Journal of Computer Science and Cybernetics, V.29, N.4 (2013), 351–357

A PRIME DECOMPOSITION ALGORITHM FOR SUPERCODES

KIEU VAN HUNG

The Office of Government, 01 Hoang Hoa Tham, Hanoi, Vietnam
kieuvanhung@chinhphu.vn

Tóm tắt. Siêu mã, một loại mã độ dài thay đổi có nhiều tính chất đặc biệt, đã được giới thiệu và nghiên cứu bởi Đỗ Long Vân và tác giả trong một số bài báo gần đây. Một siêu mã được gọi là nguyên tố nếu nó không thể phân tích thành tích ghép của hai siêu mã khác. Phân tích nguyên tố đối với một siêu mã là phân tích nó thành tích ghép của các siêu mã nguyên tố. Trong bài báo này, một thuật toán phân tích nguyên tố đối với các siêu mã có độ phức tạp tuyến tính đã được đề xuất. Tính duy nhất của việc phân tích một siêu mã thành một tích ghép duy nhất các siêu mã nguyên tố cũng được chứng minh.

Từ khóa. Mã, siêu mã, siêu mã nguyên tố, phân tích nguyên tố.

Abstract. Supercode, a particular case of hypercodes, has been introduced and considered by D. L. Van and the author in previous papers. A supercode is called prime if it cannot be decomposed as a catenation of two supercodes. The prime decomposition of a supercode L is to decompose L into prime supercodes. In this paper, a linear-time prime decomposition algorithm for supercodes is proposed. The uniqueness of the prime decomposition for supercodes is presented.

AMS Mathematics Subject Classification: 94A45, 68Q45.

Key words. Code, supercode, prime supercode, prime decomposition.

1. INTRODUCTION

Theory of length-variable codes has been initiated by M. P. Schützenberger and then developed by many others. Codes are closely related to formal languages. A code is a language such that every text encoded by words of the language can be decoded in a unique way or, in other words, every encoded message admits only one factorization into code-words. Codes are useful in many application areas such as information processing, data compression, cryptography, information transmission and so on [9]. For background of the theory of codes we refer to [1, 9].

Given a class C of codes. A decomposition of a code L in C is a catenation of several codes $L_1, L_2, \ldots, L_k$ in C such that $L = L_1.L_2 \ldots L_k$ and $k \geq 2$. If L cannot be further decomposed except for $L.\{\lambda\}$ or $\{\lambda\}.L$, where λ is the empty word, we say that L is a *prime* code in C. A. Mateescu, A. Salomaa, and S. Yu [10, 11] examined prime decompositions for regular languages and showed that it is decidable whether or not a given regular language has a decomposition and that, in general, the prime decomposition is not unique. J. Czyzowicz et al. [3] studied the prime decomposition problem for the class of prefix codes and proved that

the prime decomposition of a regular prefix code is unique. They also showed the importance of the prime decomposition for prefix codes in practice. In [6], Y. -S. Han et al. examined the prime decomposition problem for regular infix codes and showed that the prime decomposition in this case is not unique. An algorithm for testing the primality of regular infix codes was proposed. Also, it was shown that the prime decomposition can be computed in polynomial time. Then, in [7] the authors proved the uniqueness of the prime decomposition for regular outfix codes. A linear-time algorithm to compute the prime decomposition for regular outfix codes was presented. In [8], K. V. Hung and D. L. Van proposed a general approach to the prime decomposition problem. As applications, solutions for the prime decomposition problem are obtained, in a unified way, for several classes of codes. These classes are all subclasses of prefix codes and can be defined by binary relations. Algorithms to compute the prime decomposition for these classes was also given. Recently, W. Wieczorek [16] present an algorithm for the decomposition of a finite language. The algorithm is based on checking through some subsets of the states of a minimal acyclic deterministic finite-state automata. This author also investigate two additional algorithms: the first is based on the use of integer linear programming, and the second is based on finding cliques in a graph. It appears that the latter approaches are inappropriate in terms of time consumption. Moreover, the language decompositions and primality problem are also studied in [2, 4, 5, 12].

In this paper, we study the prime decomposition of supercodes and propose a linear-time prime decomposition algorithm for supercodes. The uniqueness of the prime decomposition for supercodes will be establshied. Note that, all supercodes are finite, and the prime decomposition problem for finite codes is not trivial at all because the primality test for finite codes is believed to be NP-complete [11]. Our work is motivated by the idea to solve the prime decomposition problem for regular infix codes [6] and regular outfix codes [7].

2. PRELIMINARIES

Let Σ be a finite alphabet, Σ^* be the set of all the words over Σ. The empty word is denoted by λ and Σ^+ stands for $\Sigma^* \setminus \{\lambda\}$. Any subset of Σ^* is a *language* over Σ. A language L of Σ^+ is a *code* over Σ if any word w in L^+ has exactly one factorization into words of L. Given a word u in Σ^*, the number of all the occurrences of letters in u is the *length* of u, denoted by $|u|$. A word u is a *subword* of a word v, if for some $n \geq 1$, $u = u_1 \dots u_n, v = x_0 u_1 x_1 \dots u_n x_n$ with $u_1, \dots, u_n, x_0, \dots, x_n \in \Sigma^*$. If $x_0 \dots x_n \neq 1$, then u is called a *proper subword* of v. A word u is called a *permutation* of a word v, if $|u|_a = |v|_a$ for all $a \in \Sigma$, where $|u|_a$ denotes the number of occurrences of a in u. And a word u is called a *permu-subword* (*proper permu-subword*) of a word v, if u is a subword (proper subword, resp.) of a permutation of v.

Definition 1. A language L of Σ^+ is a *supercode* over Σ, if no word in L is a proper permu-subword of another word in it.

Supercodes, a special kind of hypercodes, were introduced and considered in [13, 14, 15]. They have some interesting properties, especially, all supercodes are finite [13].

Example 1. a) Every uniform code over Σ which is a subset of $\Sigma^k, k \geq 1$, is a supercode.

b) The language $L = \{a^2b, ab^4\}$ over $\Sigma = \{a, b\}$ is a supercode because a^2b is not a proper permu-subword of ab^4 and vice versa.

c) Consider the language $L = \{ab^2ac, ab^5c, ac^3bac, ac^3b^4c\}$ over $\Sigma = \{a, b, c\}$. It is not difficult to check that no word in L is a proper permu-subword of another word in it. Thus, L

is a supercode.

d) The $L = \{abab, a^2b^3\}$ over $\Sigma = \{a, b\}$ is not a supercode because $abab$ is a proper subword of the permutation $abab^2$ of a^2b^3.

Lemma 1. *If L is a supercode and $L = L_1.L_2$ then L_1 and L_2 are also supercodes.*

Proof. Assume that L_1 is not a supercode. By definition, there are two distinct words u and v are in L_1 such that u is a proper permu-subword of v. Let w be a word in L_2. Since $L = L_1.L_2$, both uw and vw are in L. Obviously, uw is a proper permu-subword of vw. It contradicts the assumption that L is a supercode. Therefore, if L is a supercode, then L_1 should be a supercode. By the same argument, we can show that L_2 is also a supercode. ■

A finite-state automaton $\mathcal{A}$ is specified by a tuple $(Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a (finite) set of transitions, $s \in Q$ is the start state and $F \subseteq Q$ is a set of final states. Let $|Q|$ denote the number of states and $|\delta|$ the number of transitions in of $\mathcal{A}$. Then, the size $|\mathcal{A}|$ of $\mathcal{A}$ is defined as $|Q| + |\delta|$. If $t = (p, a, q)$ is a transition, where $p, q \in Q$ and $a \in \Sigma$, then we say that t is an *out-transition* of p and an *in-transition* of q. Also, p is called a *source state* of q and q a *target state of* p. Instead of writing $(p, a, q) \in \delta$, we often write also $\delta(p, a) = q$. Then δ is extended to a mapping from $Q \times \Sigma^*$ to Q in a normal way. A word w over Σ is accepted by $\mathcal{A}$ if there is a labeled path from s to a state in F, which is called a successful path in $\mathcal{A}$, and which spells out the word w. Thus, the language recognized by $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of the labels of all the successful paths in $\mathcal{A}$. The languages recognized by finite-state automata are called regular languages. Acyclic deterministic finite-state automata (ADFAs) are a proper subfamily of DFAs that define finite languages.

We say that the automaton $\mathcal{A}$ is *non-returning* if the start state of $\mathcal{A}$ has no any in-transitions, and that $\mathcal{A}$ is *non-exiting* if no final state of $\mathcal{A}$ has out-transitions. Clearly, if $\mathcal{A}$ is non-exiting then we may always assume that $\mathcal{A}$ has only one final state. Moreover, we always assume that $\mathcal{A}$ has only useful states, which means that each state of $\mathcal{A}$ must appears on at least one successful path in $\mathcal{A}$.

In this paper, we restrict ourselves to consider only acyclic deterministic finite-state automata, which are non-returning and non-exiting, denoted by N-ADFAs, for short.

Lemma 2. *For every supercode L there exists an N-ADFA recognizing L.*

Proof. Suppose L is a supercode. There is then an acyclic deterministic finite-state automata $\mathcal{A}$ recognizing $L, L = L(\mathcal{A})$. It is easy to see that if the start state of $\mathcal{A}$ has an in-transition then L cannot be a suffix code, and if a final state of $\mathcal{A}$ has an out-transition then L cannot be a prefix code; i.e. L cannot be a supercode. Hence, $\mathcal{A}$ must be both non-returning and non-exiting, i.e. $\mathcal{A}$ is an N-ADFA. ■

3. A PRIME DECOMPOSITION ALGORITHM

In this section we design a linear-time prime decomposition algorithm for supercodes and demonstrate the uniqueness of the prime decomposition for supercodes. For this we need some additional definitions and notations.

Definition 2. A supercode L is a *prime supercode* if $L \neq L_1.L_2$ for any supercodes L_1 and L_2.

Example 2. The supercode $L = \{a^2b, ab^4\}$ in Example 1, is not prime because $L = \{a\}.\{ab, b^4\}$,

where $\{a\}$ and $\{ab, b^4\}$ are supercodes. Clearly, the supercode $\{a\}$ is prime.

Definition 3. A state b in an N-ADFA $\mathcal{A}$ is called a *bridge state* of $\mathcal{A}$ if the following conditions hold:

(i) The state b is neither the start state nor a final state;

(ii) Every successful path in $\mathcal{A}$ must pass through b.

Thus, if b is a bridge state in the N-ADFA $\mathcal{A}$, then we can partition $\mathcal{A}$ into two subautomata $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $\mathcal{A}_1$ consists of all the states incomming to b, b including, and $\mathcal{A}_2$ consists of all the states outgoing from b, b including. It is easy to verify that $L(\mathcal{A}) = L(\mathcal{A}_1).L(\mathcal{A}_2)$ from the second requirement in Definition 3. Fig. 1 illustrates a partition at a bridge state.

Lemma 3. *If a minimal N-ADFA $\mathcal{A}$ has a bridge state, where $L(\mathcal{A})$ is a supercode, then $L(\mathcal{A})$ is not prime.*

Proof. Since $\mathcal{A}$ has a bridge state b, we can partition $\mathcal{A}$ into two subautomata $\mathcal{A}_1$ and $\mathcal{A}_2$ at b. Then, by Lemma 1, $L(\mathcal{A}_1)$ and $L(\mathcal{A}_1)$ are supercodes and, therefore, $L(\mathcal{A}) = L(\mathcal{A}_1).L(\mathcal{A}_2)$ is not prime. ■

Lemma 4. *If a minimal N-ADFA $\mathcal{A}$ does not have any bridge states and $L(\mathcal{A})$ is a supercode, then $L(\mathcal{A})$ is prime.*

Proof. Assume that $L = L(\mathcal{A})$ is not prime. Then, L can be decomposed as $L_1.L_2$, where L_1 and L_2 are supercodes. By Lemma 2, there exist two minimal N-ADFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ for L_1 and L_2, respectively. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are non-returning and non-exiting, there are only one start state and one final state for each of them. We catenate $\mathcal{A}_1$ and $\mathcal{A}_2$ by merging the final state of $\mathcal{A}_1$ and the start state of $\mathcal{A}_2$ as a single state b. Then, the catenated automaton is the minimal N-ADFA for $L(\mathcal{A}_1).L(\mathcal{A}_2) = L$ and has a bridge state b, a contradiction. ■

