Implementasi Algoritma Genetika Dalam Menyelesaikan Sebuah Persoalan Anagram *Scrabble*

Mohammad Gilang Kautzar HW

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
e-mail: if15101@students.if.itb.ac.id

ABSTRAK

Anagram merupakan suatu permainan kata di mana sebuah kata atau frase dihasilkan dengan mengacak urutan huruf-huruf dari suatu kata atau frase lain. Salah satu jenis permainan papan (*board game*) *puzzle* yang paling umum dewasa ini adalah *scrabble*. Pada permainan *Scrabble* ini setiap huruf memiliki poin tertentu, sehingga ada susunan huruf anagram tertentu yang memiliki nilai terbesar. Pada permainan *scrabble* ini, pemain harus mencari susunan huruf (kata) yang paling optimum dari huruf-huruf yang dimilikinya untuk menambah poinnya secara maksimum. Dalam menyelesaikan sebuah anagram *scrabble* ini dapat digunakan berbagai jenis algoritma. Salah satu jenis algoritma yang dapat diimplementasikan adalah algoritma genetika. Dalam tulisan ini akan dibahas bagaimana menerapkan algoritma genetika pada sebuah program untuk mencari susunan kata dari sebuah anagram.

Kata kunci: Algoritma Genetika, *Puzzle*, *Scrabble*, Anagram *Solving*.

1. Pendahuluan

Anagram merupakan sebuah tipe permainan kata yang dahulu populer di Eropa pada abad pertengahan. Seni beranagram diciptakan oleh seorang penyair Yunani Lycophron.

Sebelum era komputerisasi, anagram dibangun menggunakan pulpen dan kertas, dengan memainkan kombinasi huruf dan bereksperimen dengan variasi. Dewasa ini, komputer dapat menghasilkan sebuah metode baru dalam membuat anagram, disebut juga *anagram server*, *anagram solver*, atau *anagrammer*. Metode-metode tadi sering digunakan untuk mencari solusi dari permainan *Crosswords*, *Scrabble*, *Boggle*, dan permainan-permainan kata lain. Ketika seseorang memasukkan sebuah kata atau frase, program atau *server* akan menggunakan sebuah *database* raksasa berisi kata-kata untuk mengeluarkan sebuah daftar yang mengandung setiap kombinasi kata atau frase yang mungkin dari kata atau frase masukan. Kebanyakan *anagram server* dapat mengatur hasil pencarian dengan memasukkan atau mengeluarkan kata tertentu, membatasi panjang kata, atau membatasi jumlah hasil pencarian.

Salah satu permainan paling terkenal yang menggunakan anagram adalah *scrabble*. *Scrabble* merupakan permainan papan (*board game*) di mana pemain harus mencari susunan paling optimum dari huruf-huruf yang dimilikinya. Pada permainan ini setiap huruf memiliki poin masing-masing yang ditentukan berdasarkan frekuensi kemunculannya dalam suatu bahasa.

2. Algoritma Genetika

Algoritma genetika adalah teknik pencarian yang digunakan dalam penghitungan untuk mencari solusi perkiraan maupun benar untuk optimasi dan masalah pencarian. Algoritma ini dikategorikan sebagai algoritma *global search heuristics*. Algoritma ini terinspirasi dari prinsip dari genetika dan seleksi alam (teori evolusi Darwin). 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 diimplementasikan sebagai simulasi komputer di mana sebuah populasi dari representasi abstrak (disebut kromosom, genotipe, atau genom) kandidat solusi (disebut individu, makhluk, atau fenotipe). Biasanya, solusi direpresentasikan dalam biner sebagai string yang terdiri dari 0 dan 1, namun penggunaan metode *encoding* lain juga mungkin dipakai.

Berbeda dengan teknik pencarian konvensional, algoritma genetika bermula dari himpunan solusi yang dihasilkan secara acak. Kromosom-kromosom berevolusi dalam suatu proses iterasi yang berkelanjutan yang disebut generasi. Pada setiap generasi, kromosom dievaluasi berdasarkan suatu fungsi evaluasi. Setelah beberapa generasi maka algoritma genetika akan konvergen pada kromosom terbaik, yang diharapkan merupakan solusi optimal.

Pertama kali, sebelum algoritma genetika dijalankan, maka perlu didefinisikan fungsi *fitness* sebagai masalah yang ingin dioptimalkan. Jika nilai *fitness* semakin besar, maka sistem yang dihasilkan semakin baik. Fungsi fitness ditentukan dengan metode heuristik.

Algoritma genetika sangat tepat digunakan untuk penyelesaian masalah optimasi yang kompleks dan sukar diselesaikan dengan menggunakan metode konvensional. Sebagaimana halnya proses evolusi di alam, suatu algoritma genetika yang sederhana umumnya terdiri dari tiga operasi yaitu: operasi reproduksi, operasi persilangan (*crossover*), dan operasi mutasi. Struktur umum dari suatu algoritma genetika dapat didefinisikan dengan langkah-langkah sebagai berikut:

- Membangkitkan populasi awal secara random.
- Membentuk generasi baru dengan menggunakan tiga operasi diatas secara berulang-ulang sehingga diperoleh kromosom yang cukup untuk membentuk generasi baru sebagai representasi dari solusi baru.
- Evolusi solusi yang akan mengevaluasi setiap populasi dengan menghitung nilai fitness setiap kromosom hingga kriteria berhenti terpenuhi. Bila kriteria berhenti belum terpenuhi maka akan dibentuk lagi generasi baru dengan mengulangi langkah 2. beberapa kriteria berhenti yang sering digunakan antara lain:
    - berhenti pada generasi tertentu
    - berhenti setelah dalam beberapa generasi berturut-turut didapatkan nilai fitness tertinggi/terendah (tergantung persoalan) tidak berubah.
    - berhenti bila dalam n generasi berikutnya tidak diperoleh nilai fitness yang lebih tinggi/rendah.

2.1. *Pseudo-code*

Berikut ini adalah fungsi umum (dasar) dari Algoritma genetika:

```
Function GeneticAlgorithm (populasi,
fitness-FN) → individu
{input berupa populasi, sebuah kumpulan
individu dan kebuagaran-FN, sebuah fungsi
yang mengukur suatu fitness individu}
Deklarasi
i,x,y : integer
Algoritma
repeat
   new_population←empty set
   for i=1 to size(population) do
      x←RandomSelection(populasi,fitness
      -FN)
      y←RandomSelection(populasi,fitness-
      FN)
      child←Reproduce(x,y)
   if(smallRandomProbability) then
      anak←mutate(anak)
      add anak to new_population
      populasi ← new_population
until (beberapa individu cukup fit) or
(waktu habis)
return individu terbaik dalam populasi
(berdasarkan fitness-FN)

Function Reproduce(x,y : parent_indiviual
s) → individual
Algoritma
n ← length(x)
c ← random angka dari 1 sampai n
return          Append(substring(x,1,c),
substring(y,c +1,n))
```

3. Persoalan Anagram *Solving* dengan implementasi *Bruteforce*

Cara termudah dalam menyelesaikan sebuah persoalan anagram adalah dengan menggunakan algoritma *bruteforce*. Hal yang dilakukan yaitu mencoba semua kombinasi huruf dari kata atau frase yang menjadi persoalan. Tetapi hal tersebut memerlukan waktu yang sangat lama karena banyaknya jumlah kombinasi yang ada. Sebagai contoh, jumlah kombinasi huruf dari sebuah kata "*tequila*" yang berjumlah 7 huruf berarti:

$$7!=5040$$

Kemudian dari angka tersebut, dicari susunan mana yang paling besar nilainya. Namun, angka 5040 tersebut merupakan jumlah yang sangat besar untuk suatu algoritma pencarian (tidak mangkus).

4. Persoalan Anagram *Solving* dengan Implementasi Algoritma Genetika

Karena algoritma *bruteforce* tidak mangkus untuk digunakan, maka program akan menggunakan implementasi algoritma genetika.

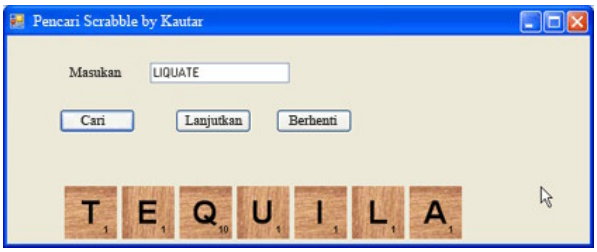
4.1. Desain

Anagram genom hanya terdiri dari satu buah set huruf. Algoritma terdiri dari sebuah populasi dari satu set genom

scrabble. Kemudian genom-genom tersebut dimutasikan dan disilang (*crossover*) di setiap generasi sampai program menuelesaikan anagram dan genom memiliki fitness yang sempurna. Karena semua kemungkinan huruf pada genom telah diketahui, kita tidak perlu lagi memutasikan kata dengan menggunakan huruf-huruf baru. Kita cukup memutasikan posisi dari huruf pada setiap genom. Persilangan juga akan mengubah posisi dari huruf-huruf, namun hanya dalam satu genome tersebut, sehingga tidak ada huruf yang hilang atau terduplikasi.

Program memiliki antarmuka. Ada tiga tombol, satu buah kotak teks, dan tempat jawaban. Tombol cari akan menjalankan prosedur cari yang mencari solusi anagram sampai menemukan jawaban yang sesuai dari sebuah *database* berisi kata-kata (*first found*). Tombol lanjutkan akan mencari solusi anagram seluruhnya (sampai habis).

Sketsa antarmuka dari program Anagram *Solver* adalah sebagai berikut:



Gambar 1. Desain Interface dengan 3 buah tombol dan contoh

4.2. *Pseudo-code implementasi*

Dalam implementasi algoritma genetika terhadap penyelesaian anagram *scrabble*, fungsi fitness menjadi sederhana karena kita hanya menerima fitness sempurna (*perfect fitness*). Berikut ini akan dijelaskan fungsi dan prosedur yang digunakan dalam penyelesaian masalah tersebut.

Fungsi berikut menghitung nilai fitness berdasar pada kebenaran pengejaan.

```
Function HitungGAScrabbleBoard(input
static) → Nilaifitness
Deklarasi
Real: Nilaifitness
String: word

Algoritma
Nilaifitness ← 0
Word ← ToString()

if (_speller.Testword(word)) then
      Nilaifitness ← TheArray.Count
else
      Nilaifitness ← 0.01
endif
return Nilaifitness
```

Program menjalankan algoritma dalam dua mode: cari (*first find*) dan lanjutkan (kontinu). Kedua algoritma dijalankan dalam *thread* terpisah. Prosedur berikut adalah mode kontinu yang berjalan apabila tombol kontinu ditekan.

```
Prosedur BtnLanjutkan (input static)
Deklarasi
_stop = boolean
oThread = Thread

Algoritma
ScrabbleGenom.HurufAwal=txtAnagram.txt
_stop ← false
oThread ← mulai prosedur GAKontinu
oThread.Start()

Prosedur GAKontinu (input static)

for i ← 0 to 100 do
   {ambil generasi berikut dari genom
   yang mendekati fitness yang
   diinginkan}
   populasi.NextGeneration()

   {Tampilkan nilai teratas genom
   setiap 50 generasi}
   if (i mod 50 = 0) then
      _answerText ←
      populasi.GetTopGenome().ToString(
      )
      _solved = true
   endif
endfor

{berhenti jika tombol stop ditekan}
if (_stop == true){
      _stop ← false
      break
}
```

Sebagai contoh, kita ingin mencari seluruh anagram dari kata "*rose*". Maka kita akan menjalankan mode kontinu ini dan mengambil gen dengan fitness terbaik yang telah dihasilkan. Hasil akan terlihat seperti berikut:

Genom 1: rose = 4
Genom 2: rose = 4
Genom 3: sore = 4
Genom 4: ores = 4
Genom 5: rose = 4
Genom 6: sore = 4

Genom 7: roes = 4
Genom 8: rose = 4
Genom 9: rose = 4

5. Kesimpulan

Algoritma genetika ini sangat tepat untuk diimplementaskan pada program penyelesaian anagram, khususnya pada permainan *scrabble*. Karena tadi telah ditunjukkan bahwa algoritma ini dapat menyelesaikan suatu persoalan yang apabila dikerjakan secara manual (otak manusia) sangat lama. Jadi, algoritma ini cukup mangkus untuk digunakan dibanding *brute-force*.

REFERENSI

[1] Munir, Rinaldi. *Diktat Kuliah IF2251 Strategi Algoritmik*, Program Studi Teknik Informatika ITB, 2005

[2] Haupt, Randy L and Sue Ellen Haupt. *Practical Genetic Algorithms*, 2nd edition, John Wiley & Sons, Inc, Hoboken, New jersey, 2004.

[3] Wikipedia, Genetic Algorithm, http://en.wikipedia.org/wiki/Genetic_algorithm, diakses tanggal 21 Mei 2007 pukul 15.00

[4] AA Pardede, SA Halim, D Nugrahadi. Penerapan algoritma genetika pada permainan Rubik's *cube*. Program Studi Teknik Informatika ITB, 2006