Penggunaan Algoritma Runut-balik Pada Pencarian Solusi dalam Persoalan *Magic Square*

Tahir Arazi – NIM : 13505052

Program Studi Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung, Jawa Barat
e-mail: if15052@students.if.itb.ac.id

ABSTRAK

Magic square **merupakan salah satu persoalan yang berkaitan dengan komposisi bilangan, sedemikian sehingga tiap elemen pada matriks dengan orde *n* mengandung nilai yang berbeda satu sama lain, terbatas pada bilangan bulat dari 1 hingga n^2, serta setiap baris, kolom, dan diagonal hasil penjumlahan yang sama. Terdapat pula banyak cara untuk mencari nilai dari tiap-tiap elemen tersebut. Namun, hingga saat ini belum ditemukan aturan yang bersifat umum untuk semua matriks *n* x *n* dengan orde *n* berapapun, kecuali *n*=2.**

Makalah ini membahas mengenai pemakaian algoritma runut-balik untuk mencari solusi berupa *magic square* yang berlaku umum untuk seluruh orde matriks *n* x *n*, kecuali *n*=2. Algoritma runut-balik juga merupakan perbaikan dari algoritma *brute force* dengan cara melakukan pemangkasan (*pruning*) pada pohon ruang status. Terdapat pula pembahasan mengenai implementasi konsep runut-balik ini dalam program, serta hasil analisis dari program yang telah dibangun oleh penulis. Kode sumber dari program yang telah dibangun oleh penulis dapat diunduh dari alamat *website* : http://students.if.itb.ac.id/~if15052/download/magicsquare

Kata kunci: *Magic square*, algortima runut-balik.

1. PENDAHULUAN

Magic Square, dalam matematika, adalah matriks yang terdiri dari beberapa angka yang berbeda satu sama lain dan disusun sedemikian sehingga penjumlahan angka pada setiap baris, kolom, dan diagonal adalah sama. Matriks pada *magic square* berukuran *n x n*, dengan elemen yang berbeda satu sama lain berupa bilangan bulat dari 1 hingga n^2 . Jumlah dari deret $1+2+3+\ldots+n^2$ dapat ditentukan melalui persamaan :

$$1+2+3+...+n^2 = \frac{n^2(n^2+1)}{2} \quad (1)$$

Dari persamaan (1), maka dapat ditentukan pula bahwa jumlah angka-angka pada tiap baris, kolom, dan diagonal adalah :

$$\frac{n^2(n^2+1)}{2} \times \frac{1}{n} = \frac{n(n^2+1)}{2} \quad (2)$$

Jumlah tersebut menghasilkan angka yang disebut dengan *the magic-square constant* [1] (konstanta *magic square*). Berikut adalah gambaran dari sebuah *magic square*.

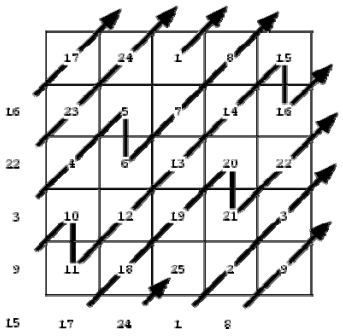
```
2  7  6
9  5  1
4  3  8
```

Gambar 1. Salah satu bentuk *Magic Square* dalam orde n=3

Magic square tidak terbatas pada kesamaan jumlah pada setiap diagonal, baris, atau kolom. Pada gambar 1, (2,5,8) dan (6,5,4) adalah contoh dari diagonal yang lebih dikenal dengan nama *triads*. Untuk set angka (7,1,4); (6,9,3); (2,1,3); dan (7,9,8) dikenal dengan nama *broken diagonal*. Sebuah *magic square* dikatakan *panmagic* apabila jumlah dari setiap *broken diagonal* sama dengan nilai dari konstanta *magic square*. Sebuah *magic square* yang ketika setiap elemennya digantikan dengan hasil kuadrat elemen tersebut dan kondisi matriks tetap sebagai sebuah *magic square*, disebut dengan *bimagic*. *Magic square* dikatakan *trimagic* apabila setiap elemennya jika digantikan dengan hasil pangkat tiga atau kuadrat dari elemen tersebut akan tetap menghasilkan sebuah *magic square*. Pada laporan teknis ini, pencarian solusi untuk elemen-elemen *magic square* akan dibatasi pada *magic square* pada umumnya, yakni yang memiliki konstanta *magic square* hanya pada setiap baris, kolom, dan diagonal (*triads*).