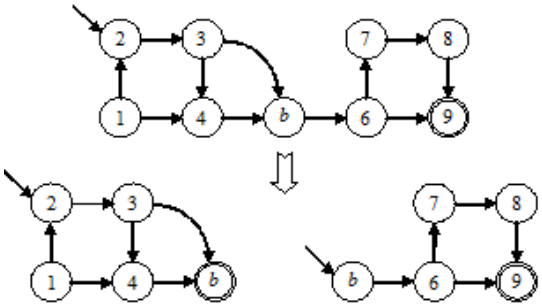

Fig.1. An example of partitioning of an N-ADFA at a bridge state b

Theorem 1. *A supercode L is prime if and only if the minimal N-ADFA for L does not have any bridge states.*

Proof. It follows immediately from Lemmas 3 and 4. ■

Lemma 3 shows that if a minimal N-ADFA $\mathcal{A}$ for a supercode L has a bridge state, then we can decompose L into a catenation of two supercodes using bridge states. In addition, we have a set B of bridge states for $\mathcal{A}$ and decompose $\mathcal{A}$ at b, then $B \setminus \{b\}$ is the set of bridge states for the resulting two automata after the decomposition.

Theorem 2. *Let $\mathcal{A}$ be a minimal N-ADFA for a supercode that has k bridge states, where $k \geq 1$. Then, $L(\mathcal{A})$ can be decomposed into $k+1$ prime supercodes, namely, $L(\mathcal{A}) = L_1.L_2 \ldots L_{k+1}$ and $L_1, L_2, \ldots, L_{k+1}$ are prime supercodes.*

Proof. Let $(b_1, b_2, \ldots, b_k)$ be the sequence of bridge states from the start state s to the final state f in $\mathcal{A}$. We prove the statement by induction on k. It is sufficient to show that $L(\mathcal{A}) = L'.L''$ such that L' is accepted by an N-ADFA $\mathcal{A}'$ with $k-1$ bridge states and L'' is a prime supercode.

We partition $\mathcal{A}$ into two subautomata $\mathcal{A}'$ and $\mathcal{A}''$ at b_k. By Lemma 1, $L(\mathcal{A}')$ and $L(AA'')$ are supercodes. Since $\mathcal{A}''$ has no bridge states, $L'' = L(\mathcal{A}'')$ is prime by Theorem 1. By definition of bridge states, all paths must pass through $(b_1, b_2, \ldots, b_{k-1})$ in $\mathcal{A}'$ and, therefore, $\mathcal{A}'$ has $k-1$ bridge states. Thus, if $\mathcal{A}$ has k bridge states, $k \geq 1$, then $L(\mathcal{A})$ can be decomposed into $k+1$ prime supercodes. ■

Note that Theorem 2 guarantees the uniqueness of prime decomposition for supercodes. Furthermore, finding the prime decomposition of a supercode is equivalent to identifying bridge states of its minimal N-ADFA by Theorems 1 and 2.

Given L, to verify the primality of L it suffices to verify whether $\mathcal{A}$ has bridge states or not. If not, then L is prime and is a prime decomposition of itself. If yes, we partition $\mathcal{A}$ at a bridge state into two subautomata $\mathcal{A}_1$ and $\mathcal{A}_2$. If both of $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ are prime then $L(\mathcal{A}_1).L(\mathcal{A}_2)$ is a prime decomposition of L. Otherwise, the above procedure is repeated for one among $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ or both of them according to the case.

Let B denote the set of bridge states of the given minimal N-ADFA $\mathcal{A}$. Clearly, the number of states in B is at most m, where m is the number of states in $\mathcal{A}$. Note that every time we partition $\mathcal{A}$ at a bridge state $b \in B$ into $\mathcal{A}_1$ and $\mathcal{A}_2$, then only states in $B \setminus \{b\}$ can be the bridge states of $\mathcal{A}_1$ and $\mathcal{A}_2$. Therefore, we can determine the primality of $L(\mathcal{A})$ by checking whether $\mathcal{A}$ has bridge states or not and compute a prime decomposition of $L(\mathcal{A})$ using only these bridge states. Since there are at most m bridge states in an N-ADFA for a supercode, we can obtain a prime decomposition of $L(\mathcal{A})$ after a finite number times, no more than m, of partitioning component automata at the bridge states in B.

The following result is due to Y. -S. Han, Y. Wang and D. Wood.

Lemma 5 ([6]). *We can compute the set of bridge states for a given N-ADFA $\mathcal{A} = (Q, \Sigma, \delta, s, f)$ in $O(|Q| + |\delta|)$ worst-case time using DFS (depth-first search).*

Based on this result we obtain the following theorem.

Theorem 3. *Given a minimal N-ADFA $\mathcal{A}$ for a supercode:*

(i) *We can determine the primality of $L(\mathcal{A})$ in $O(\mu)$ time;*

(ii) *We can compute the unique prime decomposition of $L(\mathcal{A})$ in $O(\mu)$ time if $L(\mathcal{A})$ is not prime;*

where μ is the size of $\mathcal{A}$.

Proof. By Lemma 5, the set of bridge states in $\mathcal{A}$ can be computed in $O(\mu)$ worst-case time. Therefore, if $\mathcal{A}$ has no bridge states then $L(\mathcal{A})$ is prime. Otherwise, by Theorem 2, we can compute the unique prime decomposition of $L(\mathcal{A})$ using bridge states in $O(\mu)$ time. ■

By virtue of Theorem 2, a prime decomposition algorithm for supercodes can be presented as follows.

Algorithm PrimeSP

Input: A supercode L.

Output: The prime supercodes $L_1, L_2, \ldots, L_{k+1}$ such that $L = L_1.L_2 \ldots L_{k+1}$ with $k \geq 0$.

1. Building a minimal N-ADFA $\mathcal{A} = (Q, \Sigma, \delta, s, f)$ for L.

2. Finding the sequence $(b_1, b_2, \ldots, b_k)$ of bridge states from s to f in $\mathcal{A}$ by using DFS (depth-first search).

3. If $\mathcal{A}$ does not have any bridge states, we obtain L. Otherwise, we partition $\mathcal{A}$ into $k+1$ subautomata $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_{k+1}$ at $b_1, b_2, \ldots$ and b_k, respectively. Therefore, we obtain $L_1 = L(\mathcal{A}_1), L_2 = L(\mathcal{A}_2), \ldots, L_{k+1} = L(\mathcal{A}_{k+1})$.

Let us give some examples.

Example 3. Consider the supercode $L = \{a^2b, ab^4\}$ over $\Sigma = \{a, b\}$ in Example 2. The minimal N-ADFA $\mathcal{A}$ for L has 2 bridge states which are b_1 and b_2 (see Fig. 2). By Algorithm PrimeSP, L may be decomposed uniquely into 3 prime supercodes, namely $L = \{a\}.\{a, b^3\}.\{b\}$, where $\{a\}, \{a, b^3\}$ and $\{b\}$ are prime supercodes.

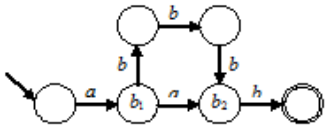

Fig.2. The minimal N-ADFA $\mathcal{A}$ for $L = \{a^2b, ab^4\}$

Example 4. Consider the supercode $L = \{ab^2ac, ab^5c, ac^3bac, ac^3b^4c\}$ over $\Sigma = \{a, b, c\}$ in Example 1. The minimal N-ADFA $\mathcal{A}$ for L has 4 bridge states which are b_1, b_2, b_3 and b_4 (see Fig. 3). By Algorithm PrimeSP, L may be decomposed uniquely into 5 prime supercodes, namely

$$L = \{a\}.\{b, c^3\}.\{b\}.\{a, b^3\}.\{c\}$$

where $\{a\}, \{b, c^3\}, \{b\}, \{a, b^3\}$ and $\{c\}$ are prime supercodes.

