JGRCS
Journal of Global Research in computer science

Volume 2, No. 1, January 2011

Journal of Global Research in Computer Science

ISSN-2229-371X

RESEARCH PAPER

Available Online at www.jgrcs.info

Data Hiding Using DNA sequence Compression

Prof. Samir Kumar Bandyopadhyay
Dept. of Computer Sc. & Engg, University of Calcutta
92 A.P.C. Road, Kolkata – 700009, India
Email: skb1@vsnl.com

Mr. Suman Chakraborty
B.P. Poddar Institute of Management and Technology
137, V.I.P. Road, Kolkata – 700052, India

Abstract: An attempt has been made to represent data hiding using DNA sequence compression. In this paper initially data has been converted into DNA sequence then compress the DNA sequence .Four compression technique has been used, one of the option will produces better compression depending upon the DNA sequence. Also decompression algorithm has been developed to get the original data.

Keywords: DNA Sequencing, Compression, Decompression, Encoding

INTRODUCTION

The term DNA sequencing refers to sequencing methods for determining the order of the nucleotide bases—adenine(a), guanine(g), cytosine(c), and thymine(t)—in a molecule of DNA.

Knowledge of DNA sequences has become indispensable for basic biological research, other research branches utilizing DNA sequencing, and in numerous applied fields such as diagnostic, biotechnology, forensic biology and biological systematic. The advent of DNA sequencing has significantly accelerated biological research and discovery. The speed of sequencing attained with modern DNA sequencing technology has been instrumental in the sequencing of the human genome, in the Human Genome Project. Related projects, often by scientific collaboration across continents, have generated the complete DNA sequences of many animal, plant, and microbial genomes [1-5].

The first DNA sequences were obtained in the early 1970s by academic researchers using laborious methods based on two-dimensional chromatography. Following the development of dye-based sequencing methods with automated analysis, DNA sequencing has become easier and orders of magnitude faster [6-8].

A massage representing a DNA sequence, with the combination of a, c, t, g is equivalent to DNA sequence of the original data. Compressing the DNA sequence by-1) 2-bits encoding method, 2) Exact matching method, 3) Approximate matching method, 4) For the approximate matching method. These compression techniques would be produce equivalent digital form of the DNA sequence and one of procedure will produce minimum number of bits [9-10].

