Penggunaan *Genetic Programming* untuk Mencari Rumus Suatu Deret Bilangan

Khandar William, NIM 13506022
Sekolah Teknik Elektro dan Informatika, ITB, Bandung 40132
e-mail: if16022@students.if.itb.ac.id
website: www.captainkuro.co.nr

Absrtact **–** *Topik dari makalah ini adalah salah perkembangan Matematika Diskrit, yaitu Genetic Programming. Makalah ini membahas salah satu penerapan genetic programming dalam bidang teori bilangan, yaitu mencari rumus dari suatu deret bilangan. Genetic programming adalah pengembangan dari genetic algorithm dengan solusi berupa program komputer. Source code yang penulis buat dalam bahasa C disertasakan di bagian lampiran makalah ini.*

Kata Kunci*: genetic algorithm, genetic programming, deret bilangan*

1. PENDAHULUAN

Salah satu pokok bahasan yang kita pelajari dalam bidang studi Matematika SMA adalah Deret Bilangan. Deret bilangan adalah barisan bilangan yang dibentuk dari rumus tertentu. Notasi yang digunakan adalah U_n untuk suku ke-n. Contohnya: 1, 4, 9, 16, ... adalah deret bilangan kuadrat dengan rumus $U_n = n^2$. Contoh lain: rumus $U_n = 3n+4$ akan membentuk deret 7, 10, 13, ...

Permasalahannya adalah jika diberikan suatu deret bilangan, apakah dapat ditentukan rumus pembentuknya dan dengan cara apa. Jawabannya adalah dapat, yaitu dengan cara menebak. Kalau deret tersebut berupa suatu deret aritmatika atau geometri mungkin masih dapat ditentukan secara manual, tetapi jika sudah melibatkan logaritma atau rekursif yang kompleks akan menjadi mustahil untuk bisa dikerjakan oleh manusia.

Untuk menjawab kemustahilan ini, komputer dimanfaatkan. Kendalanya adalah bagaimana komputer dapat menebak karena komputer tidak memiliki otak seperti manusia. Solusinya adalah menggunakan metode heuristik, salah satunya *genetic programming*.

Dahulu, istilah komputer yang berevolusi hanya terdapat pada cerita-cerita fiksi seperti Terminator dan Matrix. Namun, dengan kemajuan teknologi sekarang, konsep *artificial intelligence* telah memungkinkan cerita-cerita fiksi tersebut nyata. Salah satu bagian dari *artificial intelligence* ini adalah *genetic algorithm*.

Ada sangat banyak penerapan dari *genetic algorithm*, terutama dalam bidang *problem solving*. Salah satu hasil dari pengembangan *genetic algorithm* adalah melahirkan konsep *genetic programming*.

Pada makalah ini akan dijelaskan *genetic algorithm* yang menjadi dasar dari *genetic programming*, setelah itu akan dijelaskan tentang *genetic programming* itu sendiri. Lalu, akan dibahas mengenai penerapan *genetic programming* untuk mencari rumus deret bilangan dan diakhiri dengan kesimpulan.

2. *GENETIC ALGORITHM*

Terlepas dari benar tidaknya teori evolusi Darwin, *genetic algorithm* menggunakan konsep evolusi ini dan mengaplikasikannya untuk memecahkan masalah.

John Holland dalam bukunya *Adaptation in Natural and Artificial Systems* (1975, 1992) menjelaskan bahwa proses evolusi dapat dipakai untuk memecahkan berbagai masalah dengan suatu teknik paralel yang kini dikenal sebagai *genetic algorithm* [2].

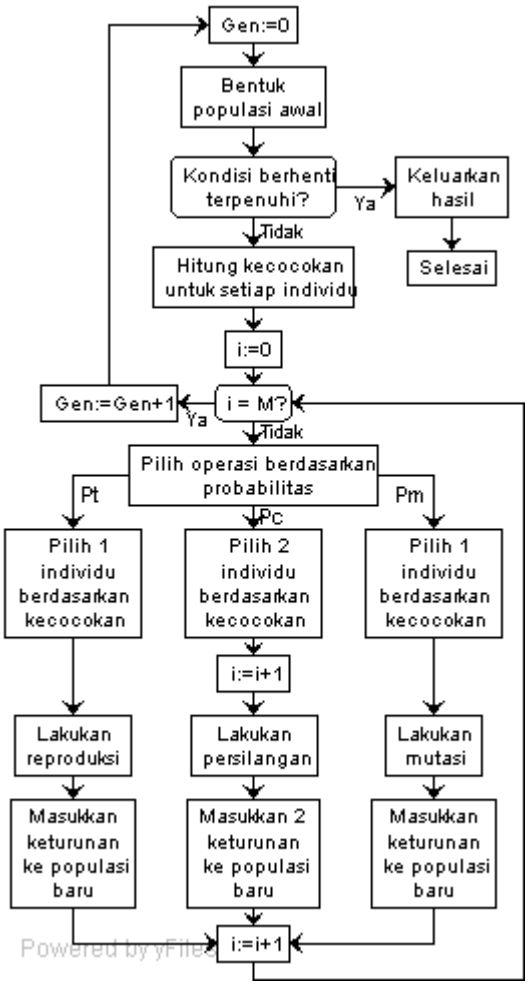
Ketika diberikan masalah, *genetic algorithm* membentuk suatu kumpulan/populasi calon solusi. *Genetic algorithm* memandang calon solusi ini sebagai individu yang dapat berkembang biak menghasilkan keturunan. Dengan prinsip “orang tua yang bagus menghasilkan keturunan yang lebih bagus”, maka di setiap calon solusi ini diberikan suatu *fitness value*/nilai kecocokan untuk menunjukkan seberapa baguskah calon solusi tersebut. Semakin tinggi nilai kecocokannya, semakin tinggi juga kemungkinannya untuk bertahan hidup dan menghasilkan keturunan (menggambarkan proses seleksi alam). Contohnya, pada *Knapsack Problem* nilai kecocokan dapat didefinisikan sebagai nilai dari barang-barang yang dimasukkan dalam tas.

Pada prakteknya, calon solusi ini biasa direpresentasikan sebagai *string*/untaian nilai (menggambarkan kromosom yang terdiri dari untaian gen). Nilai ini dapat berupa angka, huruf, atau kata di mana tiap-tiap nilai ini berkorespondensi dengan variabel tertentu. Contohnya, pada masalah *Travelling*

Salesman Problem calon solusi dapat digambarkan sebagai untaian angka yang menunjukkan urutan kota yang dikunjungi, jadi individu `[3 5 2 1 4]` mempunyai arti kota pertama = kota 3, kota kedua = kota 5, dst.

Agar suatu masalah dapat diselesaikan dengan *genetic algorithm*, perlu ditentukan hal-hal berikut:

1. representasi solusi dalam untaian nilai, apakah akan digunakan untaian bit atau angka dan apa makna dari tiap nilai tersebut,
2. fungsi untuk menghitung nilai kecocokan, berdasarkan apa kita menentukan suatu individu itu bagus atau tidak,
3. parameter untuk mengontrol *genetic algorithm*, yaitu jumlah populasi, probabilitas persilangan, dan probabilitas mutasi,
4. kondisi berhenti, yaitu kapan solusi yang diperoleh dianggap cukup.



Gambar 1. *Flowchart* dari *genetic algorithm.*

Pseudo-code dari *genetic algorithm* adalah sebagai berikut:

1. Bentuk populasi awal secara acak.
2. Lakukan proses perkembangbiakan berikut pada semua individu untuk membentuk populasi baru:
   a. Hitung nilai kecocokan tiap-tiap individu.
   b. Berdasarkan nilai kecocokannya, pilih individu yang akan mengalami reproduksi, *crossover*/persilangan, dan mutasi.
3. Ganti populasi yang lama dengan populasi yang baru dibentuk, jika kondisi berhenti belum tercapai, kembali ke langkah 2.

Reproduksi adalah menghasilkan keturunan yang sama persis dengan orang tuanya. Probabilitas dari suatu indvidu untuk mengalami reproduksi tergantung dari nilai kecocokannya.

Crossover/persilangan adalah memilih 2 individu sebagai orang tua lalu melakukan pertukaran sebagian untaian menghasilkan 2 keturunan baru. Contohnya, jika individu A `[10011010]` disilangkan dengan individu B `[00010111]` maka pertama-tama ditentukan dulu bagian mana dari untaian yang akan ditukarkan, misalkan bagian 1 sampai 3 yang akan ditukarkan maka

```
A [10011010] \/ C [00011010]
B [00010111] /\ D [10010111]
```

menghasilkan keturunan C `[00011010]` dan D `[10010111]`.

Mutasi adalah menghasilkan keturunan dari suatu individu dengan mengganti satu atau lebih nilai dalam untaiannya. Contohnya, jika individu A `[10011010]` mengalami mutasi maka pertama-tama ditentukan dulu nilai mana yang akan mengalami mutasi, misalnya nilai ke-2 yang mengalami mutasi maka

```
A [10011010] ==> A' [11011010]
```

menghasilkan keturunan A' `[11011010]`. Mutasi jika terlalu sering terjadi akan mengubah *genetic algorithm* menjadi *random searching* biasa.

Solusi dari permasalahan tersebut adalah individu dengan nilai kecocokan terbaik sepanjang "sejarah", yaitu dari generasi awal sampai generasi terakhir ketika kondisi berhenti tercapai. Solusi yang diperoleh dari *genetic algorithm* belum tentu merupakan solusi terbaik, tetapi hanya solusi sangat baik. Meskipun begitu, dengan menggunakan parameter (jumlah populasi, probabilitas mutasi, probabilitas persilangan) yang tepat, solusi yang diperoleh dapat sangat mendekati solusi terbaik dengan *runnung-time* yang lebih singkat daripada algoritma lain.

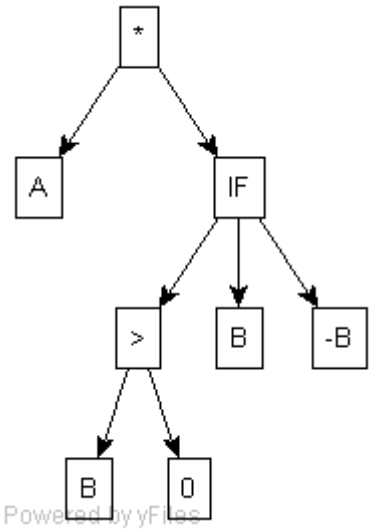
3. *GENETIC PROGRAMMING*

Genetic programming hadir untuk menjawab pertanyaan yang diutarakan oleh Arthur Samuel: "*How can computers be made to do what needs to be done, without being told exactly how to do it?*" atau dengan kata lain bagaimana komputer belajar untuk menyelesaikan masalah tanpa diprogram secara eksplisit [2].

Secara sederhana, *genetic programming* adalah *genetic algorithm* di mana tiap-tiap individu calon solusi tidak berupa untaian nilai, tetapi berupa sebuah program komputer. Program komputer ini digambarkan sebagai sebuah pohon berakar di mana setiap simpul adalah fungsi atau parameter fungsi tersebut. Contohnya jika diberikan sebuah program LISP:

```
(* A (IF (> B 0) B -B))
```

maka representasinya adalah



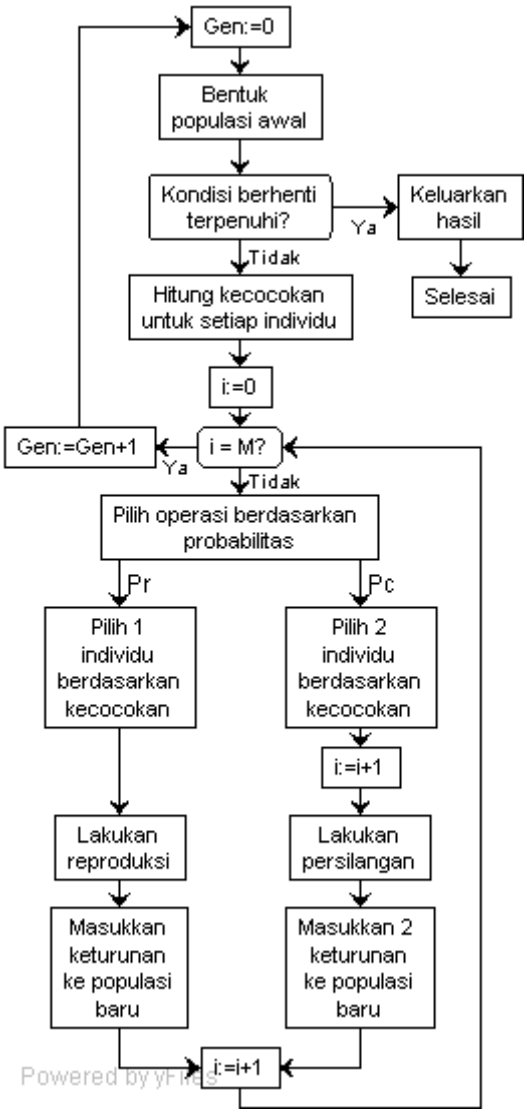
Gambar 2. Contoh representasi program dalam pohon.

Seperti *genetic algorithm*, *genetic programming* juga memerlukan beberapa hal untuk didefinisikan dahulu:

1. *terminal*, yaitu variabel atau konstanta yang terdapat pada solusi dan menjadi daun-daun pada pohon representasi program,
2. fungsi-fungsi primitif yang dapat dipakai, sebagai pembangun program yang akan dicari,
3. fungsi untuk menghitung nilai kecocokan, berdasarkan apa calon program ini dikatakaan bagus atau tidak,
4. parameter untuk mengontrol *genetic programming*, sama seperti pada *genetic algorithm*,
5. kondisi berhenti, juga sama seperti *genetic algorithm*.

Pseudo-code dari *genetic programming* adalah sebagai berikut:

1. Bentuk populasi awal secara acak berdasarkan cakupan *terminal* dan fungsi primitif yang sudah didefinisikan.
2. Lakukan proses perkembangbiakan berikut pada semua individu untuk membentuk populasi baru:
   a. Hitung nilai kecocokan dengan menjalankan program tersebut.
   b. Berdasarkan nilai kecocokannya, pilih individu yang akan mengalami reproduksi dan persilangan.
3. Ganti populasi yang lama dengan yang baru, jika kondisi berhenti belum terpenuhi, kembali ke langkah 2.



Gambar 3. *Flowchart* dari *genetic programming*.

Reproduksi adalah proses menghasilkan keturunan yang sama persis dengan individu orang tuanya, atau dengan kata lain menggandakan dirinya ke populasi yang baru. Reproduksi dilakukan untuk mempertahankan individu terbaik yang ada supaya tidak "musnah". Semakin tinggi nilai kecocokan suatu individu, semakin tinggi kemungkinannya untuk bereproduksi.

Crossover/persilangan adalah proses menukar *subprogram*/upaprogram dari dua individu orang tua menghasilkan dua keturunan baru hasil persilangan tersebut. Prosesnya mirip dengan *genetic algorithm*, tetapi yang ditukar bukanlah deretan nilai melainkan *subprogram* atau yang direpresentasikan sebagai *subtree*/upapohon. Contohnya jika ada dua individu A dan B sebagai orang tua yang merupakan program dalam bahasa LISP.

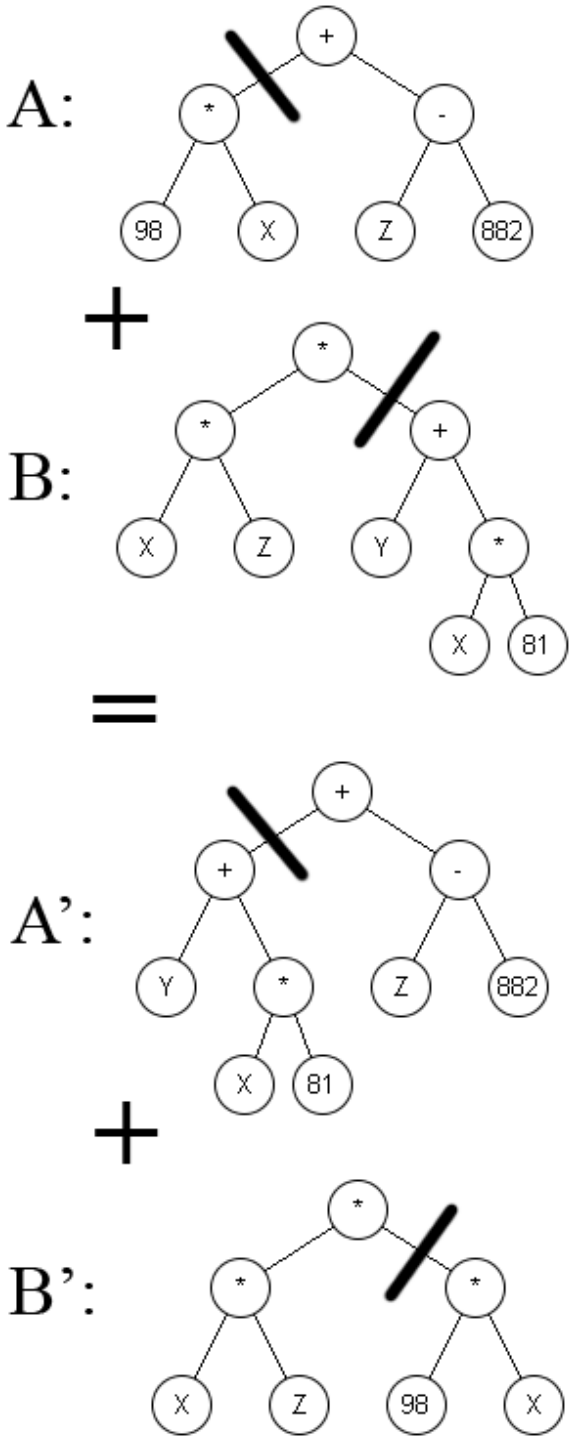
```
A: (+ (* 98 X) (- Z 882))
B: (* (* X Z) (+ Y (* X 81)))
```

Pertama akan dipilih titik persilangan secara acak pada individu masing-masing dan tidak harus terletak pada titik yang sama karena mungkin bentuk pohonnya tidak sama. Sebagai contoh terpilih titik persilangan sebagai berikut:

```
A: (+ (* 98 X) (- Z 882))
B: (* (* X Z) (+ Y (* X 81)))
```

Maka hasil persilangannya adalah:

```
A': (+ (+ Y (* X 81)) (- Z 882))
B': (* (* X Z) (* 98 X))
```



Gambar 4. Proses persilangan A dan B.

Selain reproduksi dan persilangan, sebenarnya masih ada lagi operasi yang dapat dilakukan dalam *genetic programming*, di antaranya: *mutation*, *permutation*, *editing*, *encapsulation*, dan *decimation.*

Sama seperti *genetic algorithm*, solusi yang dihasilkan pun tidak menjamin solusi terbaik, hanya mendekati terbaik. Namun, dapat seberapa dekat solusi ini dengan solusi terbaik itulah yang menjadi alasan *genetic programming* ini dipakai.

4. PENERAPAN *GENETIC PROGRAMMING*

Masalah yang dihadapi adalah menentukan rumus pembangun deret bilangan dari sampel deret yang diberikan. Contohnya, jika diberikan sampel 10 deret pertama: 0, 4, 18, 48, 100, 180, 294, 448, 648, 900 maka program harus menemukan rumus yang membangun deret ini, untuk contoh ini digunakan $U_n = n^3 - n^2$.

Agar lebih mudah merepresentasikan individunya, digunakan *syntax* bahasa pemrograman LISP untuk menuliskan calon-calon solusi. Alasan menggunakan *syntax* LISP adalah formatnya yang dapat langsung menggambarkan srtuktur pohon, seperti pada contoh yang sudah dipaparkan di atas.

Langkah pertama dalam menyelesaikan masalah ini adalah mendefinisikan beberapa hal berikut:

1. *Terminal*, dalam hal ini variabel yang terdapat pada rumus U_n adalah variabel n. Jadi *terminal* `T = {n}`.
   Lalu apakah tidak perlu bagi kita memasukkan konstanta angka ke dalam *terminal*? Apakah nanti tidak ada koefisien pada rumus tersebut? Jawabannya koefisien akan ada tetapi akan dengan sendirinya terbentuk, contohnya jika program memperoleh rumus:
   `(/ x (+ x x)) = x / 2`
   Lihat, koefisien ½ terbentuk tanpa perlu memasukkan konstanta. Dengan cara seperti inilah program akan menemukan koefisien.
2. Fungsi-fungsi primitif, dalam hal ini semua operator aritmatika. Namun, sebagai contoh, dibatasi deret yang ada hanya terdiri dari operasi aritmatika biasa. Jadi fungsi-fungsi primitif
   `F = {+, -, *, /}`
3. Fungsi untuk menghitung nilai kecocokan, dalam hal ini adalah satu dibagi jumlah selisih dari ouput program dengan output yang seharusnya ditambah angka satu.
   `f(x) = 1/(1 + ∑ selisih)`
   Artinya semakin bagus program tersebut, semakin mendekati angka satu nilai kecocokannya.
4. Parameter pengontrol, untuk contoh dimisalkan jumlah populasi = 50, probabilitas persilangan = 80%, dan probabilitas reproduksi = 80%.

5. Kondisi berhenti, yaitu jika terdapat program yang mencapai nilai kecocokan = satu.

Lalu algoritma untuk menyelesaikan masalah ini adalah sebagai berikut:

1. Bentuk suatu populasi awal secara acak, agar tidak mubazir, tidak boleh ada individu yang kembar. Setelah itu hitung nilai kecocokan masing-masing individu.
   (baris 72 s/d 83 pada *source code*).
2. Lakukan proses berikut untuk membentuk generasi selanjutnya:
   a. Berdasarkan probabilitas persilangan, lakukan persilangan terhadap dua individu yang terpilih menurut nilai kecocokannya.
      (baris 92 s/d 98 pada *source code*).
   b. Berdasarkan probabilitas reproduksi, lakukan proses reproduksi terhadap satu individu yang terpilih menurut nilai kecocokannya.
      (baris 99 s/d 107 pada *source code*).
   c. Lakukan langkah a dan b sampai jumlah keturunan yang dihasilkan sama dengan jumlah populasi.
      (pengkondisian di baris 90 pada *source code*).
3. Hitung nilai kecocokan masing-masing individu. Ganti populasi yang lama dengan yang baru saja terbentuk, jika kondisi berhenti belum tercapai, kembali ke langkah 2.
   (baris 109 s/d112, pengkondisian di baris 84 pada *source code*).

Pada pembahasan ini, dipakai program dari *source code* yang disertakan pada lampiran. *Source code* ini di-*compile* menggunakan GCC pada sistem operasi Linux (untuk sistem operasi Windows perlu disesuaikan sedikit karena terdapat perbedaan rentang tipe `int` pada *compiler* GCC di Linux dan Windows).

Implementasi *genetic programming* pada program ini adalah sebagai berikut:

1. Parameter pengontrol berada di baris 7 s/d 11 *source code*.
2. Fungsi yang menjadi calon solusi direalisasikan sebagai *binary tree*/pohon biner di mana simpul-simpulnya dapat berupa `{+, -, *, /, N}` yaitu fungsi primitif dan *terminal*.
3. Individu direalisasikan sebagai *record*/tipe bentukan yang berisi *fitness value*/nilai kecocokan dan sebuah *binary tree*.
4. Populasi direalisasikan sebagai *array*/larik yang terdiri atas individu.
5. Fungsi untuk menghitung nilai kecocokan terdapat pada baris 261 s/d 273 yaitu fungsi `GetFitness()`.

Program ini diberi *input* berupa deret bilangan dengan rumus $U_n = n^2 - n$. Jadi *input*-nya adalah:

```
0
2
6
12
20
30
42
56
72
90
110
132
156
182
210
240
272
306
342
380
```

Hasil akhir yang didapat oleh program ini adalah:

```
Generasi: 55; Best: 1.000000; Total:
1.034044; Average: 0.020681
(/ (/ (- (/ N (- N (* (/ N (* (/ (- N N ) (-
N N ) ) (* N (/ N N ) ) ) ) (- N N ) ