ISSN: 2089-1121

ANALISIS PENYELESAIAN PUZZLE SUDOKU DENGAN MENERAPKAN ALGORITMA BACKTRACKING

Rina Dewi Indah Sari
Magister Teknologi Informasi
Sekolah Tinggi Teknik Surabaya
rideinsar30@gmail.com

ABSTRAK

Algoritma yang banyak diterapkan dalam mengaplikasikan permainan adalah algoritma backtracking (runut-balik). Dengan menggunakan algoritma ini, penyelesaian suatu permainan, yang melibatkan banyak kemungkinan, dapat diselesaikan dengan lebih cepat. Hal ini dikarenakan, jika kita menggunakan algoritma backtracking, tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang perlu dipertimbangkan.

Salah satu jenis permainan yang dapat diselesaikan dengan algoritma runut-balik adalah permainan teka-teki Sudoku. Permainan Sudoku adalah salah satu puzzle yang paling banyak digemari saat ini, dan juga merupakan salah satu permasalahan paling sulit di bidang informatika. Hingga saat ini banyak programmer yang mencari algoritma yang paling mangkus untuk menyelesaikan puzzle ini.

Kata kunci: Sudoku, Puzzle, Algoritma Backtracking

ABSTRACT

The algorithm is many applications in applying the game is a backtracking algorithm (trace-back). By using this algorithm, the completion of a game, which involves a lot of possibilities, can be completed more quickly. This is because, if we use a backtracking algorithm, no need to examine all possible solutions exist. Only search that leads to solutions that need to be considered.

One type of game that can be resolved with the trace-back algorithm is a puzzle game Sudoku. Sudoku puzzle is one of the most popular today, and also one of the most difficult problems in the field of informatics. Until today many programmers who are looking for the most better algorithm to solve this puzzle.

Keywords: Sudoku, Puzzles, Backtracking Algorithm

1. PENDAHULUAN

Permainan *Sudoku* adalah permainan yang dapat melatih logika manusia dalam berpikir cepat dan teliti. Permainan ini tidak bisa sembarang dimainkan, karena bila bermain dengan sembarangan di awal permainan, tidak bisa menyelesaikan game ini.

Permasalahan *puzzle Sudoku* sulit untuk dipecahkan karena masuk dalam permasalahan *NP-complete*, sehingga tidak bisa diselesaikan dalam waktu yang sama. Hingga saat ini banyak programmer yang mencari algoritma yang tepat untuk menyelesaikan *puzzle* ini. Cara yang paling gampang adalah algoritma *Brute Force*

ISSN: 2089-1121

yaitu dengan cara mengenumerasikan semua kemungkinan isi sel dengan angka 1 sampai 9. Tetapi cara ini tentu saja tidak tepat karena kemungkinannya akan sangat banyak sekali. Karena itu algoritma ini diperbaiki dengan menambahkan batasan (*constraints*), yaitu tidak boleh ada angka yang sama dalam satu baris, kolom atau *subgrid*. Cara ini bisa mereduksi jumlah kemungkinan secara signifikan sehingga algoritma menjadi lebih tepat.

Untuk memecahkan teka-teki *Sudoku*, dapat digunakan algoritma *backtracking* (runut-balik). Algoritma ini merupakan perbaikan dari algoritma *Brute Force*, dimana solusi dapat ditemukan dengan penelusuran yang lebih sedikit dan dapat mencari solusi permasalahan secara lebih efektif karena tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang perlu dipertimbangkan.

Algoritma *Backtracking* ini mudah diimplementasikan dengan bahasa pemrograman yang mendukung pemanggilan fungsi/ prosedur rekursif. Salah satu bahasa pemrograman yang mendukung pemanggilan fungsi adalah Visual Basic 6.0.

2. TINJAUAN PUSTAKA

2.1 Dynamic Problem Solving

Dynamic Problem Solving dapat diartikan sebagai penyelesaian/ pemecahan masalah di mana *state* (kondisi) permasalahan tersebut selalu berubah-ubah dan mempunyai banyak kemungkinan solusi. Perubahan tersebut bisa terjadi dengan mengikuti suatu pola (*model*) tertentu yang dapat didefinisikan sebagai suatu fungsi terhadap waktu, maupun tanpa pola.

Prinsip dasar dari *Dynamic Problem Solving* ini adalah menganalisa semua jalur yang bisa menghasilkan solusi dan kemudian memilih satu jalur terbaik sehingga didapatkan solusi terbaik. Dalam beberapa kasus, program tidak hanya akan melihat/ menganalisa langkah-langkah yang ada sebagai pertimbangan dalam memilih solusi, contohnya pada beberapa *board games* seperti catur dan *sheckers*, karena besar kemungkinannya bahwa persoalan tersebut memiliki solusi yang sangat banyak. Apabila semua solusi tersebut ditelusuri satu per satu, maka program tidak akan berjalan dengan efisien.

