APLIKASI ALGORITMA GENETIK PADA PERMAINAN MASTERMIND

Jonathan Marcel T (13507072)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl Ganeca 10 Bandung
e-mail: cel_tum@yahoo.co.id

ABSTRAK

***Mastermind* merupakan salah satu *board game* yang cukup populer. Di dalam permainan mastermind terdapat dua orang pemain, yaitu seorang sebagai *codemaker* dan seorang lagi sebagai *codebreaker*. *Codebreaker* bertujuan untuk menebak kode yang dibuat oleh *codemaker*.**
Algoritma genetik merupakan salah satu pendekatan untuk memecahkan permasalahan pada permainan *mastermind*. Algoritma genetik ini ialah pendekatan komputasional untuk memecahkan permasalahan dimana permasalahan ini dimodelkan sebagai proses evolusi biologis.
Diharapkan dengan menggunakan algoritma genetik kode yang dibuat oleh *codemaker* dapat ditebak secara cepat dan tepat.

Kata kunci: mastermind, algoritma genetik

1. PENDAHULUAN

1.1 Permainan *Mastermind*

Gambar 1. Sebuah Permainan *Mastermind*

Permainan *mastermind* ialah permainan pemecahan kode untuk dua orang pemain. Permainan ini diciptakan oleh Mordecai Meirowitz pada tahun 1970.

Permainan *mastermind* menggunakan:

1. Sebuah *decoding board*, dengan sebuah *shield* untuk menutupi sebuah baris dengan empat buah lubang, sertaduabelas (atau sepuluh atau delapan) baris lainnya dengan empat buah lubang besar sertaempat buah lubang kecil setiap barisnya.
2. *Code pegs* dari enam macam warna yang berbeda, untuk diletakkan pada lubang besar.
3. *Key pegs*, terdiri dari dua warna (hitam/merah dan putih), untuk diletakkan pada lubang kecil.

Terdapat dua orang pemain di dalam permainan ini, seorang sebagai codemaker dan seorang sebagai codebreaker. Codemaker membuat sebuah kode dengan empat buah *code pegs*, dimana duplikasi warna ialah dimungkinkan. Kode ini akan dilindungi oleh *shield* sehingga tidak dapat dilihat oleh *codebreaker*. Sedangkan *codebreaker* akan berusaha menebak kode dengan meletakkan *code peg* pada *decoding* board. Setiap tebakan akan mendapat *feedback* dari *codemaker*, yaitu berupa *key pegs* yang diletakkan pada lubang kecil. Sebuah *key peg* berwarna merah (atau hitam) menandakan sebuah code peg yang diletakkan *codebreaker* adalah benar baik warna maupun posisinya, sedangkan sebuah *key peg* berwarna putih menandakan sebuah *code peg* yang diletakkan *codebreaker* hanya benar warnanya saja. Demikian seterusnya *codebreaker* akan terus mencoba menebak hingga 12 (atau 10 atau 8) *turn*.

1.2 Algoritma Genetik

Algoritma genetik ditemukan oleh John Holland pada tahun 1975 melalui sebuah penelitian dan dipopulerkan oleh salah satu muridnya yaitu David Goldberg.

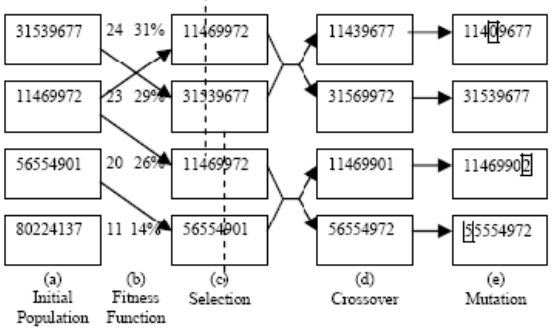
Algoritma genetik berusaha menerapkan pemahaman tentang evolusi alamiah biologis untuk tugas-tugas pemecahan-masalah (problem solving). Pendekatan yang diambil oleh algoritma ini adalah dengan menggabungkan secara acak berbagai pilihan solusi terbaik di dalam suatu kumpulan untuk mendapatkan generasi 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 genetik, solusi permasalahan direpresentasikan sebagai khromosom. Sebuah algoritma genetik memerlukan dua hal:

1. Representasi genetik dari domain solusi. Standar representasi genetik ini berupa *array of bits*.
2. Fitness function untuk mengevaluasi domain solusi

Jika kedua aspek di atas telah didefinisikan, algoritma genetik generik akan bekerja dengan baik.



