PENYELESAIAN PERMAINAN *RUBIK'S CUBE* DENGAN METODE ALGORITMA GENETIKA

Zulhaydar Fairozal Akbar[1], Entin Martiana, S.Kom, M.Kom[2], Setiawardhana, S.T, M.T[2], Rizky Yuniar H., S.Kom, M.T[2]
Mahasiswa Jurusan Teknik Informatika[1] , Dosen Pembimbing [2]
Jurusan Teknik Informatika
Politeknik Elektronika Negeri Surabaya
Institut Teknologi Sepuluh Nopember
Kampus PENS-ITS Keputih Sukolilo Surabaya 60111
Telp (+62)31-5947280, 5946114, Fax. (+62)31-5946114
Email : akbar07@student.eepis-its.edu

ABSTRAK

Rubik's Cube merupakan permainan kubus berukuran 3x3 yang paling populer di dunia saat ini. Tiap sisi kubus memiliki enam warna yang berbeda dan tiap sisi dapat diputar sebanyak 90^o dan 180^o. Sehingga *rubik's cube* memiliki 43,252,003,274,489,856,000 kombinasi warna. Banyak cara dan algoritma yang ditemukan oleh beberapa pakar / peneliti untuk menyelesaikan permainan ini termasuk dengan algoritma genetika.

Dalam proyek akhir ini, solusi direpresentasikan sebagai langkah – langkah untuk menyelesaikan *rubik's cube*. Dibutuhkan beberapa komponen dalam algoritma genetika yaitu fungsi fitness, seleksi dan operasi genetika yaitu crossover dan mutasi. Dan adanya pengoptimalan *move sequence* dapat menghasilkan solusi yang optimal karena dapat menghilangkan gen – gen yang *redundant*. Optimal yang dimaksud adalah jumlah langkah yang dibutuhkan untuk menyelesaikan *rubik's cube* sedikit mungkin. Dan setelah ditemukan solusi, maka solusi akan disimulasikan dalam bentuk rubik's cube tiga dimensi.

Berdasarkan hasil uji coba pembangkitan random populasi awal mempengaruhi waktu dan generasi yang dicapai dalam menemukan hasil akhir. Contoh untuk jumlah scramble = 5 waktu yang dibutuhkan lebih cepat dan generasi yang didapat kurang dari 100 generasi dibanding jumlah scramble = 15 dan 30. Dan hasil perbandingan membuktikan bahwa algoritma genetika dapat menemukan solusi yang lebih baik yaitu tidak lebih dari 25 langkah dibandingkan dengan algoritma pemula yang biasanya digunakan oleh manusia untuk menyelesaikan *rubik's cube*.

Kata Kunci : Algoritma genetika, *rubik's cube* , optimal.

1. PENDAHULUAN

Banyak permainan yang dibuat di dunia selama ini. Dan *rubik's cube* merupakan salah satu permainan populer yang dimainkan banyak kalangan di dunia pada saat ini.

Rubik's cube diciptakan oleh seorang professor arsitektur asal negara Hungaria yang bernama Erno Rubik pada tahun 1974. Sampai saat ini permainan *rubik's cube* merupakan mainan puzzle terlaris dalam sejarah. *Rubik's cube* mempunyai 6 sisi. Setiap lapisan terdiri dari sembilan *cube* yang dapat memutar dan lapisan - lapisan dapat tumpang tindih. *Rubik's cube* dapat diputar secara vertical maupun horizontal dan mempunyai $(8! \times 3^{8-1}) \times (12! \times 2^{12-1})/2$ = 43,252,003,274,489,856,000 posisi warna yang berbeda Walaupun kemungkinan posisinya yang besar tapi *rubik's cube* dapat diselesaikan dalam 25 langkah atau kurang(Tomas Rokicki, 2008)[12].

Dengan banyaknya solusi yang ditawarkan oleh para juara dunia seperti Erik, Yu Nakajima, Fridrich, Bob Burton, Pochmann, Chris H, dan banyak lagi sehingga permainan ini menjadi popular kembali.

Pada proyek akhir ini dibangun sebuah aplikasi untuk menyelesaikan permainan *rubik's cube* dengan metode algoritma genetika. Aplikasi ini bertujuan untuk menyelesaikan dari kondisi awal rubik's cube yang teracak hingga kondisi *solved*. Algoritma genetika digunakan untuk mencari solusi yang paling optimal atau dapat dikatakan mencari jumlah langkah yang paling sedikit. Setelah ditemukan solusi, selanjutnya akan diterapkan pada rubik's cube dalam bentuk tiga dimensi yang nantinya user dapat mengikuti langkah – langkah solusi tersebut.

2. PENELITIAN SEBELUMNYA

Dalam pengerjaan proyek akhir ini, penulis mengambil 2 referensi mengenai penerapan algoritma genetika pada *rubik's* cube yaitu dari Abigael, dkk tahun 2006[3] dan Borschbach tahun 2008[4] Untuk menyelesaikan *rubik's cube* dengan algoritma genetika ada beberapa komponen

algoritma genetika yang digunakan yaitu populasi, fungsi fitness, seleksi, *crossover* dan mutasi.

Pada makalah dari Abigael, dkk menjelaskan beberapa komponen algoritma genetika yang digunakan dalam menyelesaikan permainan *rubik's cube* yaitu populasi, fungsi fitness, seleksi, *crossover* dan mutasi. Tapi pada makalah tersebut kurang menjelaskan proses penggunaan algoritma genetika pada rubik's cube hanya gambaran umum. Selain itu tidak menjelaskan mengenai hasil dari penerapan algoritma genetika.

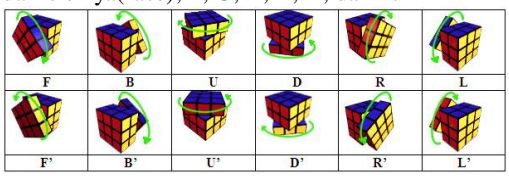
Sedangkan Borschbach menulis untuk menyelesaikan *rubik's cube*, membutuhkan 3 fase. Fase pertama menyelesaikan rubik's cube dengan kondisi 2x2x3, fase kedua membuat menjadi two-generator dan fase terakhir menyelesaikan kondisi two-generator.

Pada proyek akhir ini, mencoba menyelesaikan permainan rubik's cube dengan algoritma genetika hanya 1 fase. Komponen algoritma genetika yang dipakai dalam proyek akhir ini merupakan gabungan dari 2 referensi di atas. Seperti *crossover*, mutasi dan *optimize sequence* diambil dari Borschbach. Sedangkan fungsi *fitness* diambil dari Abigael, dkk.

3. *RUBIK'S CUBE*

Rubik's cube adalah sebuah permainan puzzle mekanik yang berupa kubus 3x3x3 yang memiliki warna yang berbeda pada tiap sisinya. Pemecahan masalah pada *rubik's cube* ini adalah dengan mengorientasikan seluruh warna pada stiker sesuai dengan warna pada tiap-tiap sisi. Yang menjadi penentu warna pada suatu sisi adalah warna pada masing-masing center

Rotasi-rotasi ini dideskripsikan oleh inisial dari sisinya(face); F, U, R, B, D, dan L.



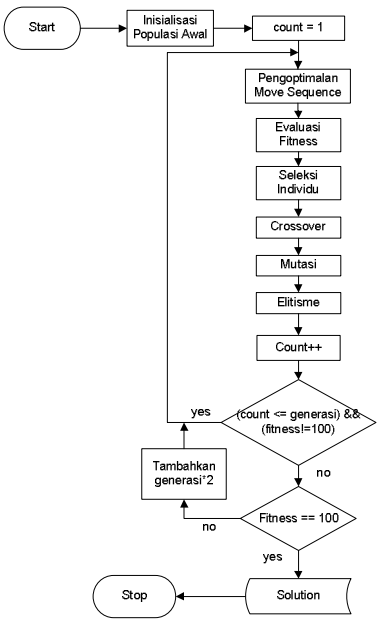
Gambar 1 Notasi Pergerakan *Rubik's cube*

Tabel 1 Notasi Pergerakan

F	Putar seluruh kubus pada sisi Depan sebesar 90^O searah jarum jam
F'	Putar seluruh kubus pada sisi Depan sebesar 90^O berlawanan arah jarum jam
F2	Putar seluruh kubus pada sisi Depan sebesar 180^O
B	Putar seluruh kubus pada sisi Belakang sebesar 90^O searah jarum jam
B'	Putar seluruh kubus pada sisi Belakang sebesar 90^O berlawanan arah jarum jam
B2	Putar seluruh kubus pada sisi Belakang sebesar 180^O
U	Putar seluruh kubus pada sisi Atas sebesar 90^O searah jarum jam
U'	Putar seluruh kubus pada sisi Atas sebesar 90^O berlawanan arah jarum jam
U2	Putar seluruh kubus pada sisi Atas sebesar 180^O
D	Putar seluruh kubus pada sisi Bawah sebesar 90^O searah jarum jam
D'	Putar seluruh kubus pada sisi Bawah sebesar 90^O berlawanan arah jarum jam
D2	Putar seluruh kubus pada sisi Bawah sebesar 180^O
L	Putar seluruh kubus pada sisi Kiri sebesar 90^O searah jarum jam
L'	Putar seluruh kubus pada sisi Kiri sebesar 90^O berlawanan arah jarum jam
L2	Putar seluruh kubus pada sisi Kiri sebesar 180^O
R	Putar seluruh kubus pada sisi Kanan sebesar 90^O searah jarum jam
R'	Putar seluruh kubus pada sisi Kanan sebesar 90^O berlawanan arah jarum jam
R2	Putar seluruh kubus pada sisi Kanan sebesar 180^O

4. PENERAPAN ALGORITMA GENETIKA PADA *RUBIK'S CUBE*

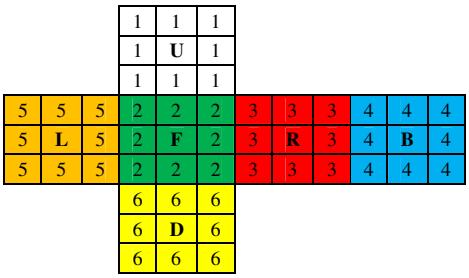
Proses algoritma genetika yang akan dilakukan adalah membangkitkan populasi awal, pengoptimalan *move sequence*, mengevaluasi nilai *fitness*, seleksi individu dengan roulette wheel selection, reproduksi yaitu *crossover* dan mutasi, elitism dan akhirnya menghasilkan populasi baru. Proses ini akan berlangsung sampai generasi yang telah ditentukan atau sampai ketemu solusi yang diharapkan. Untuk lebih jelasnya lihat pada Gambar 2 :



Gambar 2 Flowchart Proses Algoritma Genetika

4.1 Representasi *Rubik's Cube*

Untuk menyederhanakan implementasi, *facelets* dari *cubies* diberi label dengan nomor integer. Terdapat 6 *faces* dan setiap *facelets* pada U (*Up*) diberi nilai 1, F (*Front*) diberi nilai 2, R (*Right*) diberi nilai 3, B (*Back*) diberi nilai 4, L (*Left*) diberi nilai 5 dan D (*Down*) diberi nilai 6. Gambar di bawah menunjukkan representasi internal kubus pada saat belum di acak atau dalam kedaaan *solved* dan dalam keadaan teracak atau *scrambled* :



Gambar 3 Representasi *Rubik's Cube* yang *Solved*

4.2 Representasi Solusi

Data input dan hasil output atau solusi yang dihasilkan oleh perangkat lunak disimpan dalam bentuk angka integer. Angka-angka ini mewakili tiap *turn* atau perputaran yang berjumlah 18 dengan range dari 1 hingga 18. Untuk lebih jelasnya dapat dilihat dari table di bawah ini.

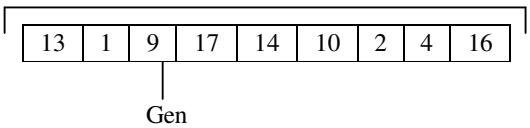
Tabel 2 Representasi Solusi

Clockwise quarter turns		***Counter-Clockwise quarter turns***		***Half turns***	
U	1	U'	2	U2	3
F	4	F'	5	F2	6
R	7	R'	8	R2	9
B	10	B'	11	B2	12
L	13	L'	14	L2	15
D	16	D'	17	D2	18

4.3 Representasi Kromosom

Representasi kromosom adalah solusi dari permasalahan rubik's yang teracak. Dan representasi yang digunakan adalah representasi integer dimana membutuhkan batasan minimum dan batasan maksimum.

Individu (Kromosom)

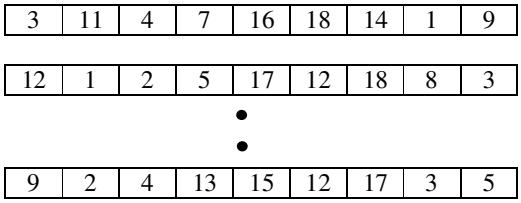


Gambar 4 Representasi Kromosom

4.4 Pembangkitan Populasi Awal

Pembangkitan populasi awal dilakukan secara acak dan jumlah populasi ditentukan oleh masukan user. Semakin besar ukuran populasi, semakin besar pula untuk menemukan solusi yang diharapkan.

Ilustrasi pembangkitan populasi awal adalah sebagai berikut :



Gambar 5 Ilustrasi Populasi Awal

Bilangan random yang dibangkitkan tiap gen dalam satu individu adalah nilai dari representasi solusi yaitu antara 1 – 18. Dan jumlah gen dalam 1 individu yaitu sebanyak 25 yang berarti ada maksimal 25[9] langkah untuk menyelesaikan permainan *rubik's cube*.

4.5 Pengoptimalan *Move Sequence*

Pada fase ini bertujuan untuk mengoptimalkan suatu kromosom (turn sequence) dengan cara menghilangkan gen yang *redundant.* Sehingga jika sudah dioptimasi, diharapkan kromosom akan menjadi lebih pendek.

Contoh : F R D D' R L

Pada *turn sequence* di atas, dapat dilihat terdapat pergerakan D D' yang jika diterapkan pada *rubik's cube* tidak memberikan perubahan posisi sama sekali. Karena setelah melakukan gerakan D (searah jarum jam) kemudian dilanjutkan oleh gerankan D' (berlawanan jarum jam) dimana akan membalikkan keadaan seperti semula. Sehingga setelah dikurangi, *turn sequence* menjadi F R R L. Setelah dikurangi ternyata masih ada pergerakan yang perlu dioptimalkan yaitu R R. Karena R R dapat disederhanakan menjadi R2. Sehingga setelah dikurangi lagi, *turn sequence* akhirnya menjadi F L. Proses tersebut dilakukan terus menerus sampai tidak menemukan kondisi yang *redundant* .

4.6 Fungsi *Fitness*

Setiap langkah pergerakan, selalu dilakukan perhitungan nilai berdasarkan fungsi *fitness*. Dari 1 individu dicoba 25 kali untuk memperoleh index ke berapa dalam menghasilkan *fitness* tertinggi. *Fitness* tertinggi dalam 1 individu tersebutlah yang digunakan dalam perbandingan dengan individu yang laen.

$$FF = \frac{48 - x}{48} 100$$

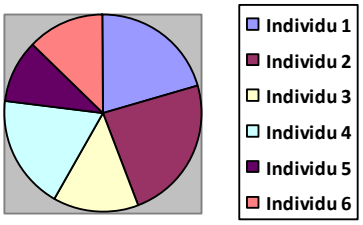
x = Jumlah posisi persegi / *facelet* pada tiap sisi yang tidak sesuai pada tempatnya (*facelet* yang memiliki warna yang berbeda pada dengan sumbu sisinya).

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

4.7 Seleksi

Seleksi dilakukan dalam rangka untuk mendapatkan induk yang baik. Karena induk yang baik akan menghasilkan keturunan yang baik. Sehingga semakin tinggi nilai *fitness* suatu individu maka semakin besar kemungkinannya untuk terpilih. Dalam tahap seleksi ini dilakukan dengan menggunakan mesin *roulette*.

Metode ini didasarkan pada putaran roda mesin *roulette*, dimana setiap kromosom memiliki tempat atau area dalam populasi yang besarnya sesuai dengan nilai *fitness*-nya. Kromosom yang nilai *fitness*-nya lebih besar akan memiliki tempat yang besar pula, selain itu kesempatan untuk terpilih juga semakin besar. Suatu bilangan random dibangkitkan dan individu yang memiliki daerah dalam bilangan random tersebut, maka individu tersebut yang akan terpilih menjadi induk Terdapat 6 individu yang telah di evaluasi *fitness* nya.



Gambar 6 Grafik 6 Individu dalam Mesin *Roulette*

4.8 *Crossover*

Crossover dilakukan terhadap individu yang terpilih dari hasil seleksi. Dan metode yang digunakan untuk *crossover* yaitu menggunakan metode rekombinasi seragam (*uniform crossover*). Berikut proses dari uniform *crossover* :

i. Tentukan nilai probabilitas *crossover* (pCO) . Biasanya nilai pCo antara range 0,5 – 0,95 (Achmad Basuki,2003)[2].
ii. Bangkitkan bilangan random antara 0 – 1, cek apabila nilai random tersebut lebih kecil dari pada probabilitas *crossover*, maka lakukan *crossover* pada pasangan individu. Jika nilai random tersebut lebih besar dari pada probabilitas *crossover* maka individu yang tidak terpilih di *crossover* akan langsung menjadi anak.
iii. Untuk setiap pasangan individu yang akan di *crossover* lakukan berikut :

- Bangkitkan bilangan random 0 atau 1 sesuai jumlah gen pada individu.
- Jika hasilnya 0, anak pertama mendapatkan gen dari induk pertama, sedangkan anak kedua mendapatkan gen dari induk kedua. Dan jika hasil nya 1, anak pertama mendapatkan gen dari induk kedua dan anak kedua mendapatkan gen dari induk pertama.

Individu 1 dan Individu 2								
3	11	4	7	16	18	14	1	9
12	1	2	5	17	12	18	8	3
Random Bit Pattern								
0	1	1	0	0	1	0	1	1
Offspring 1 dan *Offspring* 2								
3	1	2	7	16	12	14	8	3
12	11	4	5	17	18	18	1	9

Gambar 7 *Random Bit Pattern* dibangkitkan untuk menghasilkan Individu Baru / *Offspring*

4.9 Mutasi

Mutasi menggunakan metode pemilihan nilai secara acak. Suatu gen yang terpilih untuk dimutasi nilainya diganti dengan gen baru yang dibangkitkan secara acak dalam interval nilai – nilai gen yang diizinkan, dalam hal ini nilai dari representasi solusi yaitu 1 – 18. Berikut proses dari mutasi :

i. Tentukan nilai probabilitas mutasi (pM) . Biasanya nilai pM antara range 0 – 0,3(Achmad Basuki,2003)[2].
ii. Bangkitkan bilangan random antara 0 – 1, cek apabila nilai random tersebut lebih kecil dari pada probabilitas mutasi, maka lakukan mutasi gen pada individu. Jika nilai random tersebut lebih besar dari pada probabilitas mutasi, maka gen pada individu tersebut tidak perlu di mutasi.
iii. Untuk gen pada individu yang akan di mutasi lakukan berikut :

- Lakukan pengacakan posisi gen yang akan dimutasi. Dengan cara membangkitkan bilangan random sesuai jumlah gen yang ada pada populasi. Misal jumlah gen pada individu ada 12, maka random bilangan antara 1-12.
- Setelah mendapatkan posisi gen yang akan dimutasi, selanjutnya yaitu membangkitkan bilangan representasi solusi yaitu antara 1 – 18 untuk menggantikan nilai gen yang baru dan nilai gen tidak boleh sama dengan nilai gen yang lama.

Individu sebelum di mutasi								
3	1	2	7	16	12	14	8	3
Individu setelah di mutasi								
3	1	2	7	4	12	14	8	3

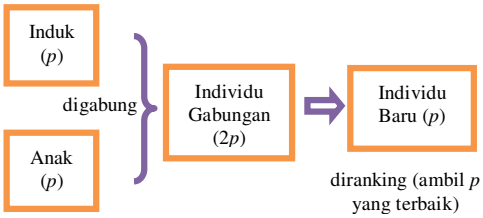
Gambar 8 Mutasi yang terjadi pada Individu

Terpilih posisi gen yang akan dimutasi adalah posisi 5 dan nilai yang akan digantikan gen pada posisi 5 yaitu 4.

4.10 Elitisme

Elitisme digunakan untuk mengikutkan individu – individu induk yang terbaik ke populasi baru dengan cara menggantikan individu – individu turunan yang terjelek, sehingga mencegah kehilangan solusi terbaik.

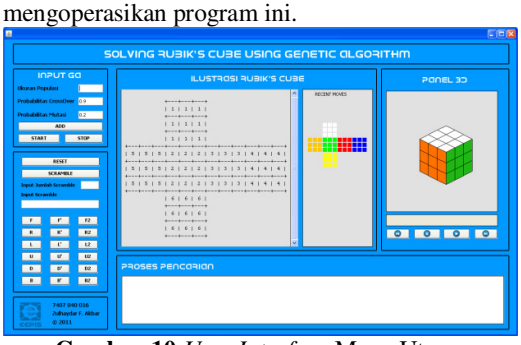
Elitisme yang dipakai adalah sistem ranking. Jika jumlah individu dalam populasi adalah sebanyak p, maka terdapat p individu dan p individu turunan. Dalam sistem ranking populasi induk dan populasi turunan digabung sehingga berjumlah 2p individu. Selanjutnya setiap individu tersebut dihitung nilai *fitness*nya dan diranking yang terbaik. Kemudian diambil p yang terbaik untuk menjadi populasi baru.



Gambar 9 Proses Elitisme

5. *USER INTERFACE*

User Interface dibuat sedemikian rupa agar menarik bagi user dan mempermudah user dalam mengoperasikan program ini.



Gambar 10 *User Interface* Menu Utama

Dalam program ini terdapat 5 panel utama yaitu panel masukan parameter GA, panel rekayasa rubik's cube, panel proses pencarian, panel ilustrasi 2D dan ilustrasi 3D.

6. PENGUJIAN DAN ANALISA

Pada pengujian perangkat lunak ini bertujuan untuk mengetahui seberapa jauh eksekusi perangkat lunak yang telah dibuat dengan tujuan awal dari proyek akhir ini. Selain itu dalam bab ini dilakukan perbandingan antara metode algoritma genetika yang digunakan dalam proyek akhir ini dengan metode algoritma rubik yang sudah ada.

6.1 Uji Coba Algoritma Genetika

Pada uji coba ini membandingkan dengan studi kasus *rubik's cube* yang teracak dengan perubahan nilai parameter probabilitas *crossover*, probabilitas mutasi, jumlah populasi dan perubahan jumlah *scramble* pada *rubik's cube*.

Untuk jumlah *scramble* pada *rubik's cube* yang digunakan dalam uji coba ini dibagi menjadi 3 yaitu sebanyak 5 acakan, 15 acakan dan 30 acakan. Dan masing – masing jumlah *scramble* tersebut digunakan untuk percobaan dengan perubahan nilai probabilitas *crossover* dan probabilitas mutasi. *Scramble* yang digunakan dalam uji coba ini dapat dilihat pada Tabel 3.

Tabel 3 Tabel Scramble untuk Uji Coba

Jumlah *Scramble*	***Scramble Sequence***
5	R D F L2 B'
15	F L D R U B L U2 R' D B2 F2 L R B
30	U L U F' D2 R2 U F D U D2 B B2 U D F' U2 R2 B2 L' U' R' L2 U2 L D' B2 F2 B' R2

Tabel 4 Hasil Uji Coba dengan *Mutation Rate* = 0.1 dan *Crossover* = 0.75

Jumlah Scramble	**Jumlah Populasi**	**Mutasi Rate = 0.1**			
		Cross over Rate = 0.75			
		Solusi di Generasi	**Panjang Solusi**	**Total *Fitness***	***Running Time* (detik)**
5	100	81	5	7090.0	78.734
	200	24	8	11066.0	37.156
	500	4	5	20681.0	44.61
15	100	329	21	7550.0	239.453
	200	336	21	15050.0	477.062
	500	291	20	37550.0	819.125
30	100	873	21	7525.0	802.016
	200	618	21	15050.0	911.062
	500	503	21	37525.0	1.790

Dari hasil uji coba di atas, menghasilkan nilai yang berbeda – beda setiap kali dijalankan. Permasalahan ini disebabkan hampir seluruh proses dalam algoritma genetika mengandalkan teknik random. Dan ditemukan solusi ditentukan juga pada saat pembangkitan populasi awal secara random. Ketika pembangkitan populasi awal, terdapat individu terbaik yang menghasilkan nilai *fitness* yang tinggi , maka untuk menemukan solusi semakin besar dan waktu yang dibutuhkan juga semakin cepat.

Dalam beberapa percobaan, tidak menemukan solusi atau terjebak dalam lokal optimum. Jika terjebak lokal optimum, proses akan

berjalan terus menerus dan kecil kemungkinan untuk menemukan solusi. Sehingga jika pada saat terjebak lokal optimum, maka proses diulangi dari awal yaitu mulai dari pembangkitan populasi awal.

Jumlah *scramble* mempengaruhi pada waktu pencarian solusi dan generasi yang dicapai. Untuk jumlah *scramble* = 5 membutuhkan waktu yang lebih cepat dan generasi yang lebih sedikit daripada jumlah *scramble* = 15 dan 30.

Untuk panjang solusi, jumlah *scramble* = 5 menghasilkan panjang solusi yang tidak jauh berbeda dengan jumlah scramble yaitu 5 langkah. Sedangkan untuk jumlah *scramble* = 15 dan 30 menghasilkan panjang solusi rata – rata 21 langkah.

Jumlah populasi berpengaruh pada waktu pencarian solusi dan generasi yang dicapai. Jumlah populasi yang besar membuat proses pencarian membutuhkan waktu yang semakin lama meskipun generasi yang dicapai menjadi lebih sedikit.

Sedangkan perubahan probabilitas *crossover* dan probabilitas mutasi berpengaruh pada hasil capaian generasi yang didapat. Meskipun hasilnya naik turun, tetepi dapat dilihat pada tabel, semakin besar nilai dari probabilitas *crossover* dan probabilitas mutasi, generasi yang dicapai cenderung semakin sedikit.

6.2 Perbandingan dengan *Rubik's Cube Solver*.

Pada uji coba ini dilakukan perbandingan metode algoritma genetika yang digunakan dalam proyek akhir ini dengan metode algoritma pemula[11], *rubik's cube solver* dari wrongway[10] dan algoritma kociemba[9].

Untuk melakukan uji coba, dibuatlah 5 buah *scramble* yang dijadikan acuan untuk perbandingan, dapat dilihat pada Tabel 6. Dan yang dijadikan acuan adalah hasil dari solusi atau jumlah langkah yang didapat dari setiap metode untuk menyelesaikan *scramble*.

Tabel 5 Tabel *Scramble* untuk Perbandingan

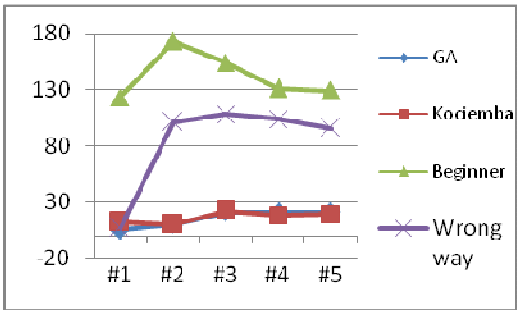
No.	*Scramble*
#1	F' B' D2 U L2
#2	B2 R2 D2 L F' B2 L2 R2 U2 D'
#3	L D' R' F2 B' L2 U2 F L B2 L' B' L2 F' D' B2 U2 R2 L' U
#4	L' D' B' F L U' F' R U' D B2 F' U' F2 B2 D' L' D2 R' F U R U' D B2
#5	B2 L F2 U2 R D2 L' U2 R2 D2 L2 F R' U' L B' L' B' D' F2 L'

Tabel 6 Hasil Perbandingan Beberapa Metode

Metode	Jumlah Langkah				
	#1	#2	#3	#4	#5
GA	5	10	21	21	21
Kociemba	12	10	22	18	19
Beginner	123	173	154	131	129
Wrongway	7	101	108	104	96

Setelah dilakukan uji coba pada beberapa metode, dapat dilihat pada Tabel 7 bahwa hasilnya berbeda – beda. Dari metode algoritma genetika sendiri menghasilkan jumlah langkah tidak lebih dari 21 langkah. Dan metode algoritma genetika menghasilkan jumlah langkah yang tidak jauh beda dengan algoritma kociemba. Algoritma genetika lebih baik daripada 2 metode lainnya yaitu metode pemula dan wrong way, dimana 2 metode tersebut membutuhkan lebih dari 100 langkah untuk menyelesaikan kasus. Meskipun metode wrongway lebih baik daripada metode pemula.

Berikut ini adalah grafik perbandingan 4 metode dengan mencoba 5 kasus yang diberikan di atas.



Gambar 11 Grafik Perbandingan Metode

7. KESIMPULAN

Berdasarkan hasil pengujian yang dilakukan pada proyek akhir ini, maka disimpulkan bahwa:

1. Algoritma Genetika dapat digunakan untuk menyelesaikan permasalahan rubiks cube dengan tipe 3x3x3.
2. Berdasarkan hasil percobaan Algoritma Genetika baik digunakan untuk mencari solusi optimal. Algoritma ini membutuhkan beberapa aspek penting untuk implementasinya yaitu pengoptimalan *move sequence*, *fitness function*, representasi genetika dan operasi genetika (*crossover* dan *mutation*).
3. Jumlah *scramble rubik's cube* sangat berpengaruh dalam menemukan solusi. Semakin banyak jumlah scramble yang di masukkan oleh user, semakin lama waktu yang dibutuhkan untuk menemukan solusi.
4. Pembangkitan individu random pada populasi awal sangat berpengaruh pada waktu yang

dibutuhkan dan jumlah generasi yang dicapai hingga menemukan solusi.

5. Berdasarkan hasil perbandingan, metode algoritma genetika dapat menemukan solusi yang optimum atau lebih baik dibanding dengan algoritma pemula[11] dan *rubik's cube solver* dari wrongway[10] dengan menyelesaikan tidak lebih dari 25 langkah.

8. DAFTAR PUSTAKA

[1]. Martiana, Entin. 2007. *Minggu10 - Algoritma Genetika*. Surabaya : PENS-ITS

[2]. Basuki, Achmad. 2003. *Strategi Menggunakan Algoritma Genetika*. Surabaya : PENS-ITS

[3]. Angela P., Abigael , dkk. 2006. *Penerapan Algoritma Genetika pada Permainan Rubik's Cube*. Bandung : ITB.

[4]. Borschbach, Markus and Grelle, Christian. 2009 *Empirical Benchmarks of a Genetic Algorithm Incorporating Human Strategies*. Germany : University of Applied Sciences.

[5]. Desi H., Murtias. 2007. *Reduksi Data Menggunakan Algoritma Genetika*. Surabaya : PENS-ITS.

[6]. Syamsuddin, Aries. 2004. *Pengenalan Algoritma Genetika*. Ilmu Komputer.

[7]. Sinangjaya, Panji. 2007. *Mengenal Java*. http://panjitapen.wordpress.com/2007/09/30/mengenal-java. . (diakses tanggal 14 Juli 2011).

[8]. *Mengenal Java* . http://java.lyracc.com/belajar/java-untuk-pemula . (diakses tanggal 14 Juli 2011).

[9]. Kociemba, Herbert. 2011. *Cube Explorer 5.00*. http://kociemba.org/cube.htm . (diakses tanggal 14 Juli 2011).

[10]. Dietz, Eric. 2003. *Rubic's Cube Solver ver .505*. http://www.wrongway.org/cube/solve.html . (diakses tanggal 14 Juli 2011).

[11]. Hua, Jiuzhao. 2006. *Rubic's Cube Solver ver .505*. http://www.vrc.freehomepage.com . (diakses tanggal 14 Juli 2011).

[12]. Rokicki, Tomas. 2008. *Twenty-Five Moves Suffice for Rubik's Cube*. http://arxiv.org/abs/0803.3435 . (diakses tanggal 14 Juli 2011).

[13]. *Rubik Formula*. http://www.scribd.com/doc/22142904/Rubiks-Formula . (diakses tanggal 14 Juli 2011).

[14]. Petrus, Lars. 2010. *Solving Rubik's Cube for speed*. http://lar5.com/cube. (diakses tanggal 14 Juli 2011).

[15]. Fridrich ,Jessica. 1997. *Speed Cubing in 17 seconds*. http://www.ws.binghamton.edu/fridrich/cube.html. (diakses tanggal 14 Juli 2011).

[16]. Cason, Kenny. 2009. *Rubik's Cube 3D* . http://ken-soft.com . (diakses tanggal 14 Juli 2011).