Persoalan Cryptarithmetic dengan Algoritma *Backtracking*

Elisa Sibarani[1], Inte Bu'ulolo[2], Rosni Lumbantoruan[3]

Program studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

E-mail : elisa@students.if.itb.ac.id[1], inte@students.if.itb.ac.id[2], rosni@students.if.itb.ac.id[3]

Abstrak

Persoalan cryptarithmetic termasuk salah satu persoalan CSP (*Constraint Satisfaction Problem*) yang bisa diselesaikan menggunakan algoritma *backtracking*. *Backtracking* adalah algoritma berbasis DFS (*Depth First Search*) untuk mencari solusi persoalan secara lebih mangkus. Pada algoritma *backtracking* tidak perlu memeriksa semua kemungkinan solusi yang ada, hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan. Dengan proses seperti ini maka perfomansi akan lebih baik karena akan menghemat waktu pencarian [1].

***Kata kunci**: cryptarithmetic, backtracking, DFS*

1. Pendahuluan

CSP merupakan teknik yang dapat digunakan untuk penyelesaian persoalan dalam dunia nyata, yang terkadang memberikan batasan tertentu yang tidak boleh dilanggar pada saat mencari solusinya. Pada saat melakukan pencarian solusi atau pemilihan urutan aksi, teknik penyelesaian ini akan selalu menyesuaikan dengan *constraint-constraint* yang sudah ditentukan sebelumnya, sedemikian sehingga semua *constraint* akan terpenuhi.

CSP dapat direpresentasikan sebagai berikut :

1. Kumpulan variabel.

   X → kumpulan dari n variabel X1,X2,X3,…..,Xn,

   Variabel adalah elemen atau *entity* yang ingin dicari nilainya sehingga pemberian nilai pada variabel dapat menjadi solusi dari CSP.
2. Domain D → Masing-masing variabel didefinisikan oleh suatu domain yang *finite* D1,D2,……,Dn, yang berisi kumpulan nilai-nilai yang mungkin untuk variabel tertentu yang bertujuan untuk menyelesaikan persoalan.
3. *Constraint* C → kumpulan dari *constraint-constraint* C1,C2,……,Cm.

   Ci merupakan *constraint-constraint* yang berisi batasan nilai untuk setiap variabel dan tidak boleh dilanggar. *Constraint-constraint* ini akan membatasi domain dari suatu variabel sehingga menjadi lebih sempit
4. *Assignment* → pemberian nilai pada suatu variabel.
5. Solusi → pemberian nilai-nilai pada setiap variabel yang memenuhi semua *constraint* yang telah ditetapkan untuk persoalan, sehingga nilai-nilai variabel tersebut merupakan solusi untuk persoalan.

Untuk mempermudah pemahaman terhadap suatu CSP, maka umumnya persoalan ini digambarkan dalam bentuk graf, dan disebut *constraint graph*. *Constraint graph* terdiri dari simpul – simpul yang merepresentasikan variabel pada CSP yang akan dicari nilainya, dan busur pada graf menggambarkan *constraint* di antara variabel yang dihubungkannya. Jika dua variabel tidak terhubung dengan suatu busur, maka di antara dua variabel tersebut tidak memiliki hubungan *constraint*, jadi tidak perlu diperiksa kekonsistenannya pada saat pemberian nilai.

Tujuan utama dari penyelesaian CSP yaitu untuk memberikan nilai pada semua komponen persoalan, dengan tidak melanggar *constraint* yang telah ditetapkan untuk persoalan tersebut.

2. Deskrispsi Masalah Cryptarithmetic

Cryptarithmetic merupakan pengetahuan dan seni untuk menciptakan dan menyelesaikan *mathematic puzzle*, dimana *digit-digit* ditukar dengan huruf-huruf alfabet atau symbol lain.

Cryptarithmetic merupakan salah satu contoh persoalan yang dapat diselesaikan dengan CSP, dengan *constraint* yang melibatkan 3 atau lebih variabel. *Cryptarithmetic* merupakan contoh yang sangat cocok untuk CSP, karena selain menyediakan deskripsi, masalah *cryptarithmetic* dapat dijelaskan lebih baik dengan *constraint-constraint.*

Constraint-constraint yang mendefinisikan sebuah *cryptarithmetic problem* antara lain :

1. Masing-masing huruf atau symbol merepresentasikan digit yang hanya satu dan unik dalam persoalan tersebut.
2. Ketika digit-digit menggantikan huruf atau simbol, maka hasil dari operasi aritmatika harus benar.

Batasan-batasan di atas memunculkan beberapa batasan baru dalam persoalan, yaitu apabila basis dari angka adalah 10, maka pasti akan ada paling banyak 10 simbol atau huruf yang unik dalam persoalan. Jika tidak, maka tidak akan mungkin untuk memberikan digit yang unik ke setiap huruf atau simbol yang unik juga dalam persoalan. Dan supaya memiliki makna yang berarti secara semantik, angka tidak boleh dimulai dengan 0, jadi huruf-huruf yang muncul untuk setiap variabel pertama sekali seharusnya tidak boleh mengandung 0.

Spesifikasi persoalan *cryptarithmetic (couple + couple = quartet)* yaitu:

1. Pemberian *digit* atau nilai harus berbeda untuk setiap variabel {C, O, U, P, L, E, Q, A, R, T} yaitu digit {0,…..9} sehingga persamaan COUPLE + COUPLE = QUARTET terpenuhi.
2. Apabila masing-masing variabel sudah diberikan nilai, maka pemberian nilai harus memenuhi *constraint* yang ada, sehingga pada saat operasi aritmatika dilakukan untuk nilai setiap variabel, maka hasil dari operasi penjumlahan COUPLE + COUPLE = QUARTET harus sesuai.
3. Variabel-variabel untuk persoalan cryptaritmetic ini ada 10 variabel, antara lain : C, O, U, P, L, E, Q, A, R, dan T.
4. Persoalan cryptaritmetic ini akan menggunakan *bit carry*, yaitu variabel X_1, X_2, X_3, X_4, dan X_5.

3. Prinsip Pencarian Solusi dengan Metode Runut Balik [1]

Disini kita hanya akan meninjau pencarian solusi pada pohon ruang status yang dibangun secara dinamis. Langkah-langkah pencarian solusi adalah sebagai berikut:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti metode pencarian mendalam (DFS). Simpul-simpul yang sudah dilahirkan disebut simpul hidup (*live node*). Simpul hidup yang sedang diperluas dinamakan simpul-E (*Expand node*). Simpul dinomori dari atas kebawah sesuai dengan urutan kelahirannya (ingat prinsip DFS).
2. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang. Jika lintasan yang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut "dibunuh" sehingga menjadi simpul mati (*dead node*). Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan fungsi pembatas (*bounding function*). Simpul yang sudah mati tidak akan pernah diperluas lagi.
3. Jika pembangunan lintaran berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan runut balik ke simpul terdekat (simpul orangtua). Selanjutnya simpul ini menjadi simpul-E yang baru. Lintasan baru dibangun kembali sampai lintasan tersebut membentuk solusi.
4. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut balik.

4. Penyelesaian Masalah dengan Algoritma *Backtracking* (Runut Balik)

Persoalan cryptarithmetic:
Diberikan sebuah bentuk cryptarithmetic dengan n buah variabel dan disediakan m buah angka. Berikan angka yang sesuai untuk setiap variabel, dan hasil aritmatika setelah setiap variabel diberi angka, harus sama dengan bentuk awal dan tidak ada variabel yang diberi angka yang sama.

Untuk persoalan ini, karena jumlah variabel ada 10, maka pastilah kesepuluh angka harus dipakai.

```
   X5  X4 X3 X2 X1
    C   O  U  P  L  E
    C   O  U  P  L  E
 ---------------------- +
 Q  U   A  R  T  E  T
```

Bit carry $X_1 = \{0,1\}$, $X_2 = \{0,1\}$, $X_3 = \{0,1\}$, $X_4 = \{0,1\}$, $X_5 = \{0,1\}$.

Constraint-contraint yang ditentukan untuk persoalan cryptarithmetic yaitu:

- E + E = T + 10 * X1
- X1 + L + L = E + 10 * X2
- X2 + P + P = T + 10 * X3
- X3 + U + U = R + 10 * X4
- X4 + O + O = A + 10 * X5
- X5 + C + C = U + 10 * Q

Solusi dinyatakan sebagai vektor X dengan n-tuple:

$X = (x_1, x_2, \ldots\ldots, x_n)$, $x_i \in \{0,1,2,\ldots..,9\}$

Angka variabel ke-*k*, x_k, ditentukan dengan algoritma berikut:

1. Bangkitkan angka *i*
2. Periksa pemberian angka berdasarkan *constraint* yang ada
3. Jika angka *i* yang dibangkitkan tidak melanggar *constraint* untuk variabel tersebut, maka variabel *k* diberi angka *i*, kalau tidak bangkitkan angka berikutnya, dan ulangi langkah 2 di atas.
4. Jika tidak ada lagi angka yang dapat dibangkitkan (angka habis), maka persoalan cryptarithmetic dengan *n* variabel dan *m* warna tidak dapat diselesaikan.

Algoritma Runut-balik untuk persoalan Cryptarithmetic

Masukan:

1. Matriks ketetanggaan GRAF [1…n, 1…n] {n = jumlah simpul graf}
   GRAF[i, j] = true
   jika ada *constraint* dari simpul i ke simpul j
   i untuk variabel yang akan dijumlahkan, bukan sebagai hasil, dan j adalah variabel hasil penjumlahan dari i
   GRAF[i, j] = false
   jika tidak ada *constraint* dari simpul i ke simpul j

   Karena penamaan simpul adalah angka, bukan variabel, sehingga untuk setiap variabel akan diberi urutan nomor dimulai dari variabel paling kanan dari cryptarithmetic yaitu E sampai ke paling kiri yaitu Q, sebab operasi yang dilakukan adalah penjumlahan sehingga harus dimulai dari sebelah kiri, yaitu:

E = 1	R = 6
T = 2	O = 7
L = 3	A = 8
P = 4	C = 9
U = 5	Q = 10

   Perlu diingat : pemberian nomor untuk variabel bukanlah solusi, tetapi hanya untuk mempermudah proses iterasi
2. Angka
   Dinyatakan dengan integer 0, 1, 2, ….., 9

Keluaran:

1. Tabel X [1.....n] yang dalam hal ini, x [i] adalah angka untuk simpul i

Algoritma:

```
1. Inisialisasi X [1.....n] dengan -1
      for i ← 1 to n do
            x [ i ] ← -1
      endfor
```

```
Procedure Cryptarithmetic (input S1: string, S2 : string S3 : string)
{
Deskripsi : Program utama
Masukan : tiga buah string.
Keluaran : jika solusi ditemukan, solusi cryptarithmetic di cetak ke piranti keluaran
Menggunakan:
  • function generate(input i : integer, j : integer output i : integer, j : integer)/*fungsi untuk mebangkitkan solusi per kolom operasi dan mengisikannya ke manyBlock, outputnya true jika solusi valid dan false jika solusi tidak valid*/
  • function cekLongLength(input S1:String, S2:String, S3:String)/*fungsi untuk mencari panjang string terbesar*/
  • procedure findFriend(input S1 : string, S2 : string, S3 : string, lengthGen : integer)/*procedure untuk mengenali karakter-karakter pada setiap kolom operasi dan menyimpannya ke charFriend*/
}
Deklarasi:
      lengthS1 : integer
      lengthS2 : integer
      lengthS3 : integer
      x      : integer
      y      : integer
      i      : integer

lengthGen : integer
/*mencatat panjang string terbesar untuk penentuan jumlah kolom operasi*/
typdef struct{
char1 : array of integer/*menyimpan nilai yang diassign ke operan pertama*/

char2 : array of integer/*menyimpan nilai yang diassign ke operan kedua*/
char3 : array of integer/*menyimpan nilai yang diassign ke hasil*/
parity : integer/*menyimpan nilai sisa*/
}block
  typdef struct{member : array of integer/*menyimpan karakter-karakter pada satu kolo*/
}friend

  manyBlock : array of struct block/*menampung solusi*/
  charFriend : array of friend/*mencatat semua kolom operasi*/
```

```
asignedNumber : array of
Boolean/*mencatat penggunaan angka
0-9*/
   arrayChar : array of
integer/*mencatat angka yang di
assign ke*/
/*setiap karakter misalnya untuk*/
/*karakter A, arrayChar[65]=5*/


Algoritma:
lengthGen=cekLongLength(S1, S2, S3)
findFriend(S1, S2, S3, lengthGen)

for i=0 to banyakKarakter
arrayChar[karakter yang terlibat] ←
NULL
end for
       x=0
       y=0
       while x < lengthGen do
         while (y< 10 and not stop)
do
              if(!assignedNumber[y])
              result=generate(&x,&y);
                 if result==true
                     {increased x;
                      y ← 0}
                else
if(x!=0)
{x= cekBacktrack(manyBlock[x].char1,
manyBlock[x].char2,
manyBlock[x].char3)

angka yang sudah di assign pada
karakter yang bersesuaian di beri
status false tapi tidak boleh di
assign pada karakter yang sama}
       increased y
              end if
              end if
              end while
end while
```

```
Function generate(input i : integer,
j : integer output i : integer, j :
integer)
{
Deskripsi : fungsi untuk
mebangkitkan solusi per kolom
operasi dan mengisikannya ke
manyBlock, outputnya true jika
solusi valid dan false jika solusi
tidak valid
Masukan : nomor kolom operasi(i) dan
salah satu angka dari 0-9 yang akan
di assign.
Keluaran : Boolean true atau false
}

Deklarasi:
       jj : integer
       temp1 : integer
       temp2 : integer
       temp3 : integer
       parity : integer
       toParity : integer

Algoritma:
if( kolom operasi pertama )
 if(karakter pertama dan kedua
 sama){
 manyBlock[i].char1 ← j
 manyBlock[i].char2 ← j
 manyBlock[i].char3 ←
 manyBlock[i].char1+
 manyBlock[i].char2;

 if(jika angka yang diasgn ke
 karakter pertama dan kedua sama)
       return false
 else
       /*assign angka ke karakter-
 karakter*/
       arrayChar[charFriend[i].member
 [0] ] ← j
 arrayChar[charFriend[i].member[2] ]
 ←  manyBlock[i].char3

 /*status angka yang diassign
 menjadi true*/
       assignedNumber[j]← true
       assignedNumber[manyBlock[i].ch
 ar3]←true

 if(manyBlock[i].char3 >9)
        manyBlock[i].parity ← 1
        else
        manyBlock[i].parity ← 0
       }
 else/*karakter pertama dan kedua
 berbeda*/{
 manyBlock[i].char1 ← j;
 assignedNumber[j] ← true
 jj ← j
 j ← j+1
 manyBlock[i].char2 ← j;
 manyBlock[i].char3 ←
 manyBlock[i].char1+
 manyBlock[i].char2;
 if(angka yang diassign ke setiap
 karakter ada yang sama)
       return false
 else
       arrayChar[charFriend[i].member
 [0]]← jj
        arrayChar[charFriend[i].membe
 r[0]] ← j
 arrayChar[charFriend[i].member[2]
 ]← manyBlock[i].char3
       assignedNumber[j]← true
       assignedNumber[manyBlock[i].ch
 ar3]← true

if(manyBlock[i].char3 >9)
       manyBlock[i].parity ← 1
       else
       manyBlock[i].parity ← 0
}
else/*kolom kedua dan seterusnya*/{
if(karakter pertama dan kedua sama){
```

```
{manyBlock[i].char1 ← j;
manyBlock[i].char2 ← j;}
            else
{manyBlock[i].char1 ← temp1;
manyBlock[i].char2 ← temp1;}

if(temp3 == NULL){
{toParity ← parity +
manyBlock[i].char1+
manyBlock[i].char2

      manyBlock[i].char3 ← toParity
mod 10
       if (assignedNumber
[manyBlock[i].char3]){
i ← cekBacktrack
(manyBlock[i].char1,
manyBlock[i].char2,manyBlock[i].cha
r3)
       return false}
             }
             else{
if(jumlah sisa + karakter pertama
dan kedua tidak sama dengan angka
yang diassign pada karakter ke
tiga)
                          return
false
                  else

      manyBlock[i].char3 ← Temp3
                          toParity
← manyBlock[i].
       }

if(jika angka yang diasgn ke
karakter pertama dan kedua sama){
i ← cekBacktrack
(manyBlock[i].char1,
manyBlock[i].char2,manyBlock[i].cha
r3)
      return false}
if(toParity>9)
       manyBlock[i].parity ← 1
       else
       manyBlock[i].parity ← 0
      }
else/*karakter pertama dan kedua
berbeda*/{
      temp1 ← arrayChar[
[charFriend[i].member[0]]]
       temp2 ← arrayChar[
[charFriend[i].member[1]]]
       temp3 ← arrayChar[
[charFriend[i].member[2]]]

      parity ← manyBlock[i-].parity
      if(temp1 == NULL)/*jika
      karakter belum diassign
      angka*/
```

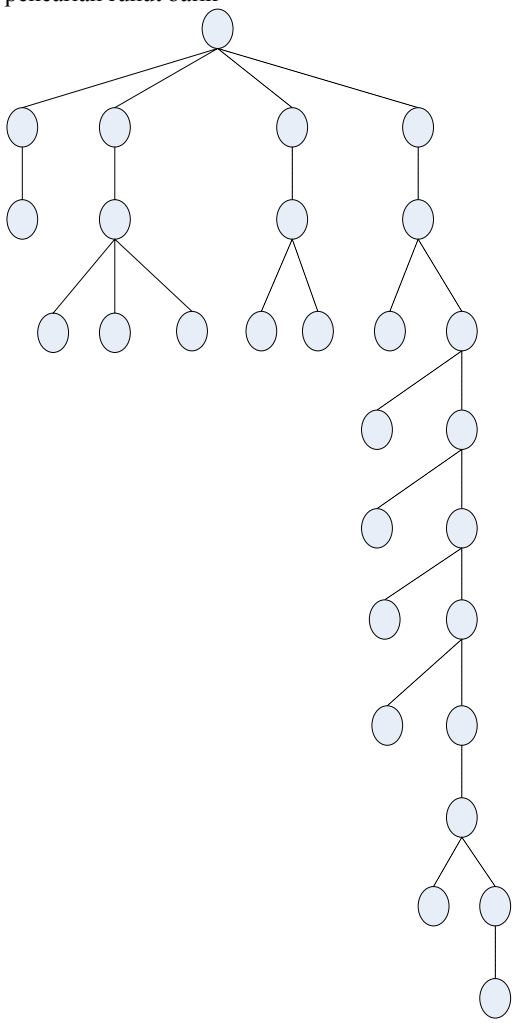
```
manyBlock[i].char1 ← j;
            if(temp21 == NULL)
            {jj ← j
       j ← j+1
       manyBlock[i].char2 ← j}

if(temp3 == NULL){
{toParity ← parity +
manyBlock[i].char1+
manyBlock[i].char2

     manyBlock[i].char3 ← toParity
mod 10

/*angka yang diassign ke karakter
ke tiga berstatus true maka
backtrack*/
       if
(assignedNumber[manyBlock[i].char3]
){
i ← cekBacktrack
(manyBlock[i].char1,
manyBlock[i].char2,manyBlock[i].cha
r3)
 return false}
} else{
if(jumlah sisa + karakter pertama
dan kedua tidak sama dengan angka
yang diassign pada karakter ke
tiga)!=temp3)
{return false}
  manyBlock[i].char3 ← Temp3
  toParity ← manyBlock[i].
}

if (angka yang diassign ke setiap
karakter ada yang sama)
i ← cekBacktrack
(manyBlock[i].char1,
manyBlock[i].char2,manyBlock[i].cha
r3)
     return false
if(toParity>9)
       manyBlock[i].parity ← 1
       else
       manyBlock[i].parity ← 0
}
return true
```

```
Procedure findFriend(input S1 :
string, S2 : string, S3 : string,
lengthGen : integer)
{
  Deskripsi : procedure untuk
  mengenali karakter-karakter pada
  setiap kolom operasi dan
  menyimpannya ke charFriend
  Masukan : tiga buah string dan
  panjang banyak kolom operasi
  Keluaran : charFriend
}
```

```
Deklarasi :
j : integer
lengthS1 : integer
lengthS2 : integer
lengthS3 : integer

lengthS1 ← length of string S1 - 1;
lengthS2 ← length of string S2 - 1;
lengthS3 ← length of string S3 - 1;

Algoritma :
for i ← 0 to lengthGen-1
j ← 0
if (lengthS1 != -1)
{charFriend[0].member[j] ←
 S1[lengthS1]}
else
{charFriend[0].member[j] ← NULL}
     j ← j + 1
     if (lengthS2 != -1)
            {charFriend[0].member[j]
            ← S2[lengthS2]}
       else
            {charFriend[0].member[j]
              ← NULL}

     j ← j + 1
     if (lengthS1 != -1)
            {charFriend[0].member[j]
              ← S3[lengthS3]}
       else
            {charFriend[0].member[j]
              ← NULL}
     lengthS1 ← lengthS1-1
     lengthS2 ← lengthS1-1
     lengthS3 ← lengthS1-1
end for
```

```
Function cekBacktrack (input char1 :
character, char2 : character , char3
: character)
{
 Deskripsi : fungsi untuk melakukan
 bactracking ke kolom operasi
 sebelumnya yaitu kolom operasi yang
 telah menggunakan salah beberapa
 dari karakter input
 Masukan : tiga buah karakter yang
 berasal dari satu kolom operasi
 yang tidak valid
 Keluaran : nomor kolom operasi
 hasil backtracking
}

Deklarasi :
       i : integer
       j : integer

for i ← 0 to lengthGen
       for j= ← 0 to 2
              if(char1 ==
charFriend[i].member[0])
              return i
       if(char2 ==
charFriend[i].member[1])
return i
              if(char2 ==
charFriend[i].member[2])
                     return i
       end for
end for
```

Pohon ruang status dinamis yang dibentuk selama pencarian runut balik



5. Referensi

1. Rinaldi Munir, *Diktat Kuliah IF2251 Strategi Algoritmik*, STEI, 2006.
2. Stuart J Russell & Peter Norvig, *Artificial Intelligence – A Modern Approach*, Prentice-Hall International, Inc, Textbook