PENYELESAIAN PUZZLE SUDOKU MENGGUNAKAN ALGORITMA BRUTE FORCE DAN BACKTRACKING

Andreas Yusuf[1] dan Hendra[2]
Program Studi Teknik Informatika
STMIK Nusa Mandiri
Jl. Kramat Raya No. 25 Jakarta Pusat
andreas.y21@gmail.com[1]; hendra@nusamandiri.ac.id[2]

Abstrak

Sudoku games is the most popular Numeric Puzzle Games in the world. This game requires you to fill in the numbers on a blank column matrix with certain regulations. This study will discuss how to solve Sudoku primarily to take advantage of the Brute Force Algorithm and Backtracking by trying all possible contents of the box element of the matrix. This testing process using black box method, where the program uses four levels of Sudoku puzzle consists of three levels of difficulty beginner level, intermediate level, advanced level, expert level, master level. Sudoku generally consists of a table with the number of boxes 9 x 9, which made the area (region) 3 x 3. Questions Beginners and Intermediate is used. Examiners performed by using an Intel Pentium III 550MHz processor with a 1.6 GHz Core2 Duo with clock speed 694ms and 320ms for about the first and second largest known differences exist at the level of the processor.

I. PENDAHULUAN

Permainan puzzle termasuk permainan yang membutuhkan nalar dan logika untuk menyelesaikan *goal*. Permainan puzzle dan teka teki sangat banyak jenisnya, salah satunya adalah teka-teki angka.

Penyelesaian *puzzle* Sudoku menggunakan logika memerlukan waktu yang cukup lama, bila dibandingkan dengan pemecahan menggunakan komputer. *Sudoku* adalah singkatan bahasa Jepang dari "*Suuji wa dokushin ni kagiru*", artinya "angka-angkanya harus tetap tunggal". Pertama kali diterbitkan di sebuah surat kabar Perancis pada 1895 dan mungkin dipengaruhi oleh matematikawan Swiss Leonhard Euler.

Permainan ini sudah di kenal di media massa sejak abad 18, ketika permainan *Puzzle Magic Squares* di Prancis ramai dibicarakan. Pada awalnya pemainan Sudoku hanya mengenal baris dan kolom saja tapi dalam perkembangannya ada yang namanya region.

Tahun 1997, sudah banyak ditemukan *puzzle* Sudoku dalam toko uku Jepang, dan dalam 6 tahun berikutnya Sudoku dikembangkan menjadi program computer.

II. KAJIAN LITERATUR

Sudoku adalah sebuah *puzzle* yang didasarkan pada konsep *Latin Square. Sudoku terdiri dari 3×3 kotak, masing-masing terbagi lagi menjadi sembilan kotak lebih kecil. Sehingga sebuah soal sudoku memiliki bagian-bagian yang perlu diberi nama (Satsuko,2007:2).*

Penerapan algoritma Brute force dalam sudoku dengan mencoba seluruh kemungkinan isi kotak elemen dari matriks. Misalkan solusi dari suatu permainan sudoku dinyatakan sebagai: X=(x1,x2,x3,…..,x981); Dari vektor solusi diatas dapat disimpulkan bahwa dengan menggunakan algoritma Brute force maka harus membangkitkan seluruh kemungkinan vektor solusi yang berjumlah 981=1,99x1077 kemungkinan.

III. METODE PENELITIAN

Beberapa cara untuk memecahkan teka-teki Sudoku, seperti algoritma Genetika, *Brute force*, *Backtracking* dan metode *Swordfish*. Pada penelitian ini menggunakan metode algoritma *Brute force* dan *Backtracking* lalu membandingkan keakuratan dan kecepatan dari penyelesaian *puzzle* Sudoku. Algoritma *Backtracking* merupakan perbaikan dari algoritma *Brute force*, dimana solusi dapat ditemukan dengan penelusuran yang lebih sedikit dan dapat mencari solusi permasalahan secara lebih cepat karena tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang perlu dipertimbangkan.

Ruang lingkup yang dikaji dalam penelitian ini yaitu pemecahan *puzzle* Sudoku dengan algoritma *Brute force* dan *Backtracking,* Algoritma yang digunakan dimaksudkan untuk membandingkan waktu yang diperlukan untuk pemecahan *puzzle*

Sudoku, *Puzzle* Sudoku yang digunakan adalah *puzzle* Sudoku, Perangkat lunak yang dibangun hanya menampilkan hasil akhir dari implementasi algoritma yang telah digunakan saja.

