PERUMUSAN SUDOKU DENGAN SOLUSI UNIK MENGGUNAKAN ALGORITMA PENCABANGAN DAN PEMBATASAN *(BRANCH AND BOUND)*

Umi Fadilah Wardati Syam

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
e-mail: if15037@students.if.itb.ac.id

ABSTRAK

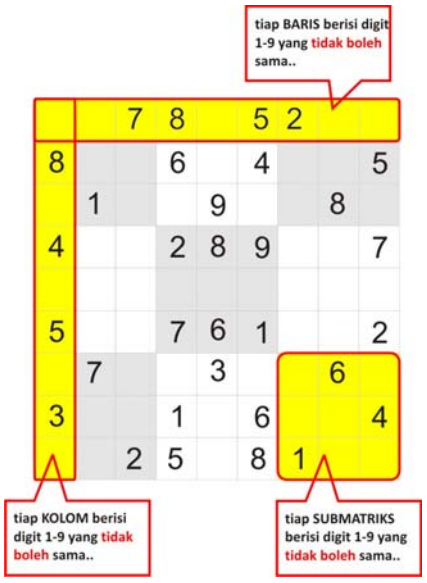
Sudoku adalah permainan yang sangat populer saat ini, memicu berlomba-lombanya *programmer* untuk menciptakan algoritma untuk menyelesaikan teka-teki ini dengan waktu sesingkat mungkin (optimasi persoalan). Namun dalam makalah ini, akan diterangkan bagaimana cara merumuskan angka-angka awal sehingga menghasilkan sebuah teka-teki Sudoku yang benar-benar unik. Maksudnya disini adalah, Sudoku yang tepat memiliki hanya satu buah solusi, sehingga tidak ada jenis teka-teki yang memiliki banyak solusi, maupun sama sekali tidak memiliki solusi. Perumusan soal *(generating puzzle procces)* Sudoku kali ini dicoba dengan menerapkan algoritma Pencabangan dan Pembatasan *(Branch and Bound)* dengan menggunakan interval data dalam proses minimasi dan penentuan batasan bawah *(lower bound)* deviasi maksimum persoalan. Algoritma Pencabangan dan Pembatasan merupakan algoritma pencarian di dalam ruang solusi secara sistematis, sehingga sangat efektif dan efisien untuk diterapkan dalam permasalahan ini.

Kata kunci: *Branch and Bound*, Sudoku.

1. PENDAHULUAN

Sudoku adalah sebuah permainan teka-teki angka berbasis logika yang pertama kali didesain oleh seorang arsitek berkebangsaan Amerika Serikat, Howard Garns, pada tahun 1979. Barulah kemudian permainan ini menjadi populer di Jepang. Kata "Sudoku" sendiri merupakan singkatan dari sebuah frase kalimat dalam bahasa Jepang, *"suji wa dokushin ni kagiru"*, yang berarti "setiap digit harus tetap satu jumlahnya".

Sudoku paling umum berbentuk matriks 9 x 9 (n^2*n^2 dengan $n=3$). Aturan permainannya sederhana, isi semua matriks sampai penuh, dengan catatan, untuk setiap kolom, baris, maupun submatriks berukuran 3 x 3 *(n x n)* hanya boleh terisi dengan angka 1 sampai 9 yang berjumlah masing-masing satu. Sesuai namanya, maka tidak boleh ada angka/digit yang sama. Hal inilah yang dinamakan dengan "Prinsip Keunikan" *(Principle of Uniqueness)*.



Gambar 1. Contoh Sudoku berukuran 9 x 9, berdasarkan prinsip keunikan

Kesulitan tingkat permainan Sudoku, selain diukur dari besar ukuran matriksnya (n^2*n^2), biasanya diukur dari problem perumusan angka-angka awal yang telah diatur posisi dan nilainya. Semakin sedikit angka-angka awal yang diberikan, tentunya akan semakin sulit.

Sebenarnya ada berbagai macam metode yang dapat digunakan untuk perumusan angka-angka awal pada Sudoku, mulai dari *Brute Force*, namun yang tersulit adalah menemukan algoritma yang dapat menghasilkan sebuah teka-teki Sudoku yang tepat memiliki hanya satu solusi, dan juga memiliki kompleksitas yang tidak terlalu besar. Oleh karena itu akan digunakan pendekatan lain,

yaitu melalui penerapan algoritma Pencabangan dan Pembatasan *(Branch and Bound)*.