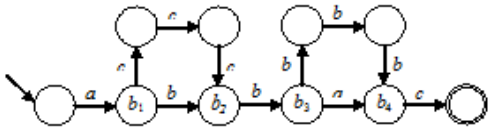

Fig.3. The minimal N-ADFA $\mathcal{A}$ for $L = \{ab^2ac, ab^5c, ac^3bac, ac^3b^4c\}$

4. CONCLUSIONS

This paper deals with factorization of codes, that is, representation of codes as a product of other codes. The problem is as follows: Let C be a class of code. Is every code of C factorizes into "prime" codes (that is ones that cannot factorize further in that way) in the same class C? And if such a factorization exists, is it unique? The problem of this kind is common and it pervades mathematics.

Motivated by the problem for infix and outfix codes by Yo-Sub Han at el., we have studied the problem over the so-called supercodes, a subclass of hypercodes. First, we have proved the "structure" lemma: every factorization of a supercode into codes is a supercode factorization (Lemma 1). Then we have invoked automata theory settings (minimal automata, key notions, etc.) of previous authors on which the consideration to characterize prime supercodes and further decide on the primality issue, shows and computes the unique supercode factorization of them.

We have proposed a linear-time prime decomposition algorithm for supercodes (Algorithm PrimeSP) and also demonstrated the uniqueness of the prime decomposition for supercodes (Theorem 3).

Acknowledgement. The author would like to thank the colleagues in Seminar "Mathematical Foundation of Computer Science" at Hanoi Institute of Mathematics for useful discussions and attention to the work. Especially, I express my sincere thanks to Professor Derick Wood and Professor Yo-Sub Han for providing their papers to my work.

REFERENCES

[1] J. Berstel and D. Perrin, *Theory of Codes*, Academic Press, New York, 1985.

[2] F. Biegler, M. Daley, and I. McQuillan, Algorithmic decomposition of shuffle on words, *Theoret. Comput. Sci.* **454** (2012) 38–50.

[3] J. Czyzowicz, W. Fraczak, A. Pelc, and W. Rytter, Linear-time prime decomposition of regular prefix codes, *Int. J. Found. Comput. Sci.* **14** (2003) 1019–1032.

[4] M. Domaratzki and K. Salomaa, On language decompositions and primality, *Lecture Notes in Computer Science* **6570** (2011) 63–75.

[5] Y.-S. Han, A. Salomaa, K. Salomaa, D. Wood, and S. Yu, On the existence of prime decompositions, *Theoret. Comput. Sci.* **376** (2007) 60–69.

[6] Y.-S. Han, Y. Wang, and D. Wood, Infix-free regular expressions and languages, *Int. J. Found. Comput. Sci.* **17** (2006) 379–394.

[7] Y.-S. Han and D. Wood, Outfix-free regular languages and prime outfix-free decomposition, *Fundam. Inform.* **81** (4) (2007) 441–457.

[8] K. V. Hung and D. L. Van, Prime decomposition problem for several kinds of regular codes, *Lecture Notes in Computer Science* **4281** (2006) 213–227.

[9] H. Jürgensen and S. Konstantinidis, Codes, In: G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages*, Springer, Berlin, 1997, 511–607.

[10] A. Mateescu, A. Salomaa, and S. Yu, On the decomposition of finite languages, *Technical Report* **222**, TUCS, 1998.

[11] A. Mateescu, A. Salomaa, and S. Yu, Factorizations of languages and commutativity conditions, *Acta Cyber.* **15** (2002) 339–351.

[12] A. Salomaa, K. Salomaa, and S. Yu, Length codes, products of languages and primality, *Lecture Notes in Computer Science* **5196** (2008) 476–486.

[13] D. L. Van, On a class of hypercodes, In: M. Ito and T. Imaoka (eds.), *Words, Languages and Combinatorics III*, World Scientific, 2003, 171–183.

[14] D. L. Van and K. V. Hung, On codes defined by binary relations, Part II: Vector characterizations and maximality, *Vietnam J. Math.* **39** (4) (2011) 425–442.

[15] D. L. Van, K. V. Hung, and P. T. Huy, Codes and length-increasing transitive binary relations, *Lecture Notes in Computer Science* **3722** (2005) 29–48.

[16] W. Wieczorek, An algorithm for the decomposition of finite languages, *Logic Journal IGPL* **18** (3) (2010) 355–366.

Received on February 26, 2013
Revised on October 16, 2013