IV. PEMBAHASAN

Konstruksi Sistem

Pada perancangan program sudoku ini, menggunakan java sebagai tools. Tampilan awal dari program ini didesain sederhana, karena tujuan dari program ini adalah untuk membuktikan bahwa algoritma yang digunakan dapat memecahkan *puzzle* sudoku.

Prinsip dasar penyelesaian Sudoku sangat sederhana: melengkapi setiap boks dan lajur agar terisi angka 1 sampai 9. Karena masing-masing terdiri dari 9 sel, maka tidak mungkin ada angka ganda dalam setiap boks atau lajur. Setiap soal Sudoku mempunyai satu solusi. Tidak perlu kepandaian menghitung untuk menyelesaikannya, hanya kemampuan membedakan sembilan macam symbol.

Waktu yang dibutuhkan untuk memecahkan satu puzzle yang sudah disediakan oleh penulis dalam program dalam memecahkan puzzle yang sama, rata-rata yang didapat adalah 1170 detik.

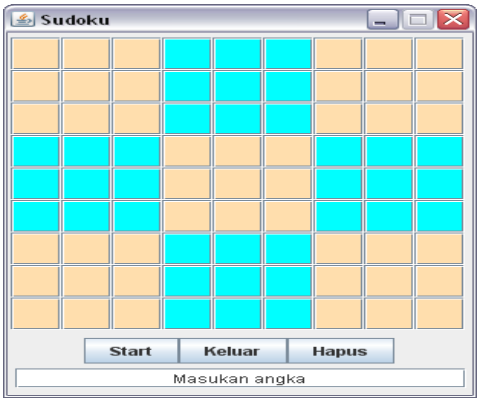

SEL: Kotak terkecil yang seharusnya berisi angka.


BOKS: Kotak lebih besar yang mengandung Sembilan sel, 3x3 sel.


DERET: Lajur-lajur horisontal
KOLOM: Lajur-lajur vertical


BLOK: Kumpulan tiga boks, vertical maupun horizontal.

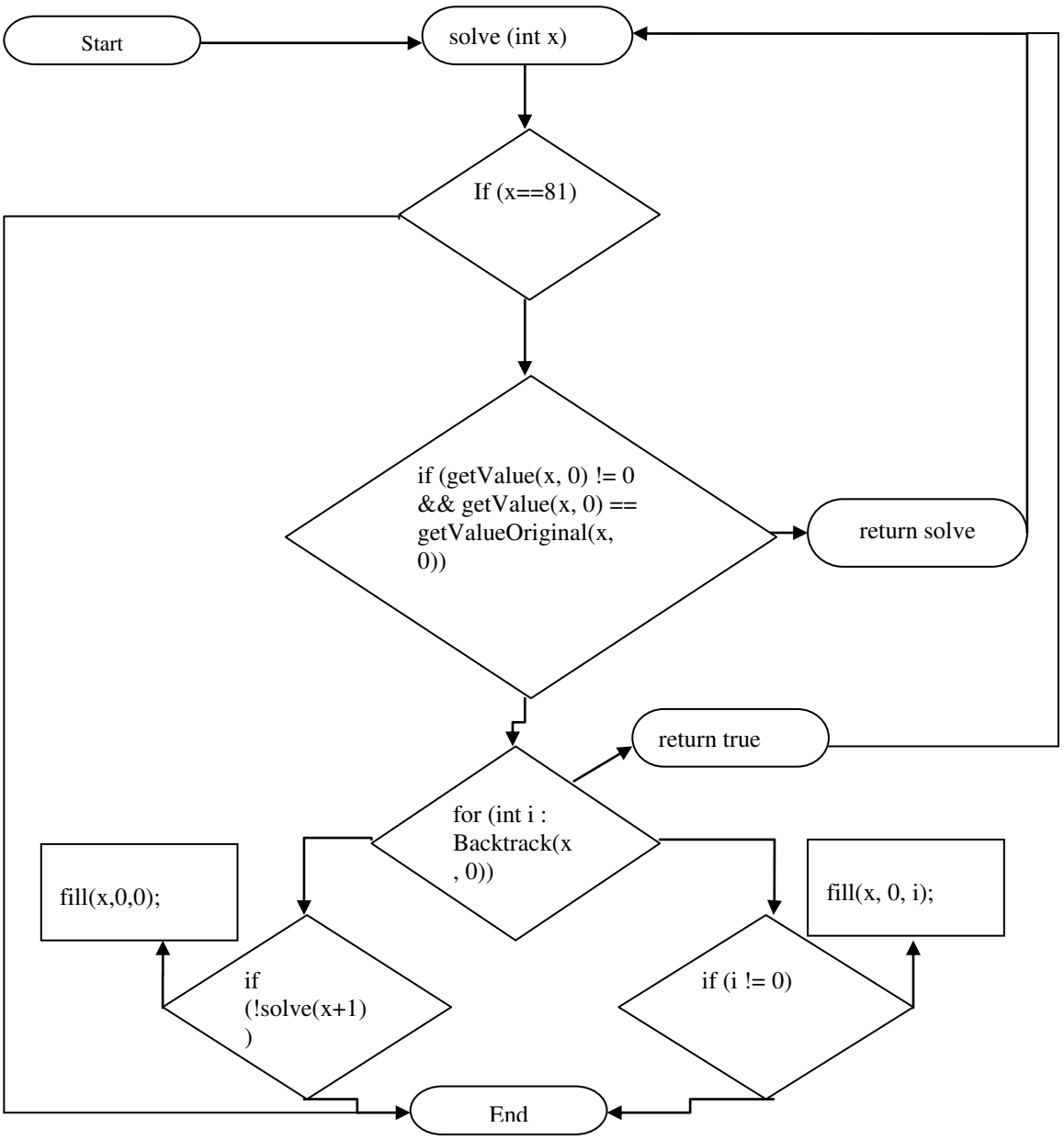
Sebuah soal sodoku terdiri dari 81 sel, atau 9 boks, atau 9 deret, atau 9 kolom, atau 3 blok. Memulainya dapat dilakukan dari mana saja, tergantung struktur soal.