Gambar 2. Contoh Aplikasi Algoritma Genetik

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

Fitness function

Fitness function digunakan untuk mengevaluasi setiap individual. Sebuah fitness function akan mengembalikan nilai tertinggi untuk individual yang terbaik. Setelah itu setiap individu akan diurutkan berdasarkan nilai atau disebut dengan selection.

Crossover

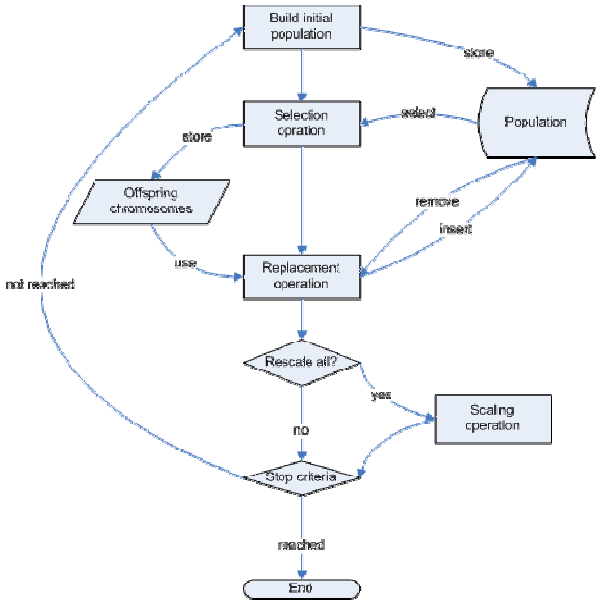
Untuk setiap pasang induk, sebuah titik *crossover* akan dipilih secara random dari posisi dalam string. Pada gambar 2 titik *crossover* terletak pada indeks ketiga dalam pasangan pertama dan setelah indeks kelima pada pasangan kedua.

Mutasi

Di dalam proses 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 diantara thread pencarian paralel. Keuntungan utamanya, bila ada, datang dari operasi *crossover*. Namun, secara matematis dapat tunjukkan bahwa bila posisi dari kode genetik dipermutasikan 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

Berdasarkan teori dari algoritma genetik, cara kerja algortima ini menggunakan suatu schema, yaitu substring dengan posisi yang tidak disebutkan. Berdasarkan penelitian, apabila 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.



Gambar 3. Diagram Alur Algoritma Genetik

2. METODE

2.1 Representasi Genetik untuk Problem *Mastermind*

Sebuah baris terdiri atas P buah *code pegs*. Jumlah warna pada *code pegs* ialah N buah warna. Representasi warna ini dengan bilangan 1, 2, …, N.

Kode yang dibuat oleh *codemaker* direpresentasi sebagai:

Tebakan yang dibuat oleh *codebreaker* direpresentasi sebagai:

$g_1 = (g_{i1}, g_{i2},\ldots, g_{iP}) \in C$, dimana i ialah tebakan ke-i.

Setelah itu feedback yang didapatkan ialah nilai X_i dan Y_i. X_i merepresentasikan jumlah key peg berwarna merah (atau hitam) pada tebakan ke-i, yaitu menghitung jumlah kemunculan $s_P = g_{iP}$, sedangkan Y_i merepresentasikan jumlah key peg berwarna putih pada tebakan ke-i. Pada permainan mastermind, nilai N = 6 dan nilai P = 4, sehingga jumlah seluruh dataset ialah $6^4 = 1296$ kode yang akan dievaluasi untuk mencapai solusi. Sebuah tebakan disebut *eligible* apabila menghasilkan nilai yang sama untuk X_k dan Y_k untuk k buah tebakan. Himpunan tebakan *eligible* ini disebut E_i, yaitu himpunan tebakan *eligible* setelah tebakan ke (i – 1). Permainan akan berakhir pada saat $X_i = P$.