© JGRCS 2010, All Rights Reserved

DATA HIDING SCHEME

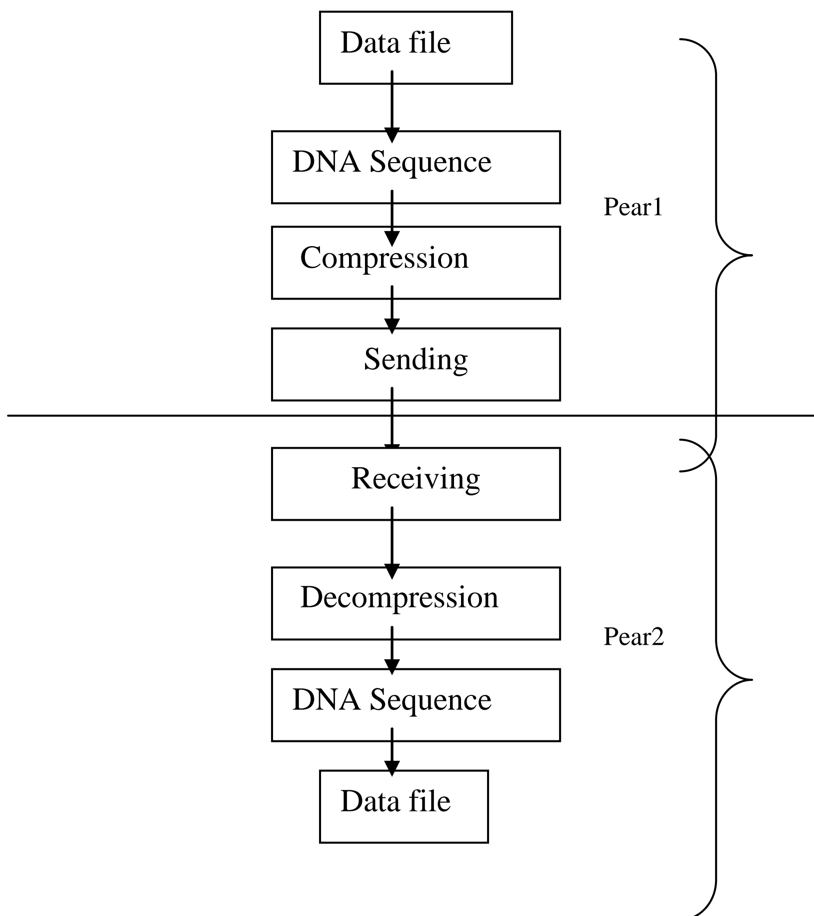

Figure 1: Flow Diagram of Data Hiding Scheme

ALGORITHM FOR CONVERTING DATA TO DNA SEQUENCE

In data hiding scheme, a letter (alphabet or digit) composed by 3 nucleotide. The possible nucleotides are U, C, A, and G. Hence, there are 4^3=64 combinations available to represents a latter .Total no. of letter is 63(26 no. for A-Z, 26 no. for a-z, 10 no. for 0-9and 1 no. for Space).

	U	C	A	G	
U	UUU-A	UCU-E	UAU-I	UGU-M	U
	UUC-B	UCC-F	UAC-J	UGC-N	C
	UUA-C	UCA-G	UAA-K	UGA-O	A
	UUG-D	UCG-H	UAG-L	UGG-P	G
C	CUU-Q	CCU-U	CAU-Y	CGU-c	U
	CUC-R	CCC-V	CAC-Z	CGC-d	C
	CUA-S	CCA-W	CAA-a	CGA-e	A
	CUG-T	CCG-X	CAG-b	CGG-f	G
A	AUU-g	ACU-k	AAU-o	AGU-s	U
	AUC-h	ACC-l	AAC-p	AGC-t	C
	AUA-i	ACA-m	AAA-q	AGA-u	A
	AUG-j	ACG-n	AAG-r	AGG-v	G
G	GUU-w	GCU-0	GAU-4	GGU-8	U
	GUC-x	GCC-1	GAG-5	GGC-9	C
	GUA-y	GCA-2	GAA-6	GGA-Space	A
	GUG-z	GCG-3	GAG-7	GGG-	G

Figire 2: DNA sequence to Alpha numeric Mapping Table

© JGRCS 2010, All Rights Reserved

Volume 2, No. 1, January 2011

Journal of Global Research in Computer Science

ISSN-2229-371X

RESEARCH PAPER

Available Online at www.jgrcs.info

DNA SEQUENCE COMPRESSION

Encoding Edit Operations

We consider three standard edit operations in our approximate matching algorithm. These are:

1) *Replace*. This operation is expressed as (*R*, *p*, *char*) which means replacing the character at position *p* by character *char*.

2) *Insert*. This operation is expressed as (*I*, *p*, *char*), meaning inserting character *char* at *p*.

3) *Delete*. This operation is written as (*D*, *p*), meaning deleting the character atposition *p*.

Let *C* denote "copy," then the following are two ways to convert the string "gaccttca" to "gaccgtca" via different edit

```
C C C C R C C C
g a  c c g  t  c a
g a  c c t  t  c a
          or

C C C C I C D C C
g a c c  g    t  ca
g a c c   t t  c  a
```

The first involves one replacement operation.The second involves one insertion and one deletion. It can be easily seen that there are infinitely many edit sequences to transform one string to another A list of edit operations that transform a string *v* to another string *u* is called an *Edit Transcription* of the two strings [9].This will be represented by an edit operation sequence $\lambda(u,v)$ that orderly lists the edit operations. For example, the edit operation sequence of the first edit transcription in the above example is $\lambda(gaccgtca, gaccttca) = \{(R,4, g)\}$; and for the second edit transcription, $\lambda(gaccgtca, gaccttca) = \{(I,4, g),(D,6)\}$.If we know the string *u* and an edit operation sequence $\lambda(u,v)$ from *v* to *u*, then the string *u* can be constructed correctly using λ.There are many ways to encode one string given another. Using the above example, we describe four ways to encode "gaccgtca" using string "gaccttca" supposing that the string "gaccttca" is located earlier in the sequence.

1) 2-bits encoding method. In this case,we can simply use 2 bits to encode each character; i.e., 00 for *a*, 01 for *c*, 10 for *g*, 11 for *t*. Thus "10 00 01 01 10 11 01 00" encodes "gaccgtca." It needs 16 bits in total.

2) Exact matching method.Wecan use (repeat position, repeat length) to represent an exact repeat. This way, for example,if we use 3 bits to encode an integer, 2 bits to encode a character, and use 1 bit to indicate if the next part is a pair (indicating an exact repeat) or a plain character, then the string "gaccgtca" can be encoded as {(0,4), *g*,(5,3)}, relative to "gaccttca." Thus, a 17-bit binary string "0 000 100 1 10 0 101 011" is required to encode the {(0,4), *g*,(5,3)}.

3) Approximate matching method. In this case, the string "gaccgtca" can be encoded as {(0,8),(*R*,4, *g*)}, or "0 000 111 100 100 10" in binary, with *R* encoded by 00, *I* encoded by 01, and*D*encoded by 11, and 0/1 indicating whether the next item is a doubleton or triple. A total of 15 bits is needed.

4) For the approximate matching method, if we use the edit operation sequence, then the string "gaccgtca" can be encoded as {(0,8),(*I*,4, *g*),(*D*,6)}, or "0 000 111 1 01 100 10 1 10 110," in total 21 bits.

C-Program for DNA Sequence Compression

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
char
str[30],temp1[60],temp[60],binary1[150],binary2[150],binar
y3[150],code1[150],str1[30],match1[4][2]={{'0','0'},{'0','1'},
{'1','0'},{'1','1'}},ch,match5[]={'a','c','g','t'},match6[]={'R','I','
D'},match7[3][2]={{'0','0'},{'0','1'},{'1','1'}};
char
match2[]={'0','1','2','3','4','5','6','7'},match3[8][3]={{'0','0','0'
},{'0','0','1'},{'0','1','0'},{'0','1','1'},{'1','0','0'},{'1','0','1'},{'1','
1','0'},{'1','1','1'}};
int
i,j=0,r,len,f2=0,len1,c=0,l,p,n,flag,k=0,f=1,z,w,len2,fl=0,len
3;
clrscr();
printf("\nEnter the string:");
gets(str);
len1=strlen(str);
    printf("\n\n1-Two bits encoding done");
for(z=0,w=1;z<len1;z++)
if((w%8==0) ||(z==len1-1))
{
temp1[w-1]=str[z];
temp1[w]='\0';
len=strlen(temp1);

         for(i=0;i<len;i++)
           for(k=0;k<4;k++)
             if(temp1[i]==match5[k])
                     {
                      binary1[c++]=match1[k][0];
                      binary1[c++]=match1[k][1];
                     }

w=1;
}
else
{
 temp1[w-1]=str[z];
 w++;
}
```

© JGRCS 2010, All Rights Reserved

```
binary1[c]='\0';
//printf(" %s",binary1);

     printf("\n\n\t2-Exact matchaing");
c=0;
for(z=0,w=1;z<len1;z++)
if((w%8==0) ||(z==len1-1))
{
temp1[w-1]=str[z];
temp1[w]='\0';
len=strlen(temp1);
            strcpy(str1,temp1);
            f=1;
              printf("\n\nString is:%s",temp1);
              printf("\n\nEnter the position");
              scanf("%d",&p);
              printf("\nEnter the chracter:");
              ch=getche();
              str1[p]=ch;
              flag=0;
              j=0;
              for(i=0;i<len;i++)
                 if(temp1[i]==str1[i])
                 {
                        j++;
                        if(flag==0)
                        {
                                  sprintf(temp,"%d",i);
                                  if(f==1)
                                  {
                                  strcpy(code1,temp);
                                  f=0;
                                  }
                                  else
                                  strcat(code1,temp);
                          flag=1;
                        }
                 }
                 else
                 {
                        if((j!=0)&&(j!=len-1))
                        {  sprintf(temp,"%d",j);
                          strcat(code1,temp);
                          len2=strlen(code1);
                          code1[len2]=str1[i];
                          code1[len2+1]='\0';
                         j=0;flag=0;
                        }
                  else
                             {
                             if(i!=0)
                               {
                               sprintf(temp,"%d",j);
                                  strcat(code1,temp);
                               len2=strlen(code1);
                               code1[len2]=str1[i];
                               code1[len2+1]='\0';
                                  j=0;flag=0;   f2=1;
                                  }
                              else
                               {

code1[0]=str1[i];code1[1]='1';code1[2]='\0';
                                            flag=1;
                                     }
                                   }
                             }
                             if(f2==0)
                             {
                             sprintf(temp,"%d",j);
                             strcat(code1,temp);
                             }
                                    len2=strlen(code1);
                                     printf("\n%s",code1);
                                     if((code1[0]>=0)&&(code1[0]<8))
                                        binary2[c++]='0';
                                     for(r=0;r<len2;r++)
                                       {

                      if((code1[r]>='a')&&(code1[r]<='t'))
                                                    {
                                                     for(i=0;i<4;i++)

if(code1[r]==match5[i])
                                                                 {

binary2[c++]='1';

binary2[c++]=match1[i][0];

binary2[c++]=match1[i][1];

binary2[c++]='0';
                                                                 }
                                                     }
                                               else
                                                  {        for(i=0;i<8;i++)

if(code1[r]==match2[i])
                                                                 {

binary2[c++]=match3[i][0];

binary2[c++]=match3[i][1];

binary2[c++]=match3[i][2];
                                                                 }
                                                    }


                                       }

w=1;
}
else
{
 temp1[w-1]=str[z];
 w++;
}
binary2[c]='\0';

               printf("\n\n\t3-Approximata matching");
 c=0;
for(z=0,w=1;z<len1;z++)
if((w%8==0) ||(z==len1-1))
{
temp1[w-1]=str[z];
```

© JGRCS 2010, All Rights Reserved

```
temp1[w]='\0';
len=strlen(temp1);
        printf("\nString is:%s",temp1);
        printf("\nEnter the position u want 2 Replace:");
        scanf("%d",&p);
        printf("\nEnter the chracter:");
        ch=getche();
         j=0;fl=0;
         code1[j++]='0';  code1[j]='\0';
         sprintf(temp,"%d",len);
         strcat(code1,temp);
         j=strlen(code1);
         code1[j]='R'; code1[j+1]='\0';
         sprintf(temp,"%d",p);strcat(code1,temp);
         j=strlen(code1);
         code1[j]=ch;
         code1[j+1]='\0';
         printf("\n%s",code1);

         for(r=0;code1[r]!='\0';r++)
                if((code1[r]>='a')&&(code1[r]<='t'))
                         {
                          for(i=0;i<4;i++)

if(code1[r]==match5[i])
                                     {

binary3[c++]=match1[i][0];

binary3[c++]=match1[i][1];
                                     }
                          }
                else if((code1[r]>='A')&&(code1[r]<='R'))
                        {   for(i=0;i<3;i++)

if(code1[r]==match6[i])
                                     {

binary3[c++]='1';

binary3[c++]=match7[i][0];

binary3[c++]=match7[i][1];
                                     }
                         }
                  else
                        {
                         if(fl==0)
                         { binary3[c++]='0';fl=1;}
                         if(code1[r]=='8')
                         {

binary3[c++]='1';binary3[c++]='1';
                          binary3[c++]='1';
                         }
                         else
                         {
                          for(i=0;i<8;i++)
                                if(code1[r]==match2[i])
                                  {

binary3[c++]=match3[i][0];

binary3[c++]=match3[i][1];

binary3[c++]=match3[i][2];
                                          }
                                   }
                                 }
w=1;
}
else
{
 temp1[w-1]=str[z];
 w++;
}
binary3[c]='\0';
//clrscr();
printf("\n\n\tEncoded form in TWO BITS ENCODING:\n");
printf(" %s",binary1);
printf("\n\n\tEncoded form in EXACT MATCHING:\n");
printf(" %s",binary2);
printf("\n\n\tEncoded form in APPROXIMATE
MATCHING:\n");
printf(" %s",binary3);
len1=strlen(binary1);
len2=strlen(binary2);
len3=strlen(binary3);
if((len1<len2)&&(len1<len3))
  printf("\nMinimum no. of bits required to encode string:%s
in TWO BITS ENCODING \nis:%d",str,len1);
else if(len2<len3)
  printf("\nMinimum no. of bits required to encode string:%s
in EXACT MATCHING \nis:%d",str,len2);
else
  printf("\nMinimum no. of bits required to encode string:%s
in APPROXIMATE MATCHING \nis:%d",str,len3);
getch();

}
```

C-Program for Decompression of DNA Sequence

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
char bit[30],code[30];
int i,k,len,n,flag1=0,flag2=0,flag3=0;
printf("\nEnter the bit stream:");
gets(bit);
len=strlen(bit);
printf("\nEnter the methord:");
printf("\n1-For Two bits methord");
printf("\n2-For Exact Matching methord");
printf("\n3-For Approximate matching methord");
scanf("%d",&n);
switch(n)
{
case 1:
    for(i=0,k=0;i<len;i=i+2)
          if((bit[i]=='0')&&(bit[i+1]=='0'))
                   code[k++]='a';
          else if((bit[i]=='0')&&(bit[i+1]=='1'))
```

```
                code[k++]='c';
            else if((bit[i]=='1')&&(bit[i+1]=='0'))
                code[k++]='g';
                else
                code[k++]='t';

        code[k]='\0';
        break;
case 2:
for(i=0,k=0;i<len;)
{       if((bit[i]=='0')&&(flag1==0))
            {
            code[k++]='0';flag1=1;i=i+4;continue;
            }

if((bit[i]=='0')&&(bit[i+1]=='0')&&(bit[i+2]=='0')&&(flag2
==0))
            {
            code[k++]='0';flag2=1;i=i+4;continue;
            }


if((bit[i]=='0')&&(bit[i+1]=='0')&&(bit[i+2]=='1')&&(flag2
==0))
            {
            code[k++]='1';flag2=1;i=i+4;continue;
            }

if((bit[i]=='0')&&(bit[i+1]=='1')&&(bit[i+2]=='0')&&(flag2
==0))
            {
            code[k++]='2';flag2=1;i=i+4;continue;
            }

if((bit[i]=='0')&&(bit[i+1]=='1')&&(bit[i+2]=='1')&&(flag2
==0))
            {
            code[k++]='3';flag2=1;i=i+4;continue;
            }

if((bit[i]=='1')&&(bit[i+1]=='0')&&(bit[i+2]=='0')&&(flag2
==0))
            {
            code[k++]='4';flag2=1;i=i+4;continue;
            }

if((bit[i]=='1')&&(bit[i+1]=='0')&&(bit[i+2]=='1')&&(flag2
==0))
            {
            code[k++]='5';flag2=1;i=i+4;continue;
            }

if((bit[i]=='1')&&(bit[i+1]=='1')&&(bit[i+2]=='0')&&(flag2
==0))
            {
            code[k++]='6';flag2=1;i=i+4;continue;
            }

if((bit[i]=='1')&&(bit[i+1]=='1')&&(bit[i+2]=='1')&&(flag2
==0))
            {
            code[k++]='7';flag2=1;i=i+4;continue;
            }
    if((bit[i]=='0')&&(bit[i+1]=='0')&&(flag3==0))
    {
    code[k++]='a';i=i+3;flag3=1;flag2=0;continue;
    }

    if((bit[i]=='0')&&(bit[i+1]=='1')&&(flag3==0))
    {
    code[k++]='c';i=i+3;flag3=1;flag2=0;continue;
    }

    if((bit[i]=='1')&&(bit[i+1]=='0')&&(flag3==0))
    {
    code[k++]='g';i=i+3;flag3=1;flag2=0;continue;
    }

    if((bit[i]=='1')&&(bit[i+1]=='1')&&(flag3==0))
    {
    code[k++]='t';i=i+3;flag3=1;flag2=0;continue;
    }
    flag2=0; i=i-1;
}
code[k]='\0';
break;

case 3:
    for(i=0,k=0;i<len;)
{       if((bit[i]=='0')&&(flag1==0))
            {
            code[k++]='0';flag1=1;i=i+4;continue;
            }

if((bit[i]=='0')&&(bit[i+1]=='0')&&(bit[i+2]=='0')&&(flag2
==0))
            {
            code[k++]='0';flag2=1;i=i+4;continue;
            }


if((bit[i]=='0')&&(bit[i+1]=='0')&&(bit[i+2]=='1')&&(flag2
==0))
            {
            code[k++]='1';flag2=1;i=i+4;continue;
            }

if((bit[i]=='0')&&(bit[i+1]=='1')&&(bit[i+2]=='0')&&(flag2
==0))
            {
            code[k++]='2';flag2=1;i=i+4;continue;
            }

if((bit[i]=='0')&&(bit[i+1]=='1')&&(bit[i+2]=='1')&&(flag2
==0))
            {
            code[k++]='3';flag2=1;i=i+4;continue;
            }

if((bit[i]=='1')&&(bit[i+1]=='0')&&(bit[i+2]=='0')&&(flag2
==0))
            {
            code[k++]='4';flag2=1;i=i+4;continue;
            }
```

© JGRCS 2010, All Rights Reserved

```
if((bit[i]=='1')&&(bit[i+1]=='0')&&(bit[i+2]=='1')&&(flag2
==0))
                {
                code[k++]='5';flag2=1;i=i+4;continue;
                }

if((bit[i]=='1')&&(bit[i+1]=='1')&&(bit[i+2]=='0')&&(flag2
==0))
                {
                code[k++]='6';flag2=1;i=i+4;continue;
                }

if((bit[i]=='1')&&(bit[i+1]=='1')&&(bit[i+2]=='1')&&(flag2
==0))
                {
                code[k++]='7';flag2=1;i=i+4;continue;
                }
      if((bit[i]=='0')&&(bit[i+1]=='0')&&(flag3==0))
      {
      code[k++]='a';i=i+2;flag3=1;flag2=0;continue;
      }

      if((bit[i]=='0')&&(bit[i+1]=='1')&&(flag3==0))
      {
      code[k++]='c';i=i+2;flag3=1;flag2=0;continue;
      }

      if((bit[i]=='1')&&(bit[i+1]=='0')&&(flag3==0))
      {
      code[k++]='g';i=i+2;flag3=1;flag2=0;continue;
      }

      if((bit[i]=='1')&&(bit[i+1]=='1')&&(flag3==0))
      {
      code[k++]='t';i=i+2;flag3=1;flag2=0;continue;
      }
      flag3=0; i=i-1;
}
code[k]='\0';
break;
}

printf("\nstring is:%s",code);
getch();
}
```

Output is obtained using the proposed algorithms.

Figure 3: Original baboon image

Figure 4: Baboon after applying the proposed scheme

CONCLUSION

This paper introduced information hiding scheme based on DNA sequence technique. In this paper initially data has been converted into DNA sequence then compressed the DNA sequence .Four compression technique has been used, one of the option will produces better compression depending upon the DNA sequence. Also decompression algorithm has been developed to get the original data.

REFERENCES

[1] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, J.D. Watson, Molecular Biology of the Cell, Garland Publishing, New York & London, 1994.

[2] A. Apostolico, D. Breslauer, Z. Galil, Optimal parallel algorithms for periods, palindromes and squares, in: Proceedings of the International Colloquium on Automata, Languages, and Programming, 1992, pp. 296–307.

[3] A. Apostolico, D. Breslauer, Z. Galil, Parallel detection of all palindromes in a string, Theoretical Computer Science 141 (1995) 163–173.

[4] D. Breslauer, Z. Galil, Finding all periods and initial palindromes of a string, Algorithmica 14 (1995) 355–366.

[5] C.C. Chang, C.C. Lin, C.S. Tseng, W.L. Tai, Reversible hiding in DCT-based compressed images, Information Sciences 177 (2007) 2768–2786.

[6] C.C. Chang, T.C. Lu, Y.F. Chang, R.C.T. Lee, Reversible data hiding schemes for deoxyribonucleic acid (DNA) medium, International Journal of Innovative Computing, Information and Control 3 (2007) 1–16.

[7] C.C. Chang, W.C. Wu, Y.H. Chen, Joint coding and embedding techniques for multimedia images, Information Sciences 178 (2008) 3543–3556.

[8] C.T. Clelland, V. Risca, C. Bancroft, Hiding messages in DNA microdots, Nature 399 (1999) 533–534.

[9] M. Crochemore, W. Rytter, Jewels of Stringology, World Scientific, 2002.

© JGRCS 2010, All Rights Reserved

[10] European Bioinformatics Institute, <http://www.ebi.ac.uk/>.

© JGRCS 2010, All Rights Reserved