2.2 Aturan Permainan Sudoku

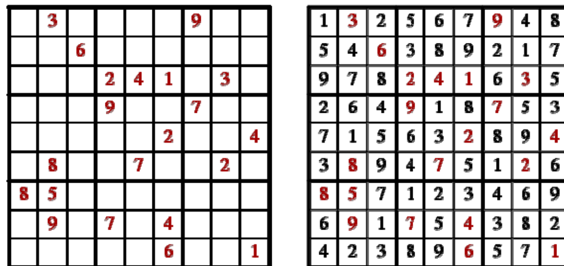
Aturan permainan untuk *puzzle* ini sangat sederhana, untuk menyelesaikan permainan ini tidak diperlukan pengetahuan umum, kepandaian atas bahasa tertentu, juga kemampuan matematika. Tetapi hanya memerlukan kecermatan, kesabaran, dan logika.

Papan *Sudoku* terbuat dari sembilan buah kotak oiberukuran 3×3 (disebut blok/ *subgrid*) yang disusun sedemikian rupa sehingga menghasilkan kotak besar berukuran 9×9. Beberapa kotak sudah diisi sebagai petunjuk awal dan tugas pemain adalah melengkapi angka-angka pada kotak yang lain sehingga keseluruhan papan permainan terisi angka secara lengkap. Aturan permainannya sangatlah sederhana:

1. Kotak-kotak pada setiap baris, kolom, dan blok/ *subgrid* harus berisi sebuah angka.
2. Angka-angka yang diisikan harus unik dari 1 hingga 9 sehingga dalam 1 blok/ *subgrid* hanya terdiri atas angka 1-9 yang tidak berulang dan tidak ada angka yang berulang dalam 1 baris maupun kolom.

ISSN: 2089-1121

Angka-angka ini sebenarnya tidak memiliki hubungan aritmetis satu sama lain. Anda boleh menggantinya dengan 9 huruf, lambang, atau warna yang berbeda.

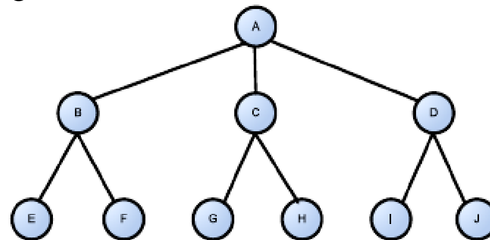


Gambar 1. Puzzle Sudoku

2.3 Algoritma Backtracking (Runut Balik)

a. Algoritma Umum *Backtracking*

Algoritma *backtracking* adalah suatu algoritma yang merupakan perbaikan dari algoritma *brute force*, secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada[(6)]. *Backtracking* merupakan bentuk tipikal dari algoritma rekursif dan berbasis pada DFS dalam mencari solusi yang tepat. Selain itu, algoritma ini juga merupakan metode yang mencoba-coba beberapa keputusan sampai kita menemukan salah satu yang "berjalan". Kita tidak perlu memeriksa semua kemungkinan solusi yang ada, tetapi cukup yang mengarah kepada solusi saja. Dengan memangkas (*pruning*) simpul-simpul yang tidak mengarah ke solusi, sehingga waktu pencarian dapat dihemat. Algoritma ini banyak diterapkan untuk program games dan permasalahan pada bidang kecerdasan buatan.



Gambar 2. Pohon Solusi (*tree*)

Saat ini algoritma *backtracking* banyak diterapkan untuk program games seperti permainan tic-tac-toe, menemukan jalan keluar dalam sebuah labirin, catur dan sebagainya serta untuk menyelesaikan masalah-masalah pada bidang kecerdasan buatan (*artificial intelligence*).

Prinsip dasar algoritma *backtracking* adalah mencoba semua kemungkinan solusi yang ada. Perbedaan dengan algoritma *brute force* adalah pada konsep dasarnya, yaitu pada *backtracking* semua solusi dibuat dalam bentuk pohon solusi (*tree*), dan kemudian pohon tersebut akan ditelusuri secara DFS sehingga ditemukan solusi terbaik yang diinginkan.

Misalkan pohon di atas menggambarkan solusi dari suatu persoalan. Jika kita ingin mencari solusi dari A ke E, maka jalur yang harus ditempuh adalah (A-B-E). Demikian juga untuk solusi-solusi yang lain. Algoritma *backtracking* akan memeriksa jalur secara DFS, yaitu dari solusi terdalam pertama yang ditemui yaitu solusi E. Jika ternyata E bukanlah solusi yang diharapkan, maka pencarian akan dilanjutkan ke F. Jalur yang harus dilalui untuk bisa mencapai E adalah (A-B-E) dan untuk mencapai F

ISSN: 2089-1121

adalah (A-B-F). Kedua solusi tersebut memiliki jalur awal yang sama, yaitu (A-B). Jadi, dari pada memeriksa ulang jalur dari A kemudian B, maka jalur (A-B) disimpan dulu dan langsung memeriksa solusi F. Untuk kasus pohon yang lebih rumit, cara ini dianggap lebih efisien daripada jika menggunakan algoritma *Brute-Force*.

