%	31.43%	31.47%	31.49%	31.50%

Table 5

Ratio of average multi-code examinations by OursB to that by OursA.

In this paper, we have proposed a multi-code assignment scheme that utilizes multiple RAKE combiners in UEs. The dynamic programming technique is used to find all feasible multi-codes for all possible requests. The result is stored in a table and can be retrieved on-line in a constant time. Among all candidate multi-codes, the scheme selects the one that minimizes both rate wastage and code fragments. The analysis shows that this strategy examines at most 12 candidate multi-codes for requests using up to five RAKE combiners. The simulation result shows that this strategy outperforms previous work in code replacement overhead. If no code replacement scheme can be used, our scheme has an equal to lower code blocking rate than any counterpart has.

References

[1] W.-T. Chen, H.-C. Hsiao, Y.-P. Wu, A novel code assignment scheme for W-CDMA systems, in: Proc. of IEEE 2001 Vehicular Technology Conference, Vol. 2, 2001, pp. 1182–1186.

[2] R.-G. Cheng, P. Lin, Ovsf code channel assignment for IMT-2000, in: Proc. of IEEE 2000 Vehicular Technology Conference, Vol. 3, 2000, pp. 2188–2192.

[3] T. Minn, K.-Y. Siu, Dynamic assignment of orthogonal variable-spreading-factor codes in W-CDMA, IEEE Journal on Selected Areas in Communications 18 (8) (2000) 1429–1440.

[4] F. Shueh, Z.-E. Liu, W.-S. Chen, A fair, efficient, and exchangeable channelization code assignment scheme for IMT-2000, in: Proc. of 2000 IEEE International Conference on Personal Wireless Communications, 2000, pp. 429–433.

[5] Y.-C. Tseng, C.-M. Chao, Code placement and replacement strategies for wideband CDMA OVSF code tree management, IEEE Trans. on Mobile Computing 1 (4) (2002) 293–302.

[6] R. Fantacci, S. Nannicini, Multiple access protocol for integration of variable bit rate multimedia traffic in UMTS/IMT-2000 based on wide-band CDMA, IEEE Journal on Selected Areas in Communications 18 (8) (2000) 1441–1454.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, Introduction to algorithms, McGraw-Hill Book Company, 1990.

[8] G. Alkauskas, Generalization of the Rodseth-Gupta Theorem on binary partitions, Lithuanian Mathematical Journal 43 (2) (2003) 103–110.

[9] C.-E. Froberg, Accurate estimation of the number of binary partitions, BIT 17 (1977) 386–391.

[10] R. Churchhouse, Binary partitions, in: A. Atkin, B. Birch (Eds.), Computers in Number Theory, Academic Press, 1971, pp. 397–400.

[11] J. L. Pfaltz, Evaluating the binary partition function when $n = 2^n$, Congressus Numeramtium 109 (1995) 3–12.

[12] H. S. Wilf, T. A. Scott, Asymptotic oscillations and binary partitions of integers, in: SIAM Annual Meeting, 1999.