2. ALGORITMA PENCABANGAN DAN PEMBATASAN *(BRANCH AND BOUND ALGORITHM)*

2.1. Prinsip Umum Algoritma

Algoritma Pencabangan dan Pembatasan *(Branch and Bound)*, pada umumnya hampir serupa dengan algoritma Runut Balik *(Backtracking)*. Perbedaan paling utamanya ada pada pembentukan pohon ruang status yang digunakan untuk mencari ruang solusi. Pada algoritma Runut Balik ruang solusi dibangun secara dinamis berbasis Pencarian Mendalam atau DFS *(Depth First Search)*, sedangkan algoritma Pencabangan dan Pembatasan membangun ruang solusi berdasarkan skema Pencarian Melebar atau BFS *(Breadth First Search)*.

Tiap simpul diberi sebuah *cost* (nilai ongkos) yang dihitung oleh suatu fungsi pembatas, dilambangkan dengan ĉ(i), yang menyatakan perkiraan ongkos termurah lintasan dari simpul ke-i menuju simpul solusi *(goal node)*. Sehingga skema BFS yang digunakan sedikit diubah pada proses ekspansi simpulnya, yang semula diekspansi berdasarkan urutan pembangkitannya (umumnya melebar ke samping kanan), menjadi berdasarkan simpul dengan nilai ongkos terkecil *(least cost search)*.

Lalu untuk menghitung taksiran atau perkiraan nilai, fungsi secara umum dinyatakan sebagai berikut:

$$\hat{c}(i) = f(i) + \hat{g}(i)$$

yang dalam hal ini,
$\hat{c}(i)$ = ongkos untuk simpul i
$f(i)$ = ongkos mencapai simpul i dari akar
$\hat{g}(i)$ = ongkos mencapai simpul tujuan dari simpul akar i

Jika perumusan angka-angka soal pada Sudoku ini diterapkan menggunakan algoritma Pencabangan dan Pembatasan, maka dibutuhkan sebuah interval data yang menjadi batas bawah *(lower bound)* dalam penentuan ruang solusi. Interval data tersebut akan menjadi nilai ongkos ĉ(i). Nilai ĉ tersebut digunakan untuk mengurutkan pencarian. Simpul berikutnya yang akan diekspansi adalah simpul dengan ĉ minimum.

3. SOLUSI ALGORITMA

3.1. Metode Perumusan Menggunakan *Brute Force*

Jika menggunakan metode *Brute Force*, cukup gunakan ide berikut:

Isi matriks berukuran misalnya 9 x 9, dengan integer 1 sampai 9 secara acak (random), lalu periksa apakah hasilnya memenuhi tiga aturan dasar Sudoku (berkaitan dengan baris, kolom, maupun submatriks). Pendekatan ini akan menghasilkan kurang lebih $9^{81} \approx 1.97 \times 10^{77}$ matriks yang butuh untuk diperiksa ke tahap kedua. Berdasarkan komputasi lojik, akan didapatkan kombinatorial sebanyak $6.670.903.752.021.072.936.960 \approx 6{,}67 \times 10^{21}$, atau setara dengan $9! \times 72^2 \times 2^7 \times 27.704.267.971$ *(Integer Programming Model for Sudoku, page 6)*.

Tentunya sangat tidak mangkus dan sangkil mengingat hanya beberapa diantaranya yang layak *(feasible)*, dan belum tentu tepat satu solusi, bisa beragam solusi, bahkan mungkin tidak terdapat solusi. Hal ini tidak mencerminkan prinsip keunikan *(principle of uniqueness)* yang terkandung dalam pengertian Sudoku. Dan paling penting, kompleksitas waktu yaang dibutuhkan jika menggunakan algoritma *Brute Force* tentunya akan sangat besar.

3.2. Metode Perumusan Menggunakan *Branch and Bound*

Dalam persoalan perumusan angka-angka awal Sudoku menggunakan algoritma Pencabangan dan Pembatasan, kita akan membagi menjadi dua langkah/prosedur utama.

Langkah pertama, kita mengisi grid atau submatriks dengan acak *(random)*, namun berdasarkan fungsi pembatas. Hal ini berarti dalam proses pengisian kotak-kotak angka, kita tetap berpegang pada prinsip keunikan Sudoku tersebut, bahwa tidak boleh ada digit yang sama dalam setiap kolom, baris, maupun submatriks *(n x n)* nya.

Langkah kedua, hapus angka-angka yang terbentuk satu per satu, juga secara acak. Jika setelah ada satu angka dihapus dan soal tersebut telah memiliki solusi unik, maka ekspansi ke simpul berikutnya. Jika tidak, keluarkan simpul tersebut alias di-*kill* (gantikan angka pada posisi aslinya), dan hapus angka lain (ekspansi ke simpul berikutnya). Dengan cara ini, perumusan soal Sudoku dapat mencegah kemungkinan banyak solusi. Keuntungan dari metode ini adalah tidak akan ada digit yang terperangkap dalam kalang tak hingga *(infinite loop)*, sehingga pasti dapat menciptakan ruang solusi.

Sekarang akan dibahas lebih jauh lagi mengenai pendekatan algoritma Pencabangan dan Pembatasan *(Branch and Bound)*. Ukuran matriks yang digunakan dalam pendekatan ini adalah Sudoku secara umum, yaitu n=3, sehingga didapat matriks berukuran 9 x 9. Ukuran matriks Sudoku bisa fleksibel, asalkan merupakan kuadrat dari integer. Metode algoritma ini dibagi menjadi tiga, satu untuk menciptakan grid keseluruhan, lalu untuk menciptakan kotak-kotak kecil (submatriks), dan yang terakhir adalah untuk mengecek apakah angka yang di-*generate* memiliki solusi atau tidak.

3.2.1.Algoritma membuat grid keseluruhan

```
//algoritma buatGridKomplit()
function buatGridKomplit(Output: boolean)

//Proses inisialisasi grid kosong
Kamus :
  kolomKosong = integer[1..9];
// list of kolom yang masih kosong di
// kolom keseluruhan matriks

  isiKotak = integer[1..9];
// list of kotak-kotak yang terisi pada
// baris

Algoritma :
for (digit=1 to n) {
 integer repetisi = 0;
 clear isiKotak;
 for (baris=1 to n, baris+=2) {
   repetisi++;
   if (repetisi == 150) then
        return false;
   clear kolomKosong;

   Inisialisai isiKotak dengan nilai 1-n

   //cari posisi di baris,untuk tempat
   //meletakkan current digit
   boolean ulangi=true;
    while (ulangi) {
     ulangi=false;
     if (isEmpty(kolomKosong)) then
       hapus kotak lain yang berisi digit
       dan mulai lagi dari baris=0;
     else {
       pilih secara acak dari array
       kolomKosong dan ambil nilai kolom
       tersebut, kolom=kol;

       if (matriks[baris,kolom]!=empty or
       isExist(digit)) then
         hapus (row,col) dari
         kolomKosong;
         ulangi = true;

         if (ulangi) then
          continue;

         jika
     }//end else
    }//end while

    //jika keluar dari loop while dan
    //mulai dari baris awal lagi
    if (ulangi) then continue;
     matriks[baris,kolom]=digit;
     add (baris,kolom) ke dalam isiKotak
  } //endfor loop2
} //endfor loop1

return true;
```

Pada algoritma tersebut, program berjalan dalam *infinite loop*, atau membutuhkan lebih banyak waktu ketika dilakukan pengecekan mulai dari baris paling atas, dalam urutan yang kontinu 1,2,3,4,… . Penjelasan yang paling intuitif adalah bahwa algoritma ini berjalan berdasarkan seleksi angka acak/*random*. Angka-angka acak tersebut akan lebih berhasil ketika angka-angka tersebut didistribusikan secara merata di keseluruhan himpunan solusi. Ketika indeks baris dinaikkan satu *(increment)*, semua nomor terisikan di baris-baris awal, (contohnya 1,2,3,4, dan 5) dan ada kemungkinan bahwa tidak ada angka valid tersisa untuk baris-baris selanjutnya. Akan tetapi jika kelima baris ini adalah 1,3,5,7, dan 9, maka akan ada domain yang lebih besar bagi fungsi acak untuk menentukan angka selanjutnya bagi baris 2,4,6,8. Maka dari itu, di dalam algoritma yang tertera diatas, baris diubah urutannya menjadi 1,3,5,7,9,2,4,6,8 (tiap *increment* baris, selalu baris+=2).

3.2.2.Algoritma untuk membuat grid soal

```
//algoritma buatGridSoal()
function buatGridSoal(Output: boolean)

Kamus :
  k = integer;
// k adalah nomor kotak kecil yang ingin
// dihapus

  points = array of list[1..n^2];
// array of list of Point sebesar ukuran
// matriks(n^2), mengandung koordinat
// kotak yang bisa dihapus

for (count = 0 to k) {
 if (isEmpty(points))  then
   return false;

   Pilih kotak secara acak, dan hapus
angkanya;

   Cari solusinya dengan menggunakan
algoritma penyelesaian teka-teki (panggil
function punyaSolusi());

if (!solusi exist dan unik) then
        count--;
        gantikan angka yang telah
terhapus;
        hapus kotak dari koordinat
points;

        continue;
   }
  Hapus kotak dari points;
  }
}

return true;
```

3.2.3. Algoritma untuk mengecek apakah Sudoku memiliki solusi atau tidak

```
//algoritma buatGridKomplit()
function punyaSolusi(Input: gridProses:
array of list of integer[][],returnArray:
array of list of integer[][], adaSolusi:
boolean)

Kamus :
   arraySolusi[][] = array of list of
                  integer[1..16][1..16]

Algoritma :
for (baris=1 to 9) {
 for (kolom=1 to 9) {
   arraySolusi[][]=
         gridProses[baris][kolom];
 }
}
boolean complete = false;
boolean bisa = true;
boolean pilihan = false;

while (complete==false) do {
   if (bisa==false and pilihan==true)
         then return bisa;
   else if(bisa==false and pilihan==true)
         then pilihan=true;
}
bisa = false;
complete = true;
kandidat = array of integer[1..16];
Inisialisai isiKotak dengan nilai 1-n

if (arraySolusi !kosong) then
   Hapus kandidat;
   if (digit kandidat==1) then
         bisa =true;
         arraySolusi[baris][kolom]=
                  kandidat[0];
   else if (digit kandidat==2 and
pilihan=true) then
         arraySolusi[baris][kolom]=
                  kandidat[0];
         if (punyaSolusi(arraySolusi, ref
arraySolusi, false))then
            arraySolusi[baris][kolom]=
                  kandidat[1];
            if (punyaSolusi(arraySolusi,
ref arraySolusi, false))then
                  return false;
                  //solusi tidak unik
            arraySolusi[baris][kolom]=
                  kandidat[0];

            pilihan = false;
            bisa = true;
         else {
            arraySolusi[baris][kolom]=
                  kandidat[1];
            if (punyaSolusi(arraySolusi,
ref arraySolusi, true))then
                  pilihan = false;
                  bisa = true;
            else {
                  arraySolusi[baris][kolom
                  ]= noNumber;
            } //end else
         } //end else
   } //end else

if (adaSolusi) then
   returnArray=arraySolusi;
   return true;
```

IV. KESIMPULAN

Ada beberapa cara yang dapat dilakukan untuk menyusun angka-angka soal pada permainan Sudoku, namun hanya sedikit yang mampu menghasilkan teka-teki Sudoku yang memiliki solusi unik. Unik disini berarti Sudoku tersebut hanya memiliki satu solusi saja.

Salah satu algoritma yang mangkus dan sangkil untuk persoalan ini adalah dengan algoritma Pencabangan dan Pembatasan (Branch and Bound). Dalam permasalahan ini, algoritma B&B dipakai untuk menentukan cost dari tiap simpul dengan menimbang apakah pada baris, kolom, atau submatriks utama yang sudah tebentuk apakah proses pengacakan angka dan posisinya telah memenuhi prinsip keunikan atau belum. Jika tidak, maka anak simpulnya tidak akan dibangkitkan alias di-*kill*.

Penerapan algoritma B&B ini lebih efektif dan kompleksitasnya lebih rendah dibandingkan dengan algoritma *Brute Force* yang mencoba semua kemungkinan, seperti yang telah dijelaskan di atas.

REFERENSI

[1] Barlett, Andrew and Langville, Amy, *An Integer Programming Model for the Sudoku Problem*, 2006

[2] Munir, Rinaldi, *Strategi Algoritmik,* Bandung, 2007.

[3] http://www.codeprojects.com/