PENYELESAIAN PERMAINAN SUDOKU DENGAN ALGORITMA *BRUTEFORCE*,*BACKTRACKING*,dan *BACKTRACKING* DENGAN OPTIMASI

Rama Adhitia-NIM:13505040

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl.Ganesha No.10
if15040@students.if.itb.ac.id

ABSTRAK

Isi Makalah ini membahas tentang algoritma-algoritma yang dapat digunakan untuk menyelesaikan permainan sudoku, salah satu permainan logika yang sangat populer dewasa ini.Terdapat tiga buah macam metode penyelesaianyang akan dibahas dalam makalah ini, yaitu penyelesaian menggunakan algoritma *Bruteforce*,penyelesaian menggunakan algoritma *Backtracking*,dan penyelesaian menggunakan algoritma *Backtracking* dengan optimasi. Ketiga penyelesaian akan dijelaskan dengan membedah detail algoritmanya.

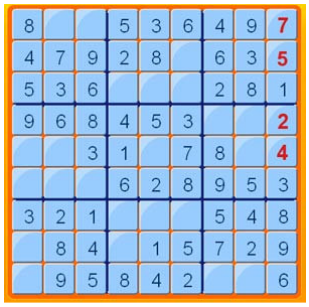
Algoritma *Bruteforce* ialah algoritma yang membangkitkan seluruh kemungkinan penempatan angka pada papan sudoku,kemudian dari seluruh kemungkinan penempatan angka itu akan dicari himpunan-himpunan penempatan angka mana yang akan memenuhi *constraint* permainan sudoku.Algoritma *Backtracking* sebenarnya mempunyai *kompleksitas asimptotik* yang sama dengan algoritma *Bruteforce*,hanya saja algoritma ini mencari secara sistematis solusi persoalan diantara semua kemungkina solusi yang ada,sehingga pada kebanyakan kasus waktu pencarian solusi permainan sudoku dapat dapat dihemat secara signifikan. Pada Algoritma *Backtracking* dengan optimasi terdapat beberapa modifikasi kecil pada cara untuk membangkitkan komponen berikut dari vektor solusi dan pada cara runut baliknya.Sebagai akibatnya pada sebagian besar kasus,algoritma ini dapat mencari solusi permainan sudoku dengan waktu lebih cepat lagi jika dibandingkan dengan algoritma *Backtracking* biasa.

Kata kunci:
Sudoku,Bruteforce,Backtracking,*Backtracking* dengan optimasi,optimasi,

1. PENDAHULUAN

Sudoku merupakan salah satu permainan logika yang sangat populer. Permainan sudoku dapat digolongkan ke dalam tipe *Latin Square,* dengan tambahan *constraint* pada isi dari setiap *region* yang terpisah.[1]Leonhard Euler terkadang dianggap sebagai sumber dari permainan *puzzle* ini didasarkan pada hasil kerjanya yaitu *Latin Square*

Puzzle modern ditemukan oleh seseorang berkebangsaan Amerika Serikat ,Howard Garns,pada tahun 1979 dan diterbitkan pada majalah Dell dengan sebutan "Number Place".Permainan ini berkembang menjadi permainan populer di Jepang ketika dipromosikan oleh Nikoli dan diberi nama Sudoku.Kepopuleran permainan Sudoku ini meledak secara luar biasa di dunia pada tahun 2005



Gambar 1.Salah satu contoh permainan sudoku

Objektif permainan sudoku sebenarnya sangat sederhana,yaitu mengisi suatu matriks yang berukuran 9x9 sehingga pada setiap kolom ,setiap baris,dan setiap dari kotak yang berukuran 3x3 mengandung digit angka dari 1 sampai 9,dan pada setiap baris,kolom,dan kotak yang berukuran 3x3 tersebut tidak terdapat dua atau lebih kemunculan angka yang sama.Daya tarik dari permainan sudoku ini ialah aturan permaina yang sederhana,tetapi penalaran yang dibutuhkan untuk

menyelesaikan permainan ini dapat saja berubah menjadi kompleks.

Karena adanya alasan seperti yang telah disebutkan di atas maka banyak orang yang tidak bisa melanjutkan lagi permainannya menggunakan program-program komputer yang didesain khusus untuk menyelesaikan permainan sudoku. Oleh karena itu timbullah kebutuhan akan adanya algoritma yang efektif dan efisien untuk menyelesaikan permainan sudoku ini.

2.Isi

2.1 Algoritma *Bruteforce*

Algoritma *Bruteforce* merupakan algoritma yang memecahkan persoalan dengan cara yang *straightforward*.Secara garis besar cara kerja algoritma ini ialah dengan membangkitkan semua permutasi dari komponen vektor solusi,kemudian setelah dibangkitkan maka akan diperiksa himpunan mana yang merupakan solusi dari sebuah persoalan

Penerapan algoritma *Bruteforce* ini dalam permainan sudoku ialah dengan mencoba seluruh kemungkinan isi kotak elemen dari matriks. Misalkan solusi dari suatu permainan sudoku dinyatakan sebagai:

$X=(x_1,x_2,x_{3,\ldots,}x_9^{81})$;

Dari vektor solusi diatas dapat kita simpulkan bahwa dengan menggunakan algoritma *Bruteforce* kita harus membangkitkan seluruh kemungkinan vektor solusi yang berjumlah $9^{81}=1{,}99\text{x}10^{77}$ kemungkinan.

Pseudocode dari algoritma *Bruteforce* ini berupa loop berkalang sebanyak 81 buah, dimana di setiap loop di enumerasi seluruh kemungkinan angka yang dapat dimiliki oleh setiap elemen matriks sudoku

```
     x1,x2,x3,x4,x5,x6,x7,x8,x9,…    :
integer
     for(x1=1 to 9)
     {
      for(x2=1 to 9)
      {
       for(x3=1 to 9)
       {
        for(x4=1 to 9)
        {
         for(x5=1 to 9)
       {
     //semua loop kalang den
       for(x81=1 to 9)
          {
           if
(Solusi(x1,x2,x3,x4,x5,x6,x7,x8,x9))
         {
         }
      }
      }
}
}
```

Gambar 2.*Psedocode* algoritma *Bruteforce*

Kompleksitas asimptotik dari algoritma *Bruteforce* diatas,dilihat dari banyaknya operasi perbandingan dilakukan, dapat disimpulkan bernilai $O(n^n)$ yang termasuk kategori *non polynomial complexity*,sehingga penggunaan algoritma ini sangatlah tidak menguntungkan jika dilihat dari waktu dan banyaknya komputasi yang dilakukan oleh komputer.

Dari hasil yang telah kita dapatkan diatas maka ,dibutuhkan algoritma lain yang lebih mangkus daripada algoritma *Bruteforce* diatas untuk menyelesaikan masalah permainan sudoku

2.2 Algoritma *Backtracking*

[2]Algoritma *Backtracking* merupakan algoritma yang berbasiskan pada algoritma DFS (*Depth First Search*) untuk mencari solusi persoalan lebih mangkus.Kalau dilihat secara lebih teliti,algoritma *Backtracking* ini sebenarnya merupakan modifikasi algoritma *Bruteforce*.Modifikasi ini menyebabkan algoritma *Backtracking* dapat secara sistematis mencari solusi persoalan tanpa harus memeriksa seluruh kemungkian solusi yang ada.Hanya pencarian yang mengarah ke solusi saja yang dipertimbangkan oleh algoritma ini,sedangkan pencarian yang tidak mengarah ke sebuah solusi tidak dilanjutkan.Hal ini menyebabkan waktu komputasi algoritma *Backtracking* jauh lebih baik.

Algoritma *Backtracking* membentuk sebuah pohon ruang status selama prosesnya Struktur pohon inilah, yang juga merupakan sebuah graf tak berarah,yang ditraversal dengan prinsip DFS(Depth First Search).Simpul-simpul pada pohon ruang status yang tidak mengarah ke solusi maka akan "dimatikan".Sedangkan simpul-simpul pohon ruang status yang masih mengarah ke solusi maka akan terus berkembang .Pematian simpul pohon ruang status yang tidak mengarah kepada solusi ini sering disebut dengan istilah *prunning.*

Penerapan algoritma *Backtracking* pada permainan sudoku adalah sebagai berikut :

1. Algoritma dimulai pada elemen matris baris ke-satu dan kolom ke satu
2. Periksa seluruh kemungkinan angka yang dapat dimiliki oleh baris tersebut
3. Jika terdapat angka yang valid dengan *constraint* permainan sudoku maka lanjutkan pemeriksaan ke elemen selanjutnya dari matriks
4. Jika tidak terdapat angka yang valid maka *backtrack* ke elemen matriks sebelumya
5. Algoritma ini akan berhenti jika sudah ditemukan suatu solusi atau jika tidak terdapat kemungkinan adanya solusi

Pseudocode algoritma *Backtracking* ini dapat dilihat lebih lanjut pada gambar di bawah :

```
Procedure            SolveSudoku(input
array:matriks)

Deklarasi
   solusi:boolean
   i,j:integer
Algoritma
1.solusi = false;
2.i=1;
3.j=1;
 4.while (!solusi && i > 0)
 5.{
 6,array[i][j].SetValue(array[i][j].G
etValue() + 1);}
 7.while(array[i][j].GetValue()<=9&&
8.!Tempat(i, j))
   9.{
10.array[i][j].SetValue(array[i][j].G
11.etValue() + 1);
 12. }
   13.if  (array[i][j].GetValue()  <=
14.maks)
 15. {
 16.  if (i == 9 && j == 9)
 17.  {
 18.  solusi = true;
 19.   }
 20.  else
 21.  {
 22.  j = j + 1;
 23.  if (j > 9)
 24.  {
 25.              i = i + 1;
 26.              j = 1;
 27.        }
 28.  }
 29. }
 30. else
 31. {
 32.  array[i][j].SetValue(0);
 33.  j--;
 34.  if (j == 0)
 35.  {
 36.  j = kolom;
 37.  i = i - 1;
 38. }
 39.}
 40.}
```

Gambar 3. *Psedocode* algoritma *Backtracking*

Pada baris 2 dan 3 dari gambar pseucode di atas dapat kita lihat inisialisasi variabel i dan j dengan 1,kemudian pernyelesaian permainan sudoku dimulai pada baris ke-4 dengan *looping* yang mempunyai kondisi berhenti jika sudah ditemukan suatu solusi atau variabel I bernilai 0 yang artinya tidak dapat ditemukan solusi.Kemudian pada baris ke-10 pseudocode dicoba seluruh kemungkinan angka yang dapat dimiliki oleh elemen matriks baris i dan kolom j.Pada baris ke-8 pseudocode terdapat fungsi Tempat yang berfungsi sebagai validasi apakah angka yang sedang dicoba tidak melanggar *constraint* permainan sudoku. Setelah melewati blok kode nomor 7 sampai 12,akan diperiksa lagi apakah angka yang dimiliki oleh elemen matriks baris i dan kolom j tersebut melebihi dari batas atau tidak.Jika melebihi batas maka akan dilakukan backtrack ke elemen matriks sebelumnya sebagaimana yang terlihat pada blok kode nomor 30 sampai 37.Jika angka pada elemen matriks baris i kolom j tidak melebihi batas maka algoritma ini akan melanjutkan pencarian solusi ke elemen berikutnya pada matriks.

Dapat disimpulkan bahwa dengan mematikan simpul pohon ruang status (*prunning*) algoritma *Backtracking* dapat menghemat waktu komputasi secara signifikan,sehingga merupakan salah satu algoritma alternatif yang dapat diterapkan untuk menyelesaikan permainan sudoku dengan bantuan komputer

2.3 Algoritma *Backtracking* dengan optimasi

Sebagaimana dengan algoritma *Backtracking* yang telah dibahas pada subbab sebelumnya,algoritma *Backtracking* dengan optimasi ini juga mengandalkan metode *prunning* dalam cara kerjanya.Yang membedakan algoritma ini dengan algoritma *Backtracking* biasa ialah cara pemilihan elemen

matriks selanjutnya dan cara pemilihan elemen matriks sebelumnya yang ingin di-*backtrack*.

Cara pemilihan elemen matriks pada algoritma *Backtracking* dengan optimasi ini tidaklah secara linier seperti algoritma *Backtracking* biasa,melainkan dengan mencari elemen matriks yang memiliki probabilitas terkecil.Probabilitas dalam konteks ini maksudnya ialah banyaknya kemungkinan angka pada suatu elemen matriks yang dapat memenuhi *constraint* sudoku dimulai dari angka yang sekarang oleh elemen matriks ini Oleh karena itu pemilihan elemen matriks berikutnya yang ingin dicari solusinya bisa saja bersifat acak tidak bersifat linier

Algoritma *Backtracking* dengan optimasi membutuhkan tipe data baru alokasi memori tambahan untuk melaksanakan kerjanya.Detail dari Tipe data baru yang bernama BackTrackWithOpt ini dapat dilihat pada *listing pseudocode* di bawah ini.

```
class BacktrackWithOpt
    {
        private int indexX;
        private int indexY;
        private int value;
        private int prob;
        private int urutan;

        public   BacktrackWithOpt(int
x, int y, int val)
        {
            indexX = x;
            indexY = y;
            value = val;
            urutan = -22;
        }
```

Gambar 4. Deklarasi tipe data *BacktrackWithOpt*

Property indexY berfungsi untuk menyimpan indeks kolom dari elemen matriks berhubungan.Properti index berfungsi untuk menyimpan indeks baris dari elemen matriks yang bersangkutan pula.Properti value berfungsi untuk menyimpan nilai dari elemen matriks yang bersangkutan.Properti prob untuk menyimpan nilai probabilitas elemen matriks yang bersangkutan,dan yang terakhir properti urutan untuk menyimpan data urutan suatu elemen matriks akan dievaluasi

Alokasi memori tambahan yang telah disebutkan di atas tadi ialah suatu *array* yang bertipe BacktrackWithOpt yang mempunyai banyak elemen sejumlah elemen matriks .Tiap-tiap elemen dari matriks ini berkorespondensi dengan elemen matriks yang bersesuaian.

Penerapan algoritma *Backtracking* dengan optimasi pada permainan sudoku adalah sebagai berikut :

1. Buat array dengan tipe BacktrackWithOpt yang banyak elemennya sejumlah dengan banyak elemen matriks .
2. Alokasi setiap elemen array ini dengan value ,indeks baris,dan indeks kolom dari elemen matriks yang bersesuaian .Properti urutan pada masing-masing elemen tabel diset dengan nilai -22(nilai ini tidak harus bernilai -22 ,tetapi dapat dipilih nilai lain yang dapat berfungsi sebagai mark).
3. Hitung setiap probabilitas elemen –elemen tabel dengan fungsi countprob(nanti akan dijelaskan pada sourcecode).
4. Proses pencarian solusi dimulai dari elemen tabel yang mempunyai nilai probabilitas yang paling kecil.
5. Periksa seluruh kemungkinan angka (value) yang dapat dimiliki oleh elemen tabel tersebut
6. Jika tidak terdapat angka yang valid maka *backtrack* ke elemen tabel sebelumnya dengan melihat urutan elemen tabel yang diproses sebelumnya,ulangi langkah 5
7. Jika terdapat angka valid maka cari elemen selanjutnya dari tabel yang yang mempunyai nilai probabilitas terkecil,ulangi langkah 5
8. Algoritma ini akan berhenti jika sudah ditemukan suatu solusi atau jika tidak terdapat kemungkinan adanya solusi

Pseudocode algoritma *Backtracking* dengan optimasi ini dapat dilihat lebih lanjut pada gambar di bawah :

```
Procedure      SolveSudoku      dengan
optimasi(input
array:matriks,tabel:array)

Deklarasi :
i,h,c,panjangarr : integer
solusi,valid : Boolean
Algoritma
i=0;j=0;c=0;panjangarr=0;
for(c=0 to 8) do
{
   for(j=0 to 8) do
   {
      if(matriks[i][j]=-99)then
       {
```

```
         tabel[panjangarr]    =    new
BackTrack(c,j,0);
         panjangarr++;
      }
   }
}

for(int x=0 to 8) do
{
   if(tabel[x].getUrutan =-22) then
   {
tabel[x].setProb(countProb(tabel[x].g
    etIndexX,tabel[x].getIndexY))
   }

}

while(tabel[h].getProb()!= minProb())
{
     h++;
 }

tabel[h].setProb(999);
i++;
tabel[h].setUrutan(i);

while (!solusi && i > 0)
{
  if (tabel[h].getValue() == 0)
  {
   tabel[h].setValue(tabel[h].getValu
e() + 1);
   matriks[tabel[h].getIndexX()][tabe
l[h].getIndexY()]=tabel[h].getValue()
;

  if(validate(tabel[h].getValue(),
tabel[h].getIndexX(),
tabel[h].getIndexY()))
 {
     valid = true;
 }
   else
 {
     valid = false;
 }
}
}


while(tabel[h].getValue()<=side&&
!valid)
 {
  tabel[h].setValue(tabel[h].getValu
e() + 1);
matriks[tabel[h].getIndexX()][tabel[h
].getIndexY()] = tabel[h].getValue();

if (validate(tabel[h].getValue(),
tabel[h].getIndexX(),
tabel[h].getIndexY()))
{
   valid = true;
}
else
{
   valid = false;
}
}
if (tabel[h].getValue() > side)
{
   tabel[h].setValue(0);
   matriks[tabel[h].getIndexX()][tabe
l[h].getIndexY()] = 0;
   matriks[tabel[h].getIndexX()][tabe
l[h].getIndexY()].Text = "";
   tabel[h].setUrutan(-22);
   i--;
   h = 0;
   while ((h < panjangArr) &&
(tabel[h].getUrutan() != i))
   {
     h++;
   }
   valid = false;
}


else
{
   if (i == panjangArr)
   {
     solusi = true;
   }
   else
   {
   for (int x =0;x<panjangArr;x++)
   {
     if (tabel[x].getUrutan() == -22)
     {
tabel[x].setProb(countProb(tabel[x].g
etIndexX(),tabel[x].getIndexY()));
      }
     }
   h = 0;
 while(tabel[h].getProb()!=minProb())
 {
   h++;
 }
     tabel[h].setProb(999);
     i++;
```

```
      tabel[h].setUrutan(i);
      }
}
}
```

Gambar5. *Psedocode* algoritma *Backtracking* dengan optimasi

3 .Analisis dan kesimpulan

Ketiga algoritma yang telah dibahas di atas mampu menyelesaikan permainan Sudoku dengan baik.Tetapi pertanyaan yang akan timbul ialah manakah algoritma yang terbaik.Untuk menjawab pertanyaan ini,haruslah diadakan analisis terhadap algoritma-algoritma di atas tersebut

Algoritma *Bruteforce* ini sebagaimana yang telah kita ketahui,menemukan solusi permainan Sudoku ini dengan membangkitkan seluruh kemungkinan vektor solusi .Setelah kemungkinan vektor solusi dibangkitkan barulah dicari vektor solusi mana saja yang memenuhi *constraint.*Kelebihan algoritma *Bruteforce* ini terletak dari cara pikir yang *straightforward* dan lugu,tidak adanya trik-trik atau metode yang canggih pada algoritma ini menyebabkan biasanya algoritma ini yang langsung terpikir untuk menghadapi masalah permainan Sudoku ini.
Tetapi ternyata algoritma ini mempunyai kelemahan yang mencolok ,yaitu dari segi komputasi yang dilakukan,dan waktu yang dibutuhkan untuk melakukan komputasi tersebut.
Bila kita teliti lebih lanjut pada kasus permainan sudoku ini kita harus membangkitkan calon vektor solusi sebanyak 9^{81} .Jika kita generalisasi permasalahan untuk permainan sudoku dengan ukuran papan permainan NxN maka harus dibangkitkan N pangkat N kuadrat solusi.Oleh karena itu algoritma *Backtracking* bukan merupakan solusi yang efektif dan efisien untuk menyelesaikan permainan sudoku

Algoritma *Backtracking* merupakan perbaikan dari algoritma *Bruteforce*.Algoritma ini mempunyai property yang dinamakan fungsi pembatas,Fungsi pembatas menentukan apakah (x1,x2,x3,….,xk) mengarah ke suatu solusi jika ya,maka pembangkitan xk+1 dilanjutkan jika tidak maka backtrack ke komponen xk-1.Dalam hal permainan sudoku ini xk+1 dianggap sebagai elemen berikut nya pada matriks,dan xk-1 merupakan elemen sebelumnya dalam matriksWalaupun dalam kasus permainan sudoku ini kompleksitas asimptotik dari algoritma *Backtracking*,fungsi pembatas dapat melakukan pruning yang menyebabkan waktu komputasi berkurang secara signifikan.

Skema yang dijalankan oleh runut-balik dengan optimasi secara umum mengutamakan pada pemilihan kotak yang memiliki kemungkinan angka yang sesuai dengan fungsi batas yang berjumlah paling sedikit. Runut-balik dengan optimasi ini memang memiliki lebih banyak kalang dan variabel untuk penyimpanan sementara, namun karena jenis kalang yang dipakai adalah while maka lebih besar kemungkinan program untuk mendapatkan solusi lebih cepat dan lebih dulu keluar dari kalang while dibandingkan skema runut-balik tanpa optimasi.

Kelemahan dari skema dengan optimasi terdapat pada penggunaan memori yang lebih besar dibandingkan skema runut-balik tanpa optimasi. Hal ini berakibat lebih lambatnya pencarian solusi pada kotak Sudoku yang berukuran besar seperti 16 x 16 (Hexudoku).

Menurut Tahir Arazi dan R Aditya Satria,dua orang yang mengimplementasikan algoritma ini pada program sudoku *solver* nya, algoritma *Backtracking* dengan optimasi selalu melakukan *traversal* dari awal hingga akhir matriks, sehingga untuk ukuran kotak yang lebih besar, akan terjadi perbedaan waktu yang lebih signifikan dibandingkan pendekatan runut-balik tanpa optimasi.

Setelah membandingkan ketiga algoritma di atas,makanya algoritma *Backtracking* dan *Backtracking* dengan optimasi merupakan solusi yang lebih efisien dibandingkan dengan algorima *Bruteforce*.

REFERENSI

[1] www.wikipedia.org. Tanggal akses 18 Mei 2007
[2]Munir,Rinaldi.(2007).Diktat Kuliah IF2251 Strategi Algoritmik.Program Studi Teknik Informatika