2.2 Teknik Algoritma Genetik untuk Problem *Mastermind*

Sebagai tebakan pertama, gunakan sebuah fixed code yaitu g_1 = (1, 1, 2, 3). Tebakan pertama ini akan mengurangi jumlah tebakan untuk memecahkan kode sebesar 0.02. Untuk langkah-langkah berikutnya, tebakan akan ditentukan oleh algoritma genetik yang sebenarnya.

Pseudocode algroritma ini adalah sebagai berikut:

```
i = 1
Mainkan tebakan pertama g1
Dapatkan feedback X1 dan Y1
While X1 ≠ P do
    i = i + 1
    Tentukan Ei = {} dan h = 1
    Inisialisasi populasi
While (h ≤ maxgen AND Ei ≤ maxsize) do
    Hasilkan populasi baru dengan
    menggunakan crossover, mutasi,
    inversi, dan permutasi
    Kalkulasi fitness
    Tambahkan kombinasi eligible ke
    dalam E
h = h + 1
end while
Mainkan tebakan gi ∈ Ei
Dapatkan feedback Xi dan Yi
End while
```

Pada persoalan ini digunakan angka populasi sebesar 150. Generasi berikutnya dari populasi akan diciptakan melalui 1-point atau 2-point *crossover*. *Crossover* kemudian akan disusul dengan mutasi yang mengganti warna salah satu *code peg* yang ditentukan secara acak dengan warna lain. Selain itu terdapat kemungkinan permutasi, yaitu menukar posisi dua buah *code peg*. Dan yang terakhir ialah inversion, yaitu ketika dua buah *code peg* yang telah diletakkan diambil secara acak, kemudian sekuens warnanya dibalikkan.

Untuk nilai fitness, secara logis dapat diberikan bobot yang lebih besar untuk perbedaan pada *key peg* berwarna merah dibandingkan perbedaan pada *key peg* berwarna putih. Akan tetapi solusi akan lebih cepat diperoleh ketika *key peg* berwarna putih dan merah mendapatkan bobot yang sama. Sehingga untuk perhitungan fitness function dapat diperoleh rumus sebagai berikut:

$$f(\mathbf{c};i) = a\left(\sum_{q-1}^{i}|X'_q(\mathbf{c}) - X_q|\right) + \sum_{q-1}^{i}|Y'_q(\mathbf{c}) - Y_q| + bP(i-1)$$

Dimana a merupakan bobot untuk *key peg* berwarna hitam dan putih, dan b menunjukkan kepentingan relatif dari nilai fitness.

Untuk setiap tebakan, generasi maxgen akan diciptakan oleh algoritma genetik. Setiap tebakan yang *eligible* akan ditambahkan ke himpunan E_i, dan tebakan pada tahap selanjutnya akan dipilih dari himpunan E_i tersebut. Untuk mengurangi kompleksitas algoritma ini, dibuat batasan terhadap maxgen dan ukuran dari E_i.

Akhirnya, algoritma ini harus menentukan tebakan *eligible* manakah di dalam E_i yang akan digunakan untuk tebakan berikutnya. Secara ideal, seseorang akan menggunakan tebakan *eligible* yang akan mengarah ke tebakan *eligible* lainnya setelah tebakan dilakukan. Akan tetapi, pencarian tebakan ini memerlukan cukup banyak waktu sehingga diperlukan pendekatan yang berbeda.

Pendekatan ini akan berbasiskan pada *feedback* yang diberikan oleh *codemaker*, yaitu X_i dan Y_i. Dengan anggapan bahwa *feedback* akan diperoleh dan/atau pada jumlah tebakan *eligible* yang masih tersisa setelah melakukan tebakan. Kualitas dari sebuah tebakan *eligible* dievaluasi dengan cara perkiraan jumlah dari tebakan *eligible* apabila suatu tebakan dilakukan berikutnya. Sehingga kita membutuhkan suatu fungsi seleksi yang akan mengecek jumlah tebakan *eligible* yang tersisa setelah melakukan tebakan. Tebakan berikutnya adalah tebakan *eligible* yang akan menyebabkan tebakan *eligible* lainnya berkurang dengan jumlah yang paling besar.

3. ANALISIS

Teknik algoritma genetik untuk pemecahan persoalan *mastermind* sudah pernah dikodekan dengan kakas Borland Delphi 6 menggunakan Microsoft Visual Studio 2005. Pengujian dilakukan dengan menggunakan komputer Dell Inspiron 1150 dengan prosesor Intel Pentium 4 2.8 GHz dan memori 512 MB RAM, serta sistem operasi Windows XP Professional.

Di dalam eksperimen digunakan nilai P = 4 dan nilai N = 6, hal ini sesuai dengan permainan *mastermind* pada umumnya, namun memungkinkan juga nilai-nilai tersebut

diganti dengan nilai lain apabila ingin melakukan eksperimen yang lain.

Tabel dibawah ini menunjukkan hasil eksperimen dari algoritma genetik ini apabila dibandingkan dengan algoritma lainnya yang sudah ada. Pada makalah ini saya tidak akan membahas tentang algoritma lainnya tersebut.

Algoritma	Rata-rata tebakan	Maksimum tebakan
Shapiro	4.78	8
Swaszek	4.758	9
Rosu	4.64	8
Temporel dan Kovacs	4.64	-
Algoritma genetik	4.39	7

Tabel 1. Hasil Pengujian untuk N = 6 dan P = 4

Berdasarkan hasil pengujian, dapat dilihat bahwa algoritma genetik akan menghasilkan pemecahan dengan jumlah tebakan paling sedikit apabila dibandingkan dengan algoritma lainnya. Salah satu penyebab hal ini dikarenakan algoritma genetik menggunakan fungsi *fitness function* yang dapat digunakan untuk mengevaluasi tebakan pada langkah selanjutnya, sehingga hal ini dapat menyebabkan berkurangnya kemungkinan tebakan *eligible* yang harus dikomputasi.

Untuk tabel berikut ini diberikan hasil pengujian untuk dataset
(nilai P dan N) yang berbeda-beda. Nilai yang diberikan ialah rata-rata tebakan dan rata-rata waktu komputasi CPU.

Algoritma	4;6	6;9	8;12
Swaszek	4.768	7.41	9.2
Rosu	4.64	6.67	8.577
Algoritma Genetik	4.39	6.475	8.366

Tabel 2. Jumlah Rata-Rata Tebakan untuk Dataset Berbeda

Algoritma	4;6	6;9	8;12
Swaszek	0.001 s	1.866 s	30 min 31 s
Rosu	0.001 s	1.282 s	25 min 44 s
Algoritma Genetik	0.614 s	1.284 s	20.571 s

Tabel 3. Rata-Rata Waktu Komputasi CPU untuk Dataset Berbeda

Berdasarkan kedua tabel diatas, dapat dilihat bahwa algoritma genetik akan memberikan jumlah tebakan yang paling sedikit untuk memecahkan *mastermind*, sedangkan berdasarkan parameter waktu komputasi, algoritma ini juga memberikan waktu komputasi yang semakin baik apabila nilai N dan P menjadi lebih besar.

4. KESIMPULAN

1. Algoritma Genetik dapat digunakan untuk memecahkan persoalan pada permainan *mastermind*
2. Pemecahan masalah dengan menggunakan algoritma genetik akan menghasilkan jumlah tebakan yang paling sedikit apabila dibandingkan dengan algoritma lain yang telah ada sebelumnya
3. Waktu komputasi algoritma genetik akan semakin efektif apabila diaplikasikan untuk nilai N dan P yang besar

REFERENSI

[1]Lotte Berghman, "Efficient Solutions for Mastermind using Genetic Algorithms", 5-9

[2]Goddard, "A Computer/Human Mastermind Player using Grids, South African Computer Journal, 30, 3-8

[3]Kooi, "Yet another Mastermind Strategy", ICGA Journal, 28, 13-20

[4]http://www.en.wikipedia.org/wiki/Genetic_algorithm Tanggal Akses 30 Desember 2009

[5]http://www.en.wikipedia.org/wiki/Mastermind_(board_game) Tanggal Akses 30 Desember 2009