Hingga kini, belum ditemukan sebuah aturan yang bersifat umum dalam menentukan elemen-elemen untuk semua *magic square* dengan orde *n*. Jumlah dari kemungkinan *magic square* dengan orde *n* juga tidak dapat ditentukan melalui sebuah persamaan. Untuk *magic square* dengan orde *n*, dimana *n* adalah bilangan ganjil, terdapat suatu metode yan dikenal dengan nama *the Siamese* [2]. Sebagai gambaran yang lebih jelas mengenai

cara kerja dari metode ini, akan diilustrasikan melalui gambar dibawah ini.



Gambar 2. Pengisian *Magic Square* dengan orde n=5

Langkah pertama dalam metode *the Siamese* adalah peletakan angka 1 pada sembarang kotak. Selanjutnya pengisian dilakukan secara berurut pada kotak disebelah kanan-atas. Hal ini berarti, ketika angka sudah berada pada baris paling atas, maka angka berikutnya diisi pada kotak dikolom sebelah kanan dengan baris paling bawah. Begitu pula pada saat angka sudah berada pada kolom paling kanan, maka selanjutnya pengisiaan angka ada pada kotak dikolom pertama pada baris diatasnya. Jika kotak yang akan diisi ternyata sudah terisi, maka pengisian dilanjutkan pada kotak yang berada dibawahnya, dan kemudian melanjutkan pengisian pada kotak disebelah kanan-atas.

Metode *the Siamese* hanya berlaku pada *magic square* dengan orde *n*, dimana *n* adalah bilangan ganjil, dan tidak berlaku untuk *n* yang merupakan bilangan genap. Penulis mencoba untuk menerapkan suatu algoritma dalam suatu program komputasi, sedemikian sehingga program tersebut berlaku umum dalam mencari solusi untuk setiap *magic square* dengan orde *n*, dimana *n* adalah bilangan bulat lebih besar dari 0 (nol) dan $n \neq 2$. Algortima yang akan diimplementasikan adalah algortima runut-balik.

2. METODE

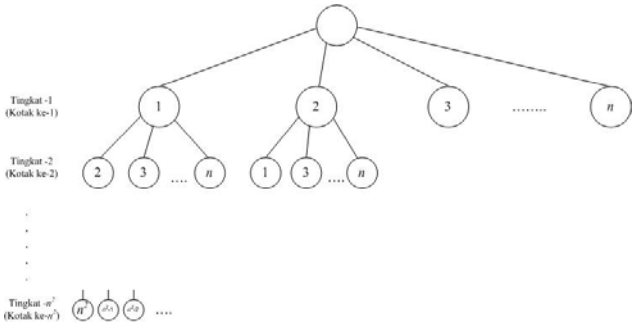
Sebelum melangkah pada implementasi algortima runut-balik dalam mencari solusi dari persoalan *magic square*, terlebih dahulu akan dijabarkan penyelesaian persoalaan *magic square* melalui pendekatan yang lempang, yakni dengan algoritma *brute force*.

2.1 Algoritma *Brute Force*

Algortima *Brute Force* merupakan algoritma dengan pendekatan yang lempang (*straight forward*), artinya tidak terdapat metode khusus untuk menyelesaikan persoalan dengan lebih efisien dan efektif. Dalam implementasinya dalam menyelesaikan persoalan *magic square*, algoritma *brute force* yang paling lempang adalah dengan mencoba semua kemungkinan angka pada setiap elemen matriks dengan orde *n*. Setelah mencoba semua kemungkinan yang ada, barulah setiap kemungkinan tersebut dihitung jumlah tiap baris, kolom, dan diagonalnya yang kemudian dicocokan dengan konstanta *magic square* sesuai dengan ordenya. Pendekatan ini lebih dikenal dengan nama *exhaustive search*.

Pada matriks yang memiliki orde *n* dengan jumlah elemen sebanyak *m*, dimana *m* adalah hasil kuadrat dari *n*. Angka-angka yang diperbolehkan untuk diisi pada setiap elemen adalah antara 1 hingga n^2 . Hal ini berarti dengan metode *exhaustive search*, akan dihasilkan sebanyak $n^2!$ kemungkinan dalam mengisi elemen-elemen pada matriks dengan orde *n*. Setelah semua kemungkinan telah diperoleh, maka selanjutnya akan dilakukan perhitungan jumlah setiap baris, kolom, dan diagonal pada tiap kemungkinan yang telah diperoleh. Dengan begitu maka akan dilakukan perhitungan sebanyak (2 (jumlah diagonal) + *n*(jumlah kolom) + *n*(jumlah baris)) * $n^2!$(jumlah kemungkinan) kali. Setelah semua kemungkinan memiliki data mengenai jumlah masing-masing baris, kolom, dan diagonal, maka selanjutnya akan dilakukan proses perbandingan antara baris, kolom, dan diagonal sebanyak $n^2 * 2 * n^2!$ atau $2n^2 * n^2!$ kali. Jika keseluruhan proses digabungkan, maka akan terdapat sebanyak $n^2! + ((2 + 2n) * n^2!) + (2n^2 * n^2!)$ kali proses perhitungan. Untuk matrik dengan orde *n*=3 maka akan terdapat sebanyak 362,880 + 2,903,040 + 6,531,840 atau 9,797,760 kali perhitungan.

Melalui perhitungan diatas dapat dibayangkan seberapa lama waktu yang dibutuhkan untuk menyelesaikan sejumlah proses diatas. Maka dari itu, metode penyelesaian persoalan *magic square* akan tidak mangkus apabiladilakukan melalui pendekatan *exhaustive search*. Gambaran secara kasar dari pembentukan pohon ruang status melalui pendekatan *exhaustive search* adalah sebagai berikut.



Gambar 3. Pohon ruang status yang terbentuk melalui pendekatan *exhaustive search*

Meskipun dengan algoritma *brute force* solusi dari persoalan *magic square* pasti akan ditemukan, namun tentunya waktu yang dibutuhkan dalam mencari satu solusi akan sangat lama. Untuk mempersingkat pencarian solusi, maka diperlukan suatu metode untuk memangkas (*pruning*) pohon ruang status sehingga setiap langkah yang dilakukan lebih menuju kepada solusi dan tidak mencoba semua kemungkinan yang ada. Algortima runut-balik merupakan salah satu metode untuk memangkas pohon ruang status yang dibentuk secara dinamis.

2.2 Algoritma Runut-balik

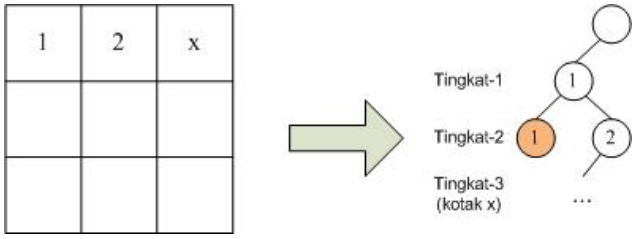
Algoritma runut-balik merupakan salah satu metode dalam pemangkasan pohon ruang status. Algortima ini merupakan kembangan dari metode DFS (*Deep First Search*) yang kemudian memiliki fungsi batas (*constraints*) agar pencarian yang dilakukan lebih mengarah kepada solusi. Fungsi batas merupakan faktor yang akan menyebabkan program untuk melakukan runut-balik. Fungsi batas yang diterapkan akan dijelaskan pada sub bab selanjutnya. Terdapat pula metode dalam pengisian elemen-elemen matriks, yakni elemen pertama yang akan diisi adalah elemen pada baris dan kolom pertama, kemudian dilanjutkan pada elemen kolom berikutnya dan setelah selesai mengisi satu baris, maka akan dilanjutkan pada kolom pertama pada baris selanjutnya. Elemen-elemen/kotak yang diisi direpresentasikan sebagai tingkatan dalam pembentukan pohon ruang status secara dinamis. Sedangkan angka-angka yang mungkin untuk diisi, direpresentasikan sebagai daun pada pohon ruang status. Selanjutnya kan dibahas mengenai fungsi batas yang digunakan dalam implementasi algoritma runut-balik pada penyelesaian persoalan *magic square*.

2.2.1 Fungsi Batas

Proses runut-balik terjadi ketika daun yang dibangkitkan oleh program pada suatu tingkatan tidak sesuai dengan fungsi batas. Untuk persoalan *magic square*, fungsi batas terdiri dari beberapa kasus. Kasus yang pertama adalah ketika tingkatan yang dicapai merupakan elemen terakhir yang akan diisi pada suatu baris dan/atau kolom dan/atau diagonal. Fungsi batas mengharuskan kolom tersebut untuk diisi dengan elemen, sedemikian sehingga elemen tesebut menjadikan baris, kolom, atau diagonal tersebut berjumlah sama dengan konstanta *magic square* yang ditentukan sesuai dengan orde matriks. Kasus kedua adalah ketika daun yang dibangkitkan untuk mengisi elemen matriks sudah berada pada elemen matriks lainnya. Fungsi batas mengharuskan setiap elemen memiliki nilai yang berbeda satu sama lain. Kasus ketiga adalah ketika pada suatu tingkatan daun yang telah dibangkitkan telah mencapai nilai maksimum, yakni jika matriks memiliki orde *n*, maka nilai maksimum dari suatu elemen adalah n^2, sehingga harus melakukan proses runut-balik ke akar.

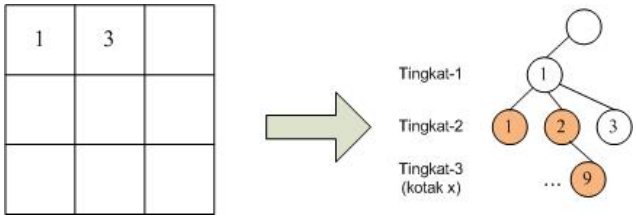
Dikarenakan metode pengisian elemen/kotak (pembangkitan tingkatan pada pohon ruang status) dilakukan secara berurutan dari kolom pertama hingga kolomterakhir dan dari baris pertama hingga baris terakhir, maka pada umumnya proses runut-balik terjadi ketika telah sampai pada kolom terakhir pada suatu baris. Proses runut-balik akan terus berlangsung hingga baris tersebut berjumlah sama dengan konstanta *magic square*. Akan menjadi kasus terburuk pada suatu baris apabila elemen yang salah berada pada kolom pertama. Hal ini akan menyebabkan proses runut-balik pada setiap kolom. Kasus terburuk lainnya, yang disebabkan metode pengisian tersebut, adalah ketika program telah sampai pada pembentukan tingkatan dimana tingkatan tersebut merupakan elemen terakhir yang harus diisi pada suatu kolom dan diagonal. Kasus ini akan semakin buruk ketika proses runut-balik tersebut terjadi akibat nilai yang salah pada elemen kolom dan baris pertama.

Sebagai gambaran proses pengisian elemen-elemen matriks dan runut-balik yang terjadi, akan diilustrasikan melalui gambar-gambar berikut ini.



Gambar 4. Pengisian elemen-elemen pada matriks 3x3 dan representasinya dalam pohon ruang status

Pada gambar 4, pengisian dilakukan dari kolom pertama dengan nilai terkecil yakni 1. Pada kolom berikutnya diisi dengan nilai dari yang terkecil, yakni 1, dan apabila nilai tersebut tidak sesuai dengan fungsi batas, maka akan terjadi proses runut balik (ditandai dengan daun yang diarsir pada pohon ruang status).



Gambar 5. Proses runut-balik pada elemen terakhir suatu baris dan representasinya dalam pohon ruang status

Pada gambar 5, ketika seluruh kemungkinan telah dicoba untuk mengisi elemen pada kolom terakhir suatu baris, maka akan terjadi proses runut-balik menuju akar dan pembangkitan daun lain untuk mencoba kemungkinan lainnya.

2.2.2 Implementasi dalam Program

Setelah konsep runut-balik dan fungsi batas telah ditentukan, maka saatnya konsep tersebut diimplementasikan pada program yang sebenarnya. Penulis mengimplementasikan konsep tersebut dalam program yang ditulis dengan bahasa Java™. Dalam program tersebut, secara garis besar terdapat 2 struktur data yang mendukung proses runut-balik, yakni matriks dengan orde *n*, yang akan diisi dengan elemen-elemen sedemikian sehigga akan membentuk *magic square*, dan *array* yang berisi bilangan bulat dari 1 hingga n^2 serta penanda (bertipe `boolean`) bahwa nilai tersebut telah diisikan atau tidak pada elemen matriks lainnya.

Pada matriks, pertama-tama akan diinisialisasi dengan nilai 0 (nol), sehingga nantinya proses pengisian elemen berarti menambahkan 1 pada nilai elemen matriks. Sebagai implementasi dari fungsi batas, maka terdapat sebuah fungsi `validate`, dimana fungsi ini akan memeriksa seluruh kasus dalam fungsi batas. Dalam skema runut-balik, diimplementasikan dengan 2 kalang `while` dengan satu kalang yang terus melakukan iterasi hingga ditemukannya solusi atau indeks telah keluar dari batas indeks matriks, dan satu kalang lagi untuk terus menambahkan nilai dari elemen matriks hingga nilai tersebut sesuai dengan fungsi batas. Untuk lebih jelasnya, berikut adalah potongan kode sumber dari program yang menunjukkan proses runut-balik.

```
public void FillIn()
{
    int i;
    int x;
    int y;
    boolean solusi=false;
    boolean bcktracked=false;

    x=0;
    y=0;
    while(!solusi && y>-1)
    {
        if(matrix[x][y]==0)
        {
          matrix[x][y]=matrix[x][y]+1;
        }

        //penambahan isi dari elemen matrix
//hingga valid atau sudah lebih besar dari
//side*side


        while((matrix[x][y]<=(size*size))&&!valid
        ate(x,y,bcktracked))
        {
           if(bcktracked)
           {
             bcktracked=false;
           }
             matrix[x][y]=matrix[x][y]+1;
        }

         if(matrix[x][y]>(size*size))//backtrack
        {
            matrix[x][y]=0;
            if((y==0)&&(x!=0))//diujung matrix
            {
              y=size-1;
               x--;
            }
            else//belum diujung
            {
               y--;
            }
            if(y!=-1)
            {
               refresh(matrix[x][y]);
            }
            bcktracked=true;
           }
           else//next element
           {
             refresh(matrix[x][y]);
             if(y==size-1)
             {
                 x++;
                 y=0;
             }
             else
             {
                 y++;
             }
             if(x>size-1)
             {
                 solusi=true;
              }
           }
         }
    }
```

Ketika indeks telah keluar dari batas (lebih kecil dari -1), maka dapat dipastikan bahwa matriks dengan orde tersebut tidak memiliki satupun solusi. Faktanya, hanya terdapat satu orde matriks yang tidak memiliki solusi berupa *magic square*, yakni matriks dengan orde *n*=2.

2.2.3 Analisis Pengujian Program

Setelah program selesai dibangun, penulis juga melakukan pengujian terhadap program tersebut. Pengujian pertama dilakukan dengan memasukkan orde matriks bernilai *n*=1. Hasil yang didapatkan berupa *magic square* dengan satu elemen bernilai 1. Pengujian kedua dilakukan dengan memasukkan orde matriks bernilai *n*=2. Hasil yang didapat adalah tidak terdapat solusi berupa *magic square* dan hanya menampilkan matriks dengan keseluruhan elemen bernilai 0 (nol). Selanjutnya pengujian dilakukan dengan memasukkan orde matriks bernilai *n*=3. Dalam waktu 0.33 detik, ditampilkan satu

solusi berupa *magic square*. Berikutnya dilakukan pengujian dengan memasukkan orde matriks bernilai *n*=4. Waktu yang dibutuhkan dalam mencari solusi mencapai 2.41 detik. Perbedaan waktu pencarian solusi ternyata tidak terlalu signifikan dibandingkan matriks dengan orde *n*=3. Waktu yang sangat signifikan berbeda terjadi ketika mencari solusi pada matriks dengan orde *n*=5. Setelah lebih dari 2 jam solusi belum juga ditemukan dan akhirnya pengujian diakhiri karena alasan tertentu.

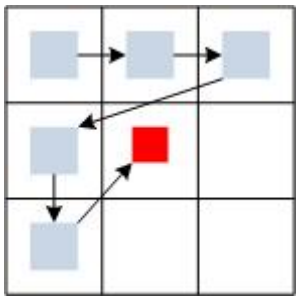
Sebelum pengujian terakhir, ternyata program berhasil mencari solusi untuk matriks dengan orde ganjil maupun genap. Namun, pada saat pengujian matriks dengan orde *n*=5 ternyata memakan waktu yang sangat lama. Hal ini antara lain dapat dikarenakan program tidak terlalu mangkus dalam pencarian solusi sehingga ketika data yang diolah cukup besar, waktu yang dibutuhkan dalam pencarian solusi juga ikut membesar. Ketidakmangkusan dari program dapat disebabkan karena implementasi dari fungsi batas yang banyak mengandung iterasi dalam memeriksa posisi dari elemen yang akan diisi serta menjumlahkan baris, kolom, dan diagonal yang kemudian dicocokkan dengan konstanta *magic square*.

Faktor lain yang dapat menyebabkan lamanya waktu pencarian solusi adalah metode pengisian elemen yang terlalu lempang dan tidak mengutamakan elemen-elemen yang akan berada baris/kolom/diagonal yang lebih terisi. Ketika pengisian dilakukan dengan skema tersebut, maka apabila terjadi proses runut-balik, maka runut-balik yang terjadi akan lebih pendek. Selain itu, fungsi batas yang ada masih menyebabkan runut-balik yang terlalu panjang, karena pada terdapat kasus dimana pemeriksaan jumlah pada baris/kolom/diagonal baru terjadi pada elemen terakhir yang akan diisi. Akan lebih baik jika terdapat suatu pendekatan heuristik agar nilai elemen yang diisi berikutnya, jika dijumlahkan dengan elemen sebelumnya akan menghasilkan jumlah yang mendekati konstanta *magic square*.

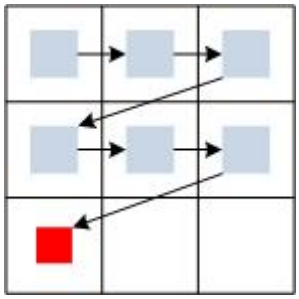
3. KESIMPULAN

Algoritma runut-balik dapat digunakan untuk pencarian solusi persoalan *magic square* untuk orde *n*, baik bernilai genap ataupun ganjil, walaupun dalam implementasi program belum memiliki cara pengisiian dan fungsi batas yang mangkus untuk mempercepat proses pencarian solusi.

Dalam cara pengisian elemen-elemen pada matriks, seharusnya dilakukan dengan skema yang mengutamakan elemen-elemen pada baris/kolom/diagonal yang lebih terisi sehingga apabila terjadi proses runut-balik, maka akan memperpendek panjang runut-balik pada kasus terburuk. Berikut adalah perbandingan skema pengisiian elemen yang lempang dan skema yang lebih baik (optimal), diilustrasikan melalui gambar.



Gambar 6. Skema pengisian elemen matriks yang lebih optimal



Gambar 7. Skema pengisisan elemen matriks yang lebih lempang

Pada gambar 6, pertama-tama pengisian dilakukan dari kolom dan baris pertama hingga kolom terakhir pada baris tersebut. Kemudian pengisian dilanjutkan pada baris kedua kolom pertama dan baris ketiga kolom pertama. Selanjutnya pengisian akan dilakukan pada kolom dan baris kedua.

Pada gambar 7, langkah awal sama dengan skema dengan optimasi, namun berbeda pada langkah ke-5, dimana pada skema ini akan mengisi kolom kedua pada baris kedua.

Jika kedua skema ini dibandingkan, maka dapat terlihat bahwa ketika masuk pada langkah pengisian elemen yang ditandai dengan warna merah, pada saat itu kedua skema fokus dalam bagaimana menghasilkan komposisi nilai elemen yang dapat menghasilkan kolom (pertama) dan diagonal (kanan ke kiri) yang memiliki jumlah sesuai dengan konstanta *magic square*. Namun, pada skema pengisian yang lempang, program juga harus melakukan proses runut balik terhadap elemen pada kolom ke-3 baris ke-2. Hal ini berarti, skema pengisian yang lempang telah melakukan pemborosan 1 elemen, yang dapat berakibat pencarian solusi yang lebih lama.

Untuk masalah fungsi batas, sesungguhnya dapat diperbaiki dengan menerapkan pendekatan heuristik. Salah satu pendekatan heuristik adalah menentukan persamaan-persamaan lain untuk mengisi elemen-elemen

matriks yang tidak terbatas pada pengisian elemen terakhir pada baris/kolom/diagonal. Berikut adalah persamaan-persamaan lain untuk elemen c_{ij} dimana *i* adalah nomor baris dan *j* adalah nomor kolom. Persamaan ini berlaku untuk *magic square* dengan orde *n*=5.

$$c_{17}=2c_{01}+c_{02}+c_{03}+c_{04}+c_{06}-c_{09}+c_{11}-c_{13}+c_{16}-65 \quad (3)$$

$$c_{18}=325-4c_{01}-2c_{02}-2c_{03}-2c_{04}-2c_{06}-2c_{07}-c_{08}-2c_{11}-c_{12}-c_{13}-c_{14}-2c_{16}-2c_{19} \quad (4)$$

$$c_{21}=65-c_{01}-c_{06}-c_{11}-c_{16} \quad (5)$$

$$c_{22}=130-2c_{01}-2c_{02}-c_{03}-c_{04}-c_{06}-c_{07}+c_{09}-c_{11}-c_{12}+c_{13}-c_{16} \quad (6)$$

$$c_{23}=4c_{01}+2c_{02}+c_{03}+2c_{04}+2c_{06}+2c_{07}+2c_{11}+c_{12}+c_{14}+2c_{16}+2c_{19}-260 \quad (7)$$

$$c_{24}=65-c_{04}-c_{09}-c_{14}-c_{19} \quad (8)$$

Dengan persamaan-persamaan ini, maka proses runut-balik akan lebih "tanggap" tanpa harus terpaku dengan elemen terakhir pada suatu baris/kolom/diagonal. Dengan begitu, proses pencarian solusi akan lebih cepat dan efektif.

Sebagai kesimpulan akhir, algoritma runut-balik berpotensi untuk mencari solusi dari persoalan *magic square* untuk orde *n*, dengan *n* bilangan ganjil atau genap selain 2. Namun, optimasi harus dilakukan baik dalam skema pengisian elemen maupun fungsi batas. Dengan begitu, implementasinya dalam program dapat lebih efisien dan efektif.

REFERENSI

[1] Berggren, J. Lennart, and Singer, James. "Magic Square." Microsoft® Student 2007 [DVD]. Redmond, WA: Microsoft Corporation, 2006.

[2] Weisstein, Eric W. "Magic Square." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/MagicSquare.html

[3] Munir, Rinaldi. (2006). Diktat Kuliah IF2251 Srategi Algoritmik. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.