Gambar 1. Sudoku dengan kotak 9 x 9

Varian-varian Sudoku

Sejak Sudoku dipopulerkan banyak variasi puzzle Sudoku, beberapa variasinya berkaitan dengan ukuran grid, objek isian, warna, dan ukuran dimensi. Sudoku mini / Sudoku yunior (Sudoku dengan ukuran grid 4x4(subgrid 2x2) dan 6x6 (subgrid 2x3)), Multisudoku (Sudoku yang berbentuk gabungan / tindihan dua atau lebih frame Sudoku), Irregular Sudoku (Sudoku dengan daerah pengisian yang dibatasi), Monomino (Sudoku dengan warna sebagai batas pengisian), Kubus Sudoku / Kubus Dion Church (Sudoku dalam bentuk kubus), Oddevensudoku (Sudoku yang mengaitkan keunikan angka genap dan ganjil dengan sel yang diarsir), Megasudoku (Sudoku dengan ukuran besar, grid 16x16 (subgrid 4x4), 12x12(subgrid 3x4), dan 25x25 (subgrid 5x5)).



Gambar 2. Penerapan konstruksi Sistem pada Puzzle Sudoku 9x9

Sudoku secara umum terdiri dari tabel dengan kotak 9 x 9, yang dibuat dengan region 3 x 3. Untuk memulai permainan in i, beberapa kotak telah diisi dengan angka-angka yang menjadi petunjuk (clue). Tujuan akhir dari logika ini adalah mengisi kotak-kotak yang masih kosong di mana setiap kotak diisi satu angka. Dalam setiap baris, kolom, dan region diisi dengan angka 1 (satu) sampai dengan 9 (sembilan) dengan kemungkinan hanya sekali. Setiap angka dalam solusi ini terjadi hanya sekali dalam 3 kondisi (baris, kolom dan region).

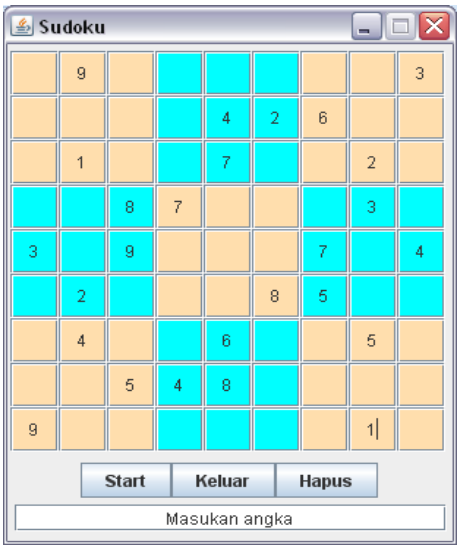
Penerapan Konstruksi sistemnya adalah sebagai berikut:

1. Membuat array yang banyak elemennya sejumlah dengan banyak elemen matriks.
2. Mengalokasikan setiap elemen array ini dengan value, indeks baris, dan indeks kolom dari elemen matriks yang bersesuaian (Penulis menggunakan x, y, value).
3. Menghitung setiap probabilitas elemen–elemen tabel dengan proses pencarian solusi dimulai dari elemen tabel x0, y0 sampai x8, y81.
4. Memeriksa seluruh kemungkinan angka (value) yang dapat dimiliki oleh elemen tabel tersebut (terbatas pada angka 1, 2, 3, 4, 5, 6, 7, 8, 9).

5. Jika tidak terdapat angka yang valid maka backtrack ke elemen tabel sebelumnya dengan melihat urutan elemen tabel yang diproses sebelumnya, ulangi langkah 5.
6. Jika terdapat angka valid maka cari elemen selanjutnya.
7. Algoritma ini akan berhenti jika sudah ditemukan suatu solusi atau jika tidak terdapat kemungkinan adanya solusi.

Pengujian Sistem

Pada proses pengujian ini menggunakan metode black box, dimana program menggunakan 4 tingkatan *puzzle* sudoku yang terdiri dari tiga tingkat kesulitan *beginners level*, i*ntermediate level, advance level, expert level, master level.* Sudoku pada umumnya terdiri dari tabel dengan jumlah kotak 9 x 9, yang dibuat dengan wilayah (region) 3 x 3.



Gambar 3. Pengujian sistem 4 tingkatan

Untuk memulai logika pada permainan Sudoku, ada beberapa kotak yang sebelum nya telah diisi dengan angka-angka yang dimana angka-angka tersebut digunakan sebagai petunjuk (clue) untuk mencari angka-angka yang lain pada kotak yang belum terisi.

Tujuan dari permainan ini adalah mengisi semua kotak yang masih kosong di mana setiap kotak hanya dapat diisi oleh 1 angka. Dalam setiap baris, kolom dan wilayah diisi dengan angka 1 (satu) sampai dengan 9 (sembilan) dengan kemungkinan hanya sekali. Setiap angka dalam solusi ini terjadi hanya sekali dalam 3 kondisi (baris, kolom dan wilayah).

Pengujian berdasarkan perbedaan prosessor

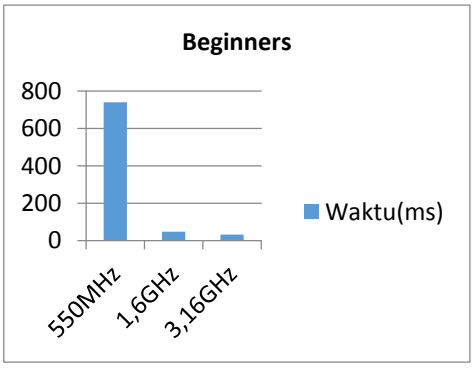
Pada pengujian menguji waktu yang diperlukan program untuk memecahkan *puzzle* dengan menggunakan platform yang berbeda

Tabel 1. Tabel prosessor

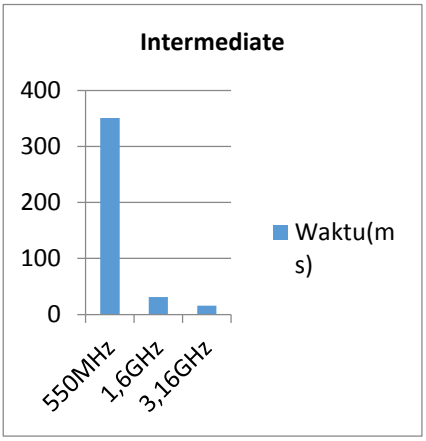
Prosessor	Beginners	Intermediate
Pentium III 550MHz	741ms	351ms
Core2 Duo 1,6Ghz	47ms	31ms
Core2 Duo 3,16GHz	32ms	16ms

Soal yang digunakan adalah *Beginners* dan *Intermediate.* Penguji dilakukan dengan dengan menggunakan prosessor Intel Pentium III 550MHz dengan Core2 Duo 1,6GHz dengan *clock speed* 694ms dan 320ms untuk soal pertama dan kedua diketahui perbedaan terbesar ada pada tingkat prosessor.

Dapat disimpulkan bahwa semakin tinggi *clock* prosessor yang digunakan, semakin cepat waktu yang diperlukan untuk menyelesaikan *puzzle*. Semua soal dapat diselesaiakan dengan baik, jawaban sesuai tes yang diujikan. Tiap tingkat kesuitan dapat diselesaikan tanpa ada masalah.



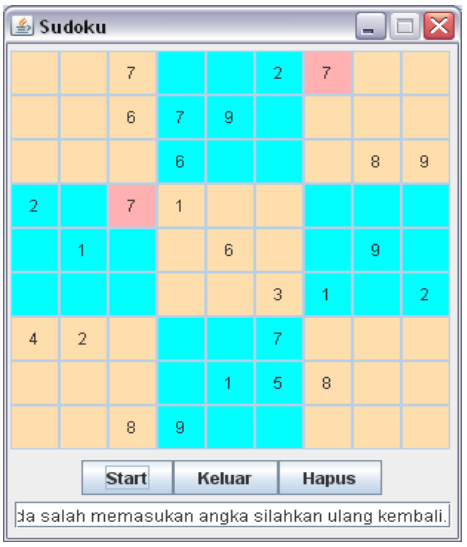
Gambar 4. Grafik Processor level Pemula



Gambar 5. Grafik Processor lanjutan

Pengujian Kesalahan

Pada tahap ini penulis akan menguji program dengan memasukan angka yang tidak valid. Dari pengujian tersebut penulis memasukan angka yang sama (angka 7) pada satu baris dan satu kolom yang sama.



Gambar 6. Pengujian kesalahan

Dari hasil proses, program akan mendeteksi kesalahan dengan menandai angka yang sama. Proses tidak akan berjalan karena tidak boleh ada angka yang sama pada satu baris dan satu kolom yang sama.

Hasil Akhir Pengujian

Perbandingan waktu tempuh untuk menyelesaikan *puzzle* yang diperlukan user bila dibandingkan dengan program dapat dilihat pada Tabel 2. Soal yang digunakan adalah soal *beginners level*. Dari tabel perbandingan dapat diketahui bahwa program dapat menyelesaikan soal jauh lebih cepat daripada menggunakan cara yang konvensional.

Gambar 7. Hasil proses kalkulasi

Tabel 2. Perbandingan waktu

Nama	Waktu(ms)
Pris	1230000
Vina	1030000
Andreas	1260000
Tony	1320000
Ferdian	960000
Kaleb	1260000
Dennis	1050000
Riyan	1240000
Feri	1180000
Program	47

Algoritma Genetika dalam Permainan Sudoku

Penerapan Algoritma Brute Force dalam mencari solusi atau penyelesaian permainan Sudoku didasarkan pada pencarian solusi yang paling optimal. Setiap solusi yang mungkin direpresentaiskan dengan sebuah kromosom. Kromosom-kromosom terrsebut kemudian diperiksa nilai fitnessnya. Kromosom yang nilai fitnessnya paling tinggi atau yang paling

optimal akan dipilih untuk terus hidup atau lajut pada generasi berikutnya dan berpeluang melakukan crossover untuk menghasilkan kromosom atau individu baru yang diharapkan mempunyai nilai fitness yang lebih baik. Dengan adanya mutasi diharapkan dapat memperbaiki kromosom yang sudah ada.

Individu-individu pada generasi-generasi berikutnya diharapkan akan memiliki nilai fitness yang lebih baik dan mengarah pada suatu solusi yang diharapkan. Solusi yang diambil adalah solusi pada individu atau kromosom yang paling besar nilai fitnessnya.

Pindah Silang (Crossover)

Salahsatu komponen paling penting dalam algoritma genetika adalah crossover atau pindah silang. Sebuah kromosom yang mengarah pada solusi yang bagus bisa diperoleh dari proses memindah-silangkan dua buah kromosom.

Skema Pengkodean

Pengkodean suatu kromosom adalah langkah pertama ketika kita menggunakan algoritma genetik untuk menyelesaikan suatu masalah. Pengkodean ini biasanya tergantung kepada masalah yang dihadapi. Pengkodean ini meliputi pengkodean terhadap gen yang terdapat dalam kromosom. Gen dapat dipresentasikan dalam bentuk : string bit, pohon, array bilangan real, daftar aturan, elemen permutasi, elemen program, atau representasi lainnya yang dapat diimplementasikan untuk operator genetika

package sudoku;

```
public class Sudoku {

private int[][] matrix = new int[9][9];
private int[][] inputMatrix = new int[9][9];

public void resetMatrix() {
        matrix = new int[9][9];
inputMatrix = new int[9][9];
}

private void fill(int x, int y, int value) {

        while (x < 0) {
                x += 9;
                y--;
        }

        while(x > 8) {
                x -= 9;
                y++;
        }

matrix[x][y] = value;
        }
public void set(int x, int y, int value) {
        while (x < 0) {
                x += 9;
                y--;
        }
        while(x > 8) {
        x -= 9;
        y++;
                }

                inputMatrix[x][y] = value;
        }


        private void copyMatrix() {
                for (int x = 0; x != 9; x++)
                        for (int y = 0; y
!= 9; y++)

        matrix[x][y] = inputMatrix[x][y];
        }


        public boolean solve() {
                copyMatrix();
                return solve(0);
        }


        public int getValue(int x, int y) {

                while (x < 0) {
                        x += 9;
                        y--;
                }

                while(x > 8) {
                        x -= 9;
                        y++;
                }

                return matrix[x][y];
        }

        public int getValueOriginal(int x,
int y) {

                while (x < 0) {
                        x += 9;
                        y--;
                }

                while(x > 8) {
```

```
                x -= 9;
                y++;
            }

            return inputMatrix[x][y];
        }

        private boolean solve(int x) {
            if (x == 81)
                return true;
        if (getValue(x, 0) != 0 &&
getValue(x, 0) == getValueOriginal(x, 0)) {
                return
solve(x+1);
            }
            else {
                for (int i :
Backtrack(x, 0)) {
                    if (i !=
0) {

        fill(x, 0, i);


        if (!solve(x+1))

        fill(x,0,0);

        else

        return true;
                    }
                }
            }
            return false;
        }


        public boolean
validNumberOriginal (int x, int y, int
number) {
        if (number < 1 || number > 9)
                return false;

            while (x < 0) {
                x += 9;
                y--;
            }

            while(x > 8) {
                x -= 9;
                y++;
            }

            int[] validNumbers =
{1,2,3,4,5,6,7,8,9};

            // cek vertikal, horizontal
            for (int i = 0; i != 9; i++) {
                if
(inputMatrix[i][y] != 0)

        validNumbers[inputMatrix[i][y]-1]
= 0;
                if
(inputMatrix[x][i] != 0)

        validNumbers[inputMatrix[x][i]-1]
= 0;
            }

            // cek kotak
            int squareX, squareY;

            if (x < 3)
                squareX = 0;
            else if (x < 6)
                squareX = 3;
            else
                squareX = 6;

            if (y < 3)
                squareY = 0;
            else if (y < 6)
                squareY = 3;
            else
                squareY = 6;

            for (int i = 0; i != 3; i++) {
                for (int j = 0; j !=
3; j++) {
                    if
(inputMatrix[i+squareX][j+squareY] != 0) {

        validNumbers[inputMatrix[i+squar
eX][j+squareY]-1] = 0;
                    }
                }
            }

            return
(validNumbers[number-1] == 0) ? false :
true;
        }
        private int[] Backtrack (int x, int y)
{
            while (x < 0) {
                x += 9;
                y--;
            }

            while(x > 8) {
                x -= 9;
                y++;
            }

            int[] validNumbers =
{1,2,3,4,5,6,7,8,9};
```

```
                // cek vertikal horizontal
                for (int i = 0; i != 9; i++) {
                        if (matrix[i][y] !=
0) {

        validNumbers[matrix[i][y]-1] = 0;
                        }
                        if (matrix[x][i] !=
0) {

        validNumbers[matrix[x][i]-1] = 0;
                        }
                }
                // cek kotak
                int squareX, squareY;
                if (x < 3)
                        squareX = 0;
                else if (x < 6)
                        squareX = 3;
                else
                        squareX = 6;
                if (y < 3)
                        squareY = 0;
                else if (y < 6)
                        squareY = 3;
                else
                        squareY = 6;

                for (int i = 0; i != 3; i++) {
                        for (int j = 0; j !=
3; j++) {
                                if
(matrix[i+squareX][j+squareY] != 0) {

        validNumbers[matrix[i+squareX][j
+squareY]-1] = 0;
                                }
                        }
                }
                return validNumbers;
        }
        public void printTable() {
                System.out.print("\n");
                for (int x = 0; x != 9; x++)
{
                        for (int y = 0; y
!= 9; y++) {

        System.out.print(" " +
inputMatrix[x][y] + " ");
                                if (y ==
2 || y == 5)

        System.out.print("|");
                        }
                        if (x == 2 || x ==
5)

        System.out.print("\n---------+--------
-+--------");

        System.out.print("\n");
                }
        }
}
```

Elitisme

Karena seleksi dilakukan secara random, maka tidak ada jaminan bahwa suatu individu bernilai fitness tertinggi akan selalu terpilih. Kalaupun individu bernilai fitness tertinggi terpilih, mungkin saja individu tersebut akan rusak (nilai fitnessnya menurun) karena proses pindah silang. Untuk menjaga agar individu bernilai fitness tertinggi tersebut tidak hilang selama evolusi, maka perlu dibuat satu atau beberapa kopinya. Prosedur ini dikenal sebagai elitisme.

PENUTUP

Pada pengujian program menggunakan soal Beginners, Intermediate, Advance, Expert, Master didapatkan bahwa waktu tempuh program berbeda-beda. Semakin tinggi tingkat kesulitan, semakin lama waktu yang diperlukan.

Untuk pengujian kedua semua soal yang diuji dapat diselesaikan dengan tepat oleh program dengan hasil Algoritma Brute force dan Backtracking terbukti dapat menyelesaikan puzzle sudoku. Algoritma Brute force dan Backtracking dapat digunakan untuk menyelesaikan puzzle sudoku dengan baik. Waktu yang diperlukan untuk memcahkan puzzle tidak dipengaruhi oleh tingkat kesulitan dari puzzle itu sendiri. Waktu untuk memecahkan puzzle lebih dipengaruhi oleh spesifikasi komputer yang digunakan terutama prosessor. Semakin tinggi clock prosessor, semakin cepat waktu yang diperlukan untuk menyelesaikan puzzle.

Proses pengujian program didapatkan hasil 47ms, 31ms, 16ms, 16ms, 31ms untuk masing-masing tingkat (Beginners, Intermediate, Advance, Expert, Master) sedangkan untuk hasil pengujian berdasarkan perbedaan prosessor (Pentium III 550MHz, Core2 Duo 1,6GHz, Core2 Duo 3,16GHz) didapatkan hasil 741ms, 47ms, 32ms untuk pengujian pertama dan 351ms, 31ms, 16ms.

DAFTAR PUSTAKA

Crook, J.F. 2009. *A Pencil-and-Paper Algorithm for Solving Sudoku Puzzle.* Notices of the AMS. Vol: 56, No. 4.

Davis, Tom. 2010. *The Mathematics of Sudoku.* Diambil dari: http://www.geometer.org/mathcircles. (20 Juni 2011).

Gurari, Eitan, 1999. *Data Structures Chapter: General Algorithms & State Search Algorithms.* www.cse.ohio-state.edu/~gurari/course/cis680/cis680No1.html##QQ1-29-103. (24 Juni 2011).

Jussien, Narendra. 2007. *A-Z Sudoku.* ISTE Ltd.

Kadir, Abdul. 2007. Dasar Pemrograman Java 2. Yogyakarta: ANDI OFFSET.

Mepham, Michael. 2005. *Solving Sudoku.* Crosswords Ltd.

Omimura, Satsuko. 2009. *Sudoku The Black Hat.* Jakarta: Prestasi Pustaka.

Hendra. Meraih gelar Sarjana Komputer dari STMIK Indonesia pada tahun 1998. Pada tahun 2010 mendapatkan gelar Magister Komputer dari STMIK Nusa Mandiri. Aktif dalam berbagai kegiatan penelitian dan seminar ilmiah. Menjadi anggota Asosiasi Dosen Indonesia (ADI) dan Asosiasi Perguruan Tinggi Ilmu Komputer (APTIKOM).