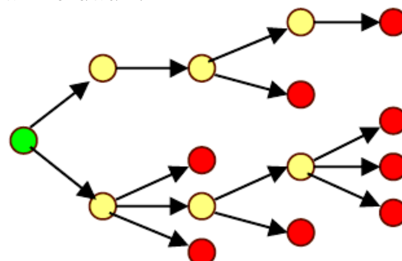
b. Properti Umum Metode Runut Balik (*Backtracking*)

1. Solusi persoalan
   Solusi dinyatakan sebagai vektor dengan *n*-index:
   $$X = (x_1, x_2, \ldots, x_n),\ \ x_i \in \text{himpunan berhingga } S_i$$
   Mungkin saja $S_1 = S_2 = \ldots = S_n$
   Keterangan: X = vektor solusi
   x = komponen vektor solusi
   S = himpunan kemungkinan solusi
   Contoh: $S_i = \{0, 1\}$, $x_i = 0$ atau 1
2. Fungsi pembangkit nilai x_k
   Dinyatakan sebagai:
   $$T(k)$$
   $T(k)$ membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.
   Keterangan: x = komponen vektor solusi
   k = index komponen vektor solusi
   T = fungsi pembangkit
3. Fungsi pembatas
   Pada beberapa persoalan fungsi ini dinamakan fungsi kriteria.
   Dinyatakan sebagai $B(x_1, x_2, \ldots, x_k)$
   Fungsi pembatas menentukan apakah $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi. *B* bernilai *true* jika $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi.
   Jika *true*, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika *false*, maka $(x_1, x_2, \ldots, x_k)$ dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi.
   Keterangan: x = komponen vektor solusi
   k = index komponen vektor solusi
   B = fungsi pembatas

3. PEMBAHASAN

3.1 Prinsip Pencarian Solusi dengan Metode Backtracking

Seperti yang telah dijelaskan bahwa pencarian solusi dengan menggunakan algoritma backtracking digunakan pohon ruang status. Cara kerjanya adalah dengan membentuk lintasan dari akar ke daun.



Gambar 3. Pohon Ruang Status

ISSN: 2089-1121

Langkah-langkah pencarian solusi pada pohon ruang status yang dibangun secara dinamis:

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 dinamakan simpul hidup (*live node*). Simpul hidup yang sedang diperluas dinamakan simpul-E (*Expand-node*). Simpul dinomori dari atas ke bawah sesuai dengan urutan kelahirannya.
2. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang. Jika lintasan yang sedang 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 pembentukan lintasan 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 *backtracking* ke simpul hidup terdekat (simpul orang tua). 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 simpul hidup untuk *backtracking* atau simpul yang dapat di diperluas.

3.2 Skema Umum Algoritma Backtracking

a. Versi rekursif

Skema versi rekursif sebagai berikut:

```
procedure RunutBalikR(input k:integer)
{Mencari semua solusi persoalan dengan metode runut-balik; skema rekursif
 Masukan: k, yaitu indeks komponen vektor solusi, x[k]
 Keluaran: solusi x = (x[1], x[2], …, x[n])
}
Algoritma:
   for tiap x[k] yang belum dicoba sedemikian sehingga
        ( x[k]←T(k)) and B(x[1], x[2], ... ,x[k])= true do
      if (x[1], x[2], ... ,x[k]) adalah lintasan dari akar ke daun
      then
         CetakSolusi(x)
      endif
      RunutBalikR(k+1)    { tentukan nilai untuk x[k+1]}
   endfor
```

b. Versi iteratif

ISSN: 2089-1121

Skema versi iteratif sebagai berikut:

```
procedure RunutBalikI(input n:integer)
{Mencari semua solusi persoalan dengan metode runut-balik; skema iteratif.
 Masukan: n, yaitu panjang vektor solusi
 Keluaran: solusi x = (x[1], x[2], …, x[n])
}
Delarasi:
   k : integer

Algoritma:
   k←1
   while k > 0 do
      if (x[k] belum dicoba sedemikian sehingga x[k]←T(k)) and
           (B(x[1], x[2], ... ,x[k])= true)
      then
         if (x[1],x[2],...,x[k]) adalah lintasan dari akar ke daun
         then
            CetakSolusi(x)
         endif
          k←k+1     {indeks anggota tupple berikutnya}
      else   {x[1], x[2], …, x[k] tidak mengarah ke simpul solusi }
         k←k-1     {runut-balik ke anggota tupple sebelumnya}
      endif
   endwhile
  { k = 0 }
```

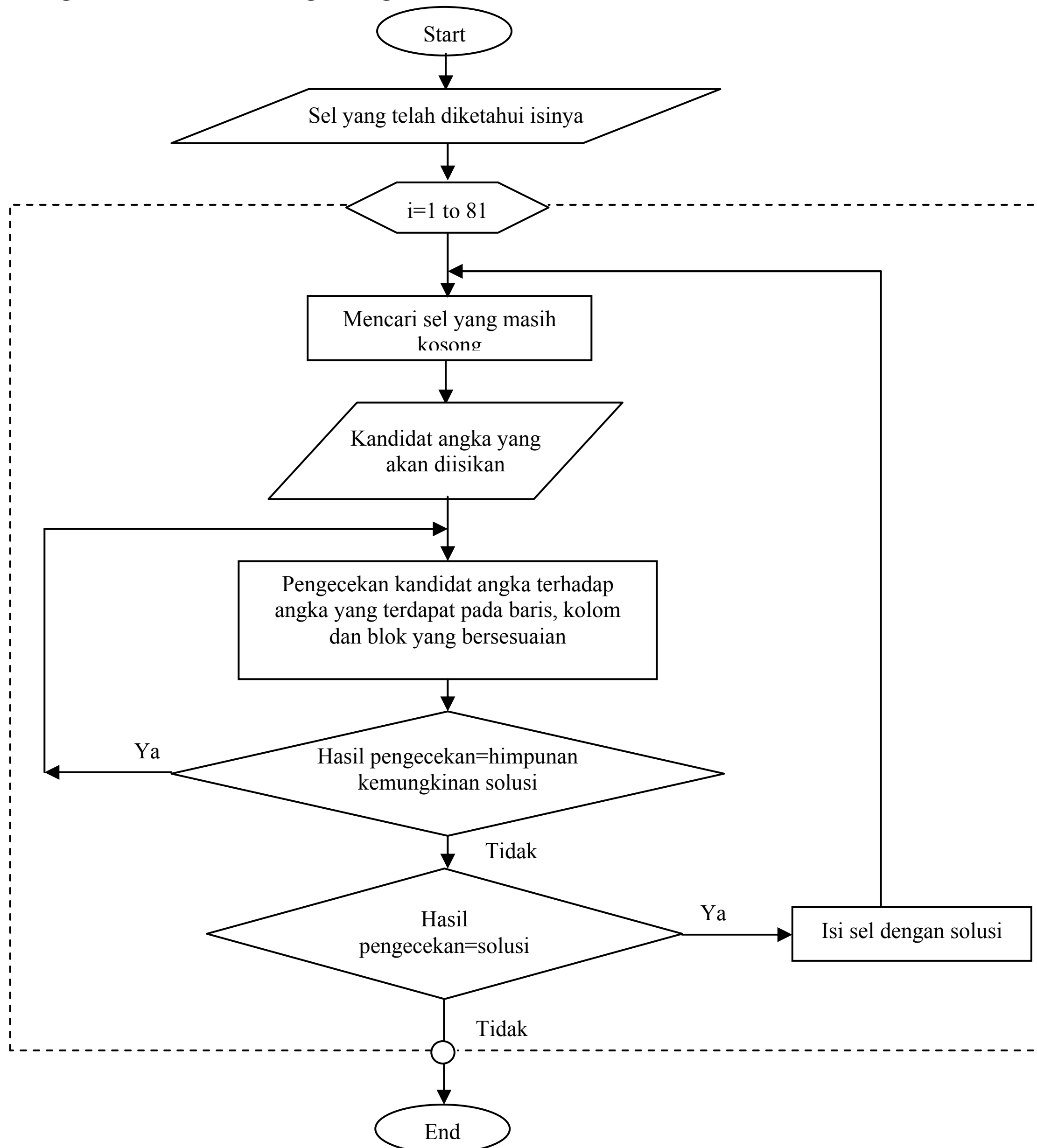
Setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemanggilan rekursif. Jika jumlah simpul dalam pohon ruang status adalah 2^n atau $n!$, maka untuk kasus terburuk, algoritma *backtracking* membutuhkan waktu dalam $O(p(n)2^n)$ atau $O(q(n)n!)$, dengan $p(n)$ dan $q(n)$ adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul.

3.3 Pengorganisasian Solusi

1. Ruang Solusi (*solution space*)
   Semua kemungkinan solusi dari persoalan dinyatakan sebagai
   vektor $X = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$, $x_i \in S_i$
   $S_1 = S_2 = S_3 = S_4 = S_5 = S_6 = S_7 = S_8 = S_9 = ... = S_{81}$
   $S_i = \{1,2,3,4,5,6,7,8,9\}$, x_i = 1 II 2 II 3 II 4 II 5 II 6 II 7 II 8 II 9
   Maka, ruang solusi = $S_1 \times S_2 \times S_3 \times S_4 \times S_5 \times S_6 \times S_7 \times S_8 \times S_9 \times ...... \times S_{81}$
2. Fungsi Pembangkit
   Dinyatakan sebagai *T(k)* yang membangkitkan nilai untuk *x(k)*, yang merupakan bagian himpunan solusi.
3. Fungsi Pembatas (*bounding space*)
   Baris dinyatakan sebagai $B(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$
   Kolom dinyatakan sebagai $K(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$
   Blok/ *Subgrid* dinyatakan sebagai $G(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$
   Fungsi pembatas =
   $$\sum_{i=1}^{k} B_i K_i G_i = 1+2+3+4+5+6+7+8+9$$

ISSN: 2089-1121

3.4 Algoritma Backtracking sebagai Pemecahan Solusi Permasalahan Puzzle Sudoku



Gambar 4. Diagram Blok Pencarian Solusi

Penggunaan algoritma *backtracking* ini akan terlihat dalam proses pengisian sel dengan sebuah angka dimana terdapat beberapa kemungkinan angka yang sesuai untuk sel tersebut. Pada pengisian selanjutnya, angka yang diisikan akan dicocokkan dengan angka-angka pada sel dalam baris, kolom dan subgrid yang bersesuaian. Metode membandingkan dan pencarian angka yang menuju ke solusi dilakukan secara rekursif. Proses pencarian solusi digambarkan pada diagram blok diatas.

Contoh pencarian solusi yang dapat dilakukan sebagai salah satu penerapan algoritma *backtracking*:

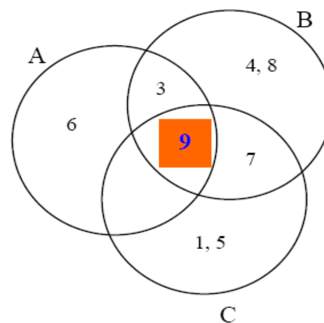
1. Pencarian sel yang kosong.

	1	2	3	4	5	6	7	8	9
1	4		8	7	5		2	1	
2	3			4					
3	6	2					7	8	
4	2			8					1
5	1	5			2			9	7
6	9					1			2
7		1	2					4	9
8						5			6
9		6	4		1	3	5		8

Gambar 5. Pencarian Sel Kosong

Sel pertama yang kosong adalah pada baris ke-1 kolom ke-2.

2. Angka yang akan diisikan adalah 1,2,3,4,5,6,7,8,9.
3. Pencarian kandidat angka yang akan diisikan
   Kandidat angka yang mungkin adalah sebagai berikut:
   Baris ke-1, kolom ke-2 ($S_{1,2}$)
   a. Kandidat yang mungkin pada baris ke-1, B={3,6,9}
   b. Kandidat yang mungkin pada kolom ke-2, K={3,4,7,8,9}
   c. Kandidat yang mungkin pada blok ke-1, G={1,5,7,9}
4. Pengecekan
   Pengecekan dapat direpresentasikan dengan himpunan seperti pada gambar berikut:



Gambar 6. Himpunan Solusi

Himpunan A = {3,6,9}
Himpunan B = {3,4,7,8,9}
Himpunan C = {1,5,7,9}
$A \cap B \cap C = \{9\}$
Jadi, solusinya adalah 9.

5. Solusi diisikan pada sel tersebut.

	1	2	3	4	5	6	7	8	9
1	4	9	8	7	5		2	1	
2	3			4					
3	6	2					7	8	
4	2			8					1
5	1	5			2			9	7
6	9					1			2
7		1	2					4	9
8						5			6
9		6	4		1	3	5		8

Gambar 7. Pengisian Sel Pertama dengan Solusi

ISSN: 2089-1121

6. Selanjutnya melakukan pencarian sel yang kosong berikutnya yaitu sel pada baris ke-1 kolom ke-6. Proses dilakukan berulang-ulang sampai didapatkan solusi untuk sel yang kosong pada baris ke-1:

   $S_{1,6} = \{6\}$, $S_{1,9} = \{3\}$

	1	2	3	4	5	6	7	8	9
1	4	9	8	7	5	6	2	1	3
2	3			4					
3	6	2					7	8	
4	2			8					1
5	1	5			2			9	7
6	9					1			2
7		1	2					4	9
8						5			6
9		6	4		1	3	5		8

Gambar 8. Pengisian Solusi pada Baris ke-1

7. Pencarian solusi pada baris ke-2 akan dihadapkan pada kasus yang berbeda yaitu terdapat himpunan solusi yang lebih dari satu. Dalam hal ini akan dilakukan *backtrack* ke kandidat angka yang lain. Hasil pengecekan berupa himpunan solusi akan disimpan yang kemudian dapat digunakan untuk proses *backtracking*.

   Pencarian kandidat angka untuk baris ke-2:

   $S_{2,2} = \{7\}$, $S_{2,3} = \{1,5\}$, $S_{2,5} = \{6,8,9\}$, $S_{2,6} = \{2,6,8,9\}$,
   $S_{2,7} = \{6,9\}$, $S_{2,8} = \{5,6\}$, $S_{2,9} = \{5\}$

8. Selanjutnya dilakukan *backtrack* berdasarkan himpunan solusi yang telah ada untuk masing-masing sel. Sehingga didapatkan solusi:

   $S_{2,2} = \{7\}$, $S_{2,3} = \{1\}$, $S_{2,5} = \{8\}$, $S_{2,6} = \{2\}$, $S_{2,7} = \{9\}$
   $S_{2,8} = \{6\}$, $S_{2,9} = \{5\}$

	1	2	3	4	5	6	7	8	9
1	4	9	8	7	5	6	2	1	3
2	3	7	1	4	8	2	9	6	5
3	6	2					7	8	
4	2			8					1
5	1	5			2			9	7
6	9					1			2
7		1	2					4	9
8						5			6
9		6	4		1	3	5		8

Gambar 9.Pengisian Solusi pada baris ke-2

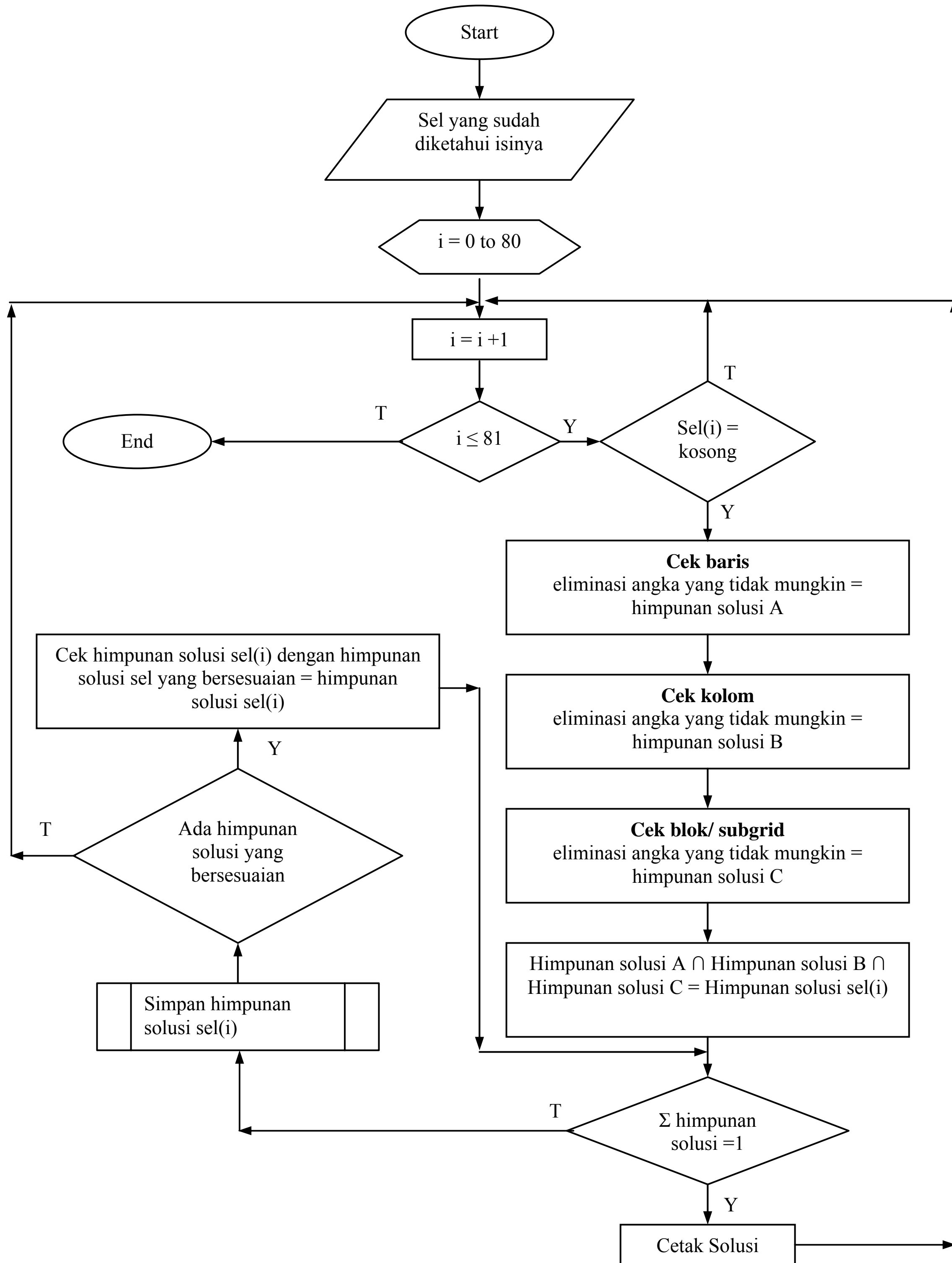
9. Proses pencarian solusi dilakukan berulang-ulang sesuai jumlah sel yang masih kosong sampai seluruh sel terisi oleh angka menurut fungsi pembatas yang ada.

	1	2	3	4	5	6	7	8	9
1	4	9	8	7	5	6	2	1	3
2	3	7	1	4	8	2	9	6	5
3	6	2	5	1	3	9	7	8	4
4	2	4	3	8	9	7	6	5	1
5	1	5	6	3	2	4	8	9	7
6	9	8	7	5	6	1	4	3	2
7	5	1	2	6	7	8	3	4	9
8	8	3	9	2	4	5	1	7	6
9	7	6	4	9	1	3	5	2	8

Gambar 10. Puzzle Sudoku yang telah Terselesaikan

ISSN: 2089-1121

3.5 Flowchart Pencarian Solusi untuk Game Sudoku

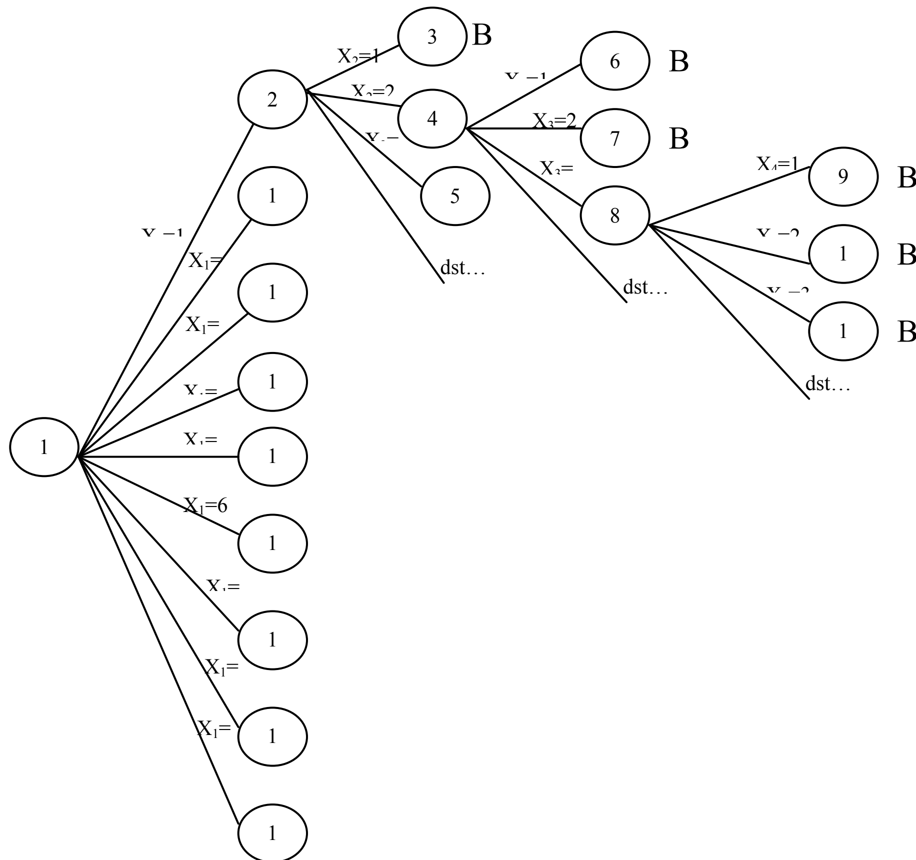


Gambar 11. Flowchart Pencarian Solusi

ISSN: 2089-1121

3.6 Pohon Ruang Status (State Space Tree)

Pohon dinamis yang dibentuk selama pencarian solusi untuk persoalan *puzzle sudoku* dengan ukuran 9x9 adalah:



Gambar 12. Pohon Ruang Status Persoalan Puzzle Sudoku

3.7 Penerapan Algoritma

Algoritma *backtracking* sangat efektif dalam mengurangi jumlah pencarian kemungkinan solusi teka-teki. Menurut perhitungan ada sebanyak 6,670,903,752,021,072,936,960 jumlah kemungkinan status untuk teka-teki sudoku berukuran 9 x 9. Suatu angka yang sangat besar jika ingin dicari secara *brute-force*.

Garis besar algoritma *backtracking* untuk menyelesaikan teka-teki diberikan di bawah ini:

a. Masukan
   - Matriks sel[1...9,1...9]
     Sel [i,j] = *true*, jika solusi ditemukan.
     Sel [i,j] = *false*, jika solusi tidak ditemukan
   - Pengisian sel
     Dinyatakan dengan *integer* 1,2,3,4,5,6,7,8,9.

b. Keluaran
     X[1…9], yang dalam hal ini x[i] adalah angka untuk sel ke-i.

c. Algoritma
   - Inisialisasi x[1…9] dengan 0
     For i ← 1 to 9

ISSN: 2089-1121

X[i] ← 0
Next i

- Pemanggilan prosedur *SolusiSudoku(1)*

4. PENGUJIAN

Pengujian dan analisa pada aplikasi yang akan dilakukan meliputi: kemampuan menyelesaikan game, kebenaran solusi dan kecepatan menemukan solusi. Analisa hasil akan dilakukan dengan membandingkan penggunaan algoritma Backtracking dan algoritma Brute Force dalam menemukan solusi game sudoku.

a. Pengujian kemampuan

Tabel 1. Data Pengujian Kemampuan

No.	Nomor Game	Penyelesaian	Kebenaran
1	108	Berhasil	Berhasil
2	112	Berhasil	Berhasil
3	113	Berhasil	Berhasil
4	114	Berhasil	Berhasil
5	115	Berhasil	Berhasil
6	116	Berhasil	Berhasil
7	117	Berhasil	Berhasil
8	118	Berhasil	Berhasil
9	119	Berhasil	Berhasil
10	120	Berhasil	Berhasil

b. Pengujian kecepatan

Tabel 2. Data Pengujian Kecepatan

Pengujian ke	Nomor Game	Waktu Penyelesaian (mili detik)
1	108	2,32
2	112	2,35
3	113	2,78
4	114	3,01
5	115	2,41
6	116	2,56
7	117	2,35
8	118	2,33
9	119	2,68
10	120	2,79

Pengukuran pengujian kecepatan (waktu) menggunakan kontrol timer pada kotak komponen Visual Basic 6.0 dengan setting interval 1000.

c. Perbandingan dengan algoritma *Brute Force*

Dalam pengujian aplikasi sudoku solution dengan menggunakan algoritma *Backtracking* akan dilakukan juga perbandingan dengan menggunakan algoritma lainnya yaitu algoritma *Brute Force*. Digunakan aplikasi yang telah ada dengan algoritma *Brute Force,*

ISSN: 2089-1121

dibandingkan dengan aplikasi yang dibangun oleh penulis menggunakan algoritma backtracking. Pengujian perbandingan ini hanya mengukur kecepatan aplikasi dalam menemukan solusi untuk menyelesaikan game *sudoku*.

Tabel 3. Data Pengujian Perbandingan dengan Algoritma Brute Force

No.	**Nomor Game**	**Waktu Penyelesaian (mili detik)**	
		Backtracking	**Brute Force**
1	108	2,32	208
2	112	2,35	193
3	113	2,78	212
4	114	3,01	202
5	115	2,41	184
6	116	2,56	176
7	117	2,35	221
8	118	2,33	231
9	119	2,68	196
10	120	2,79	203

5. PENUTUP

Dari analisa-analisa yang telah dipaparkan sebelumnya, dapat disimpulkan bahwa:

1. Permainan teka-teki *Sudoku* termasuk *dynamic problem* yang dapat diselesaikan dengan algoritma *Backtracking*. Dengan menggunakan algoritma ini, teka-teki *Sudoku* dapat diselesaikan dan waktu untuk menyelesaikan teka-teki ini lebih singkat dibandingkan dengan algoritma *brute-force*. Dapat dilihat pada tabel hasil pengujian.
2. Teka-teki *Sudoku* merupakan permasalahan yang memiliki solusi dengan jalur kombinasi, sehingga diperlukan proses *backtrack* untuk mendapatkan solusi secara optimal.
3. Algoritma *Backtracking* mudah diimplementasikan dengan bahasa pemrograman yang mendukung pemanggilan fungsi/ prosedur rekursif karena di dalam algoritma backtracking terdapat proses rekursif untuk melakukan pengecekan kemungkinan solusi untuk sebuah sel kosong.
4. Penerapan algoritma menggunakan versi rekursif akan lebih menyederhanakan penulisan program, sehingga ruang memori yang diperlukan lebih kecil dan waktu eksekusi program lebih cepat jika dibandingkan dengan versi iteratif.

DAFTAR PUSTAKA

Wirasati, Ajeng. Ronny Adry. *Analisa Penerapan Algoritma Backtracking Pada Game " Crossword Puzzle*. Sekolah Tinggi Teknologi Telkom. Bandung. 2005.

Ramadiyan, Deasy Sari, Wulan Widyasari, Eunice Sherta Ria. *Penerapan Algoritma Backtracking pada Pewarnaan Graf*. Fakultas Teknologi Industri ITB. Bandung. 2005.

Pardede, D. L. Crispina. *Himpunan dan Operasi Biner*, Teknik Informatika Universitas Gunadarma. Jakarta. 2003.

Munir, Rinaldi. *Algoritma Runut-balik(Backtracking)*. Departemen Teknik Informatika Institut Teknologi Bandung. 2004.

ISSN: 2089-1121

Meiliawati, Gede Yudha, Sukma Rahadian. *Analisis Penerapan Algoritma Backtracking Dalam Pencarian Solusi Permainan "The Knight In The Green Castle"*. Sekolah Tinggi Teknologi Telkom. Bandung. 2005.

Morenvino, Ray A, Anton R. *Penerapan Algoritma Runut-Balik Untuk Penyelesaian Teka-Teki Sudoku.* Sekolah Tinggi Teknologi Telkom. Bandung. 2005.

Fachtiasmi, Nunu, Jefry Herikson, Jurnalis J. Hius. *Permainan Penyusunan Balok dengan BFS, DFS, dan Backtracking.* Sekolah Tinggi Teknologi Telkom. Bandung. 2005.

Syarif, Iwan. *Pengantar Algorithma & Pemrograman*. Politeknik Elektronika Negeri Surabaya (ITS). 2004.

Setia, Yudha, Ditto Narapratama, Amahl AbdalKarim Salim. *Analisis Beberapa Algoritma Untuk Menyelesaikan Puzzle Sudoku*. Departemen Teknik Informatika, Institut Teknologi Bandung. 2005.