Penerapan Algoritma Genetika pada Permainan *Rubik's Cube*

Abigael Angela Pardede[1], Shanny Avelina Halim[2], Denny Nugrahadi[3]

Laboratorium Ilmu dan Rekayasa Komputasi
Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if14026@students.if.itb.ac.id[1], if14027@students.if.itb.ac.id[2], if14054@students.if.itb.ac.id[3]

Abstrak

Permainan mekanik merupakan salah satu sarana hiburan bagi anak-anak maupun dewasa. Perkembangan teknologi yang begitu pesat menyebabkan banyaknya jenis permainan mekanik yang dijual ke masyarakat. Beberapa dari permainan ini tidak hanya dimanfaatkan untuk hiburan semata, melainkan juga dapat untuk mengasah otak. Permainan-permainan tersebut membutuhkan strategi yang tepat untuk penyelesaian yang cepat. Salah satu permainan mekanik tersebut adalah *Rubik's Cube*. *Rubik's Cube* adalah permainan kubus yang berukuran 3x3x3. Tiap sisi kubus memiliki enam warna yang berbeda. Kubus ini terdiri atas dua puluh enam kubus kecil yang disatukan. Setiap sisi *Rubik's Cube* dapat diputar sebanyak 90^o dan 180^o. Tiap sisi dapat diputar secara acak sehingga tiap sisinya akan memiliki persegi-persegi dengan warna yang berbeda. Permainan selesai ketika setiap sisi kubus menjadi satu kesatuan warna. Untuk menyelesaikan *Rubik's Cube* dapat digunakan berbagai macam algoritma. Pada makalah ini akan dikemukan cara penyelesaian *Rubik's Cube* dengan menggunakan algoritma genetika.

Kata kunci: algoritma genetika, *rubik's cube*, permainan kubus.

1. Pendahuluan

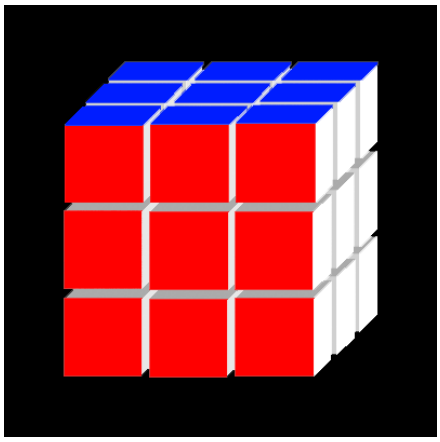
Berbagai macam cara dapat digunakan untuk mengasah kemampuan otak sambil bermain. Salah satunya adalah permainan mekanik yang menggunakan strategi dan kecepatan dalam penyelesaiannya. *Rubik's Cube* adalah permainan yang mulai dikenal pada tahun 1974 dan cukup terkenal sampai sekarang. Untuk menyelesaikan permainan ini dibutuhkan kecepatan, kesabaran dan ketelitian.

2. *Rubik's Cube*

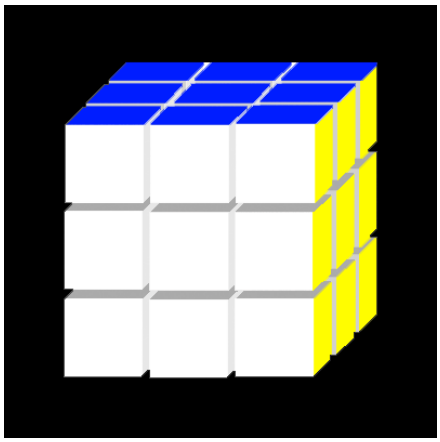
Rubik's Cube adalah permainan kubus 3x3x3 yang memiliki warna yang berbeda pada tiap sisinya. Kubus ini terdiri atas dua puluh enam kubus kecil yang disatukan. Satu kubus kecil di tengah dianggap tidak ada karena berperan sebagai sumbu kubus saat memutar sisinya. Sebuah *Rubik's Cube* dapat memiliki $(8! \times 3^{8-1}) \times (12! \times 2^{12-1})/2 = 43{,}252{,}003{,}274{,}489{,}856{,}000$ posisi warna yang berbeda.

Notasi sisi *Rubik's Cube*

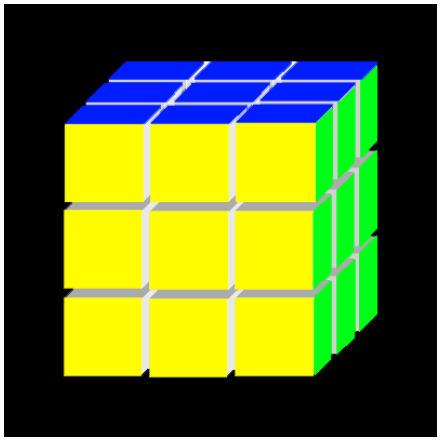
D	Depan; Sisi yang saat ini dipandang
Bk	Belakang; Sisi yang bertolak-belakang dengan sisi Depan
A	Atas; Sisi sebelah atas dari sisi Depan
Bw	Bawah; sisi yang bertolak-belakang dengan sisi Atas
Ki	Kiri; sisi yang yang langsung berhubungan dengan sisi Depan dari arah kiri
Ka	Kanan; sisi yang langsung berhubungan dengan dengan sisi depan dari arah kanan



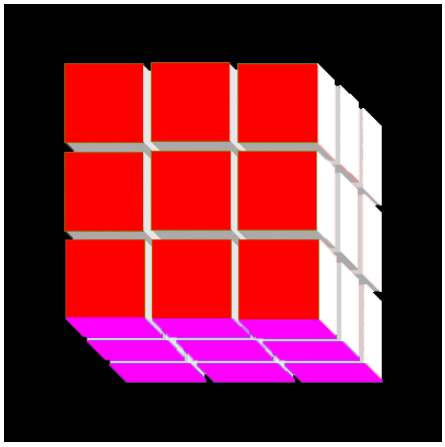
Gambar 1



Gambar 2



Gambar 3



Gambar 4

Gambar 1 menunjukkan tampilan awal sebuah *Rubik's Cube* dengan sisi depan yang berwarna merah (D), sisi atas yang berwarna biru(A), sisi kanan yang berwarna putih(Ka)

Gambar 2 menunjukkan tampilan ketika *Rubik's Cube* pada Gambar 1 diputar ke kiri sebanyak 90 ^{o} dan memperlihatkan sisi belakang kubus yang berwarna kuning (Bk)

Gambar 3 menunjukkan tampilan ketika *Rubik's Cube* pada Gambar 1 diputar ke kanan sebanyak 90 ^{o} dan memperlihatkan sisi kiri kubus yang berwarna hijau (Ki)

Gambar 4 menunjukkan tampilan ketika Rubik's Cube pada Gambar 1 dilihat pada bagian bawah. Sisi bawah kubus berwarna merah muda (Bw).

3. Algoritma Genetika

3.1 Pengertian Algoritma Genetika

Algoritma ini ditemukan di Universitas Michigan, Amerika Serikat oleh John Holland (1975) melalui sebuah penelitian dan dipopulerkan oleh salah satu muridnya, David Goldberg.

Algoritma genetika adalah algoritma yang berusaha menerapkan pemahaman mengenai evolusi alamiah pada tugas-tugas pemecahan-masalah (problem solving). Pendekatan yang diambil oleh algoritma ini adalah dengan menggabungkan secara acak berbagai pilihan solusi terbaik di dalam suatu kumpulan untuk mendapatkan generasi solusi terbaik berikutnya yaitu pada suatu kondisi yang memaksimalkan kecocokannya atau lazim disebut *fitness*. Generasi ini akan merepresentasikan perbaikan-perbaikan pada populasi awalnya. Dengan melakukan proses ini secara berulang, algoritma ini diharapkan dapat mensimulasikan proses evolusioner. Pada akhirnya, akan didapatkan solusi-solusi yang paling tepat bagi permasalahan yang dihadapi.

Untuk menggunakan algoritma genetika, solusi permasalahan direpresentasikan sebagai khromosom. Tiga aspek yang penting untuk penggunaan algoritma genetika:
1. Definisi *fitness function*
2. Definisi dan implementasi representasi genetika
3. Definisi dan implementasi operasi genetika

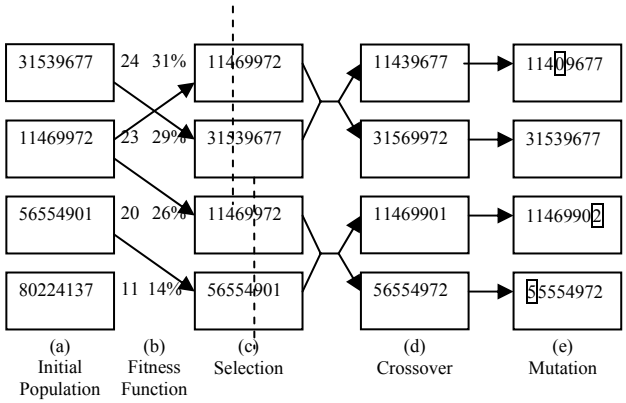
Operasi genetika terdiri atas mutasi dan crossover. Jika ketiga aspek di atas telah didefinisikan, algoritma genetika generik akan bekerja dengan baik.

Tentu saja, algoritma genetika bukanlah solusi terbaik untuk memecahkan segala masalah. Sebagai contoh, metode tradisional telah diatur untuk untuk mencari penyelesaian dari fungsi analitis convex yang "berperilaku baik" yang variabelnya sedikit. Pada kasus-kasus ini, metode berbasis kalkulus lebih unggul dari algoritma genetika karena metode ini

dengan cepat menemukan solusi minimum ketika algoritma genetika masih menganalisa bobot dari populasi awal. Untuk problem-problem ini pengguna harus mengakui fakta dari pengalaman ini dan memakai metode tradisional yang lebih cepat tersebut. Akan tetapi, banyak persoalan realistis yang berada di luar golongan ini. Selain itu, untuk persoalan yang tidak terlalu rumit, banyak cara yang lebih cepat dari algoritma genetika. Jumlah besar dari populasi solusi, yang merupakan keunggulan dari algoritma genetika, juga harus mengakui kekurangannya dalam dalam kecepatan pada sekumpulan komputer yang dipasang secara seri – *fitness function* dari tiap solusi harus dievaluasi. Namun, bila tersedia komputer-komputer yang paralel, tiap prosesor dapat mengevaluasi fungsi yang terpisah pada saat yang bersamaan. Karena itulah, algoritma genetika sangat cocok untuk perhitungan yang paralel.

3.2 Teknik Penggunaan Algoritma Genetika

Algoritma genetika dimulai dengan sekumpulan set status yang dipilih secara random, yang disebut populasi. Algoritma ini mengkombinasikan dua populasi induk. Setiap status atau individual direpresentasikan sebagai sebuah *string*.



Gambar 5

Fitness Function

Setiap individual dievaluasi dengan *fitness function*. Sebuah *fitness function* mengembalikan nilai tertinggi untuk individual yang terbaik. Individu akan diurutkan berdasarkan nilai atau disebut dengan *selection*.

Crossover

Untuk setiap pasang induk, sebuah titik *crossover* akan dipilih secara random dari posisi dalam string. Pada gambar titik *crossover* terletak pada indeks ketiga dalam pasangan pertama dan setelah indeks kelima pada pasangan kedua.

Mutation

Pada mutasi, tiap lokasi menjadi sasaran mutasi acak, dengan probabilitas independen yang kecil. Sebuah digit dimutasikan pada anak pertama, ketiga, dan keempat. Algoritma genetik mengkombinasikan suatu kecenderungan menaik dengan pengeksplorasian acak di antara *thread* pencarian paralel. Keuntungan utamanya,bila ada, datang dari operasi *crossover*. Namun, secara matematis dapat tunjukkan bahwa bila posisi dari kode genetik di permutasikan di awal dengan urutan acak, *crossover* tidak memberikan keunggulan. Secara intuisi, keuntungannya didapat dari kemampuan *crossover* untuk menggabungkan blok-blok huruf berukuran besar yang telah berevolusi secara independen untuk melakukan fungsi yang bermanfaat sehingga dapat menaikkan tingkat *granularity* di mana pencarian dilakukan.

Schema

Teori dari algoritma genetik menjelaskan cara kerjanya menggunakan ide dari suatu *schema*, suatu sub-string di mana beberapa posisi tidak disebutkan. Dapat ditunjukkan bahwa, bila *fitness* rata-rata dari *schema* berada di bawah *mean* maka jumlah instansiasi dari *schema* di dalam populasi akan bertambah seiring bertambaahnya waktu. Jelas sekali bahwa efek ini tidak akan signifikan bila *bit*-bit yang bersebelahan sama sekali tidak berhubungan satu sama sekali, karena akan ada beberapa blok kontigu yang memberikan keuntungan yang konsisten. Algoritma genetik paling efektif dipakai bila *schema-schema* berkorespondensi menjadi komponen berati dari sebuah solusi. Sebagai contoh, bila *string* adalah representasi dari sebuah antena, maka *schema* merepresentasikan komponen-komponen dari antena, seperti *reflector* dan *deflector*. Sebuah komponen yang baik cenderung akan berkerja baik pada rancangan yang berbeda. Ini menunjukkan bahwa penggunaan algoritma genetik yang benar memerlukan rekayasa yang baik pada representasinya.

3.3 Pseudo Code Algoritma Genetika

```
function GeneticAlgorithm(population,
Fitness-FN)→ an individual
{input berupa population, sebuah
kumpulan individual dan Fitness-FN,
sebuah fungsi yang mengukur fitness
suatu individual}
deklarasi
i,x,y : integer

algoritma
repeat
  new_population←empty set
  for i=1 to size(population) do
     x←RandomSelection(population,Fit
     ness-FN)
     y←RandomSelection(population,Fit
     ness-FN)
     child←Reproduce(x,y)
     if(smallRandomProbability) then
     child←mutate(child)
     add child to new_population
  population←new_population
  until some individual is fit enough
or the time has elapsed
return the best individual in
population(based on Fitness-FN)

function Reproduce(x,y : parent
individuals)→individual
deklarasi
algoritma
n←length(x)
c←random number from 1 to n
return Append(substring(x,1,c),
substring(y,c +1,n))
```

4. Penerapan Algoritma Genetika pada permainan *Rubik's Cube*

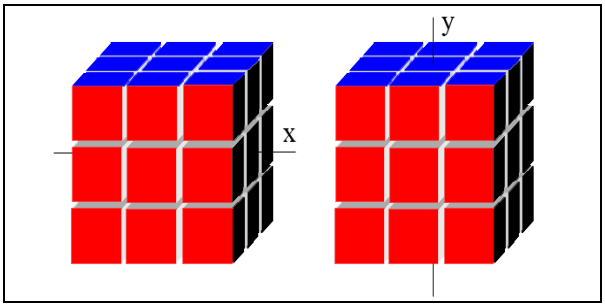
4.1 Komponen Algoritma

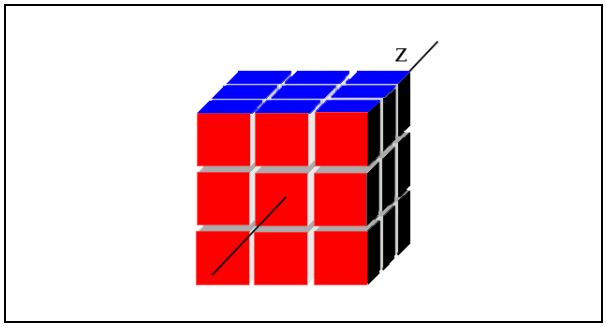
Populasi

Populasi pada persoalan ini terdiri atas notasi pergerakan *Rubik's Cube*. Panjang *string* tiap individu adalah dua puluh sembilan. Artinya, ada maksimal dua puluh sembilan langkah untuk menyelesaikan permainan ini dengan menggunakan algoritma. Hal ini telah dibuktikan oleh Michael Reid pada tahun 1995.

Notasi Pergerakan *Rubik's Cube*

D	Memutar seluruh kubus pada sisi Depan sebesar 90° searah jarum jam
D'	Memutar seluruh kubus pada sisi Depan sebesar 90° berlawanan arah jarum jam
D2	Memutar seluruh kubus pada sisi Depan sebesar 180°
Bk	Memutar seluruh kubus pada sisi Belakang sebesar 90° searah jarum jam
Bk'	Memutar seluruh kubus pada sisi Belakang sebesar 90° berlawanan arah jarum jam
Bk2	Memutar seluruh kubus pada sisi Belakang sebesar 180°
A	Memutar seluruh kubus pada sisi Atas sebesar 90° searah jarum jam
A'	Memutar seluruh kubus pada sisi Atas sebesar 90° berlawanan arah jarum jam
A2	Memutar seluruh kubus pada sisi Atas sebesar 180°
Bw	Memutar seluruh kubus pada sisi Bawah sebesar 90° searah jarum jam
Bw'	Memutar seluruh kubus pada sisi Bawah sebesar 90° berlawanan arah jarum jam
Bw2	Memutar seluruh kubus pada sisi Bawah sebesar 180°
Ki	Memutar seluruh kubus pada sisi Kiri sebesar 90° searah jarum jam
Ki'	Memutar seluruh kubus pada sisi Kiri sebesar 90° berlawanan arah jarum jam
Ki2	Memutar seluruh kubus pada sisi Kiri sebesar 180°
Ka	Memutar seluruh kubus pada sisi Kanan sebesar 90° searah jarum jam
Ka'	Memutar seluruh kubus pada sisi Kanan sebesar 90° berlawanan arah jarum jam
Ka2	Memutar seluruh kubus pada sisi Kanan sebesar 180°





Gambar 6

Gambar 6 menunjukan sumbu putar kubus.

Misalnya, dibangkitkan empat buah induk seperti berikut

Bk2 Ka' A' A' D2 Ka2 Bw' Ki A' D2 Ki' Ka' A Ka2 Bw' Ki A' D2 A' A' D2 Ka2 Ki A' D2 Ki' Ka' Ka' A'

A' D2 Ka2 Bw' Ki Bk2 Ka' A' D2 Ki' Ka' A Ka2 A' D2 Ki' Ka' A D2 Ki' Ka' Ka' A' D2 A' A' D2 Ka2 Ki

D2 Ka2 Bw' Ki Bk2 Ka' A' D2 Ki' D2 Ki' Ka' A D2 Ki' Ka' A D2 Ki' Ka' A' D2 A' A' D2 Ka2 Ki Ka2 A'

Ka' Ka' A' D2 A' Ka2 A' D2 Ki' Bw' Ki A' D2 A' A' D2 Ka' A' D2 A' A' D2 Ka2 Ki Ka2 A' D2 Ki' Ka'

Setiap langkah pergerakan, selalu dilakukan perbandingan. Jika telah ditemukan solusi sebelum akhir dari sebuah individu, maka pencarian dihentikan. Jika belum, pada akhir setiap individu, akan dilakukan penghitungan nilai berdasarkan *fitness function*.

Fitness Function

cost = 48 – x

cost = *fitness function*
x = jumlah posisi persegi pada tiap sisi yang tidak sesuai pada tempatnya (persegi yang memiliki warna yang berbeda pada dengan sumbu sisinya)

Pada *fitness function*, dilakukan pengurangan dari empat puluh delapan posisi warna di tiap sisi. Enam warna yang berada pada sumbu kubus tidak diperhitungkan karena dianggap tidak bergerak.

Bk2 Ka' A' A' D2 Ka2 Bw' Ki A' D2 Ki' Ka' A Ka2 Bw' Ki A' D2 A' A' D2 Ka2 Ki A' D2 Ki' Ka' Ka' A' — 23%

A' D2 Ka2 Bw' Ki Bk2 Ka' A' D2 Ki' Ka' A Ka2 A' D2 Ki' Ka' A D2 Ki' Ka' Ka' A' D2 A' A' D2 Ka2 Ki — 11%

D2 Ka2 Bw' Ki Bk2 Ka' A' D2 Ki' D2 Ki' Ka' A D2 Ki' Ka' A D2 Ki' Ka' A' D2 A' A' D2 Ka2 Ki Ka2 A' — 37%

Ka' Ka' A' D2 A' Ka2 A' D2 Ki' Bw' Ki A' D2 A' A' D2 Ka' A' D2 A' A' D2 Ka2 Ki Ka2 A' D2 Ki' Ka' — 29%

Hasil *Selection*

D2 Ka2 Bw' Ki Bk2 Ka' A' D2 Ki' D2 Ki' Ka' A D2 Ki' Ka' A D2 Ki' Ka' A' D2 A' A' D2 Ka2 Ki Ka2 A' — 37%

Ka' Ka' A' D2 A' Ka2 A' D2 Ki' Bw' Ki A' D2 A' A' D2 Ka' A' D2 A' A' D2 Ka2 Ki Ka2 A' D2 Ki' Ka' — 29%

Bk2 Ka' A' A' D2 Ka2 Bw' Ki A' D2 Ki' Ka' A Ka2 Bw' Ki A' D2 A' A' D2 Ki' Ka' Ka' A' — 23%

A' D2 Ka2 Bw' Ki Bk2 Ka' A' D2 Ki' Ka' A Ka2 A' D2 Ki' Ka' A D2 Ki' Ka' Ka' A' D2 A' A' D2 Ka2 Ki — 11%

Setelah dilakukan penghitungan dengan *fitness function*, induk tersebut diurutkan berdasarkan persen yang terbesar.

Crossover

D2 Ka2 Bw' Ki A' Ka2 A' D2 Ki' Bw' Ki A' D2 A' A' D2 Ka' A' D2 A' A' D2 Ka2 Ki Ka2 A' D2 Ki' Ka'

Ka' Ka' A' D2 Bk2 Ka' A' D2 Ki' D2 Ki' Ka' A D2 Ki' Ka' A D2 Ki' Ka' A' D2 A' A' D2 Ka2 Ki Ka2 A'

Bk2 Ka' A' A' D2 Ka2 Bw' Ki A' D2 Ki' Ka' A Ka2 Bw' Ki A' D2 A' A' D2 Ka2 Ki A' D2 A' D2 Ka2 Ki

A' D2 Ka2 Bw' Ki Bk2 Ka' A' D2 Ki' Ka' A Ka2 A' D2 Ki' Ka' A D2 Ki' Ka' Ka' A' D2 A' Ki' Ka' Ka' A'

Pada langkah berikutnya, dilakukan *cross over* secara berpasangan. Pada aplikasi ini, dilakukan cross over pada indeks kelima dan keduapuluhenam.

Mutation

D2 Ka2 Bw' Ki D2 Ka2 A' D2 Ki' Bw' Ki A' D2 A' A' D2 Ka' A' D2 A' A' D2 Ka2 Ki Ka2 A' D2 Ki' Ka'

Ka' Ka' A' D2 Bk2 Ka' A' Ka2 Ki' D2 Ki' Ka' A D2 Ki' Ka' A D2 Ki' Ka' A' D2 A' A' D2 Ka2 Ki Ka2 A'

Bk2 Ka' A' A' D2 Ka2 Bw' Ki A' D2 Ki' Ka' A A' Bw' Ki A' D2 A' A' D2 Ka2 Ki A' D2 A' D2 Ka2 Ki

A' D2 Ka2 Bw' Ki Bk2 Ka' A' D2 Ki' Ka' A Ka2 A' D2 Ki' Ka' A D2 Ki' Ka' Ka' A' D2 A' Ki' Ka' Ka' Ki'

Setelah itu, dilakukan *mutation* yang dilakukan secara random. Maka, telah terbentuk generasi baru. Hal ini dilakukan berulang-ulang sampai ditemukan solusi terbaik.

5. Kesimpulan dan Saran

Dengan menggunakan algoritma genetika, kita dapat menemukan solusi terbaik untuk permainan *Rubik's Cube*. Algoritma ini membutuhkan tiga aspek penting untuk implementasinya yaitu *fitness function*, representasi genetika dan operasi genetika(*crossover* dan *mutation)*.
Jika ketiga aspek tersebut telah didefinisikan, algoritma genetika generik akan bekerja dengan baik.

Daftar Pustaka

[1] http://wikipedia.org. Diakses tanggal 16 Mei 2006 pukul 20.00 WIB.
[2] *Introduction to Genetic Algorithms*. http:// lancet.mit.edu/~mbwall/presentations/ IntroToGAs/. Diakses tanggal 17 Mei 2006 pukul 17.00 WIB.
[3] Randy L. Haupt . 2004. "*Practical Genetic Algorithms*". A John Wiley & Sons, Inc.
[4] Russell Norvig. 2003. "*Artificial Intelligence*". Pearson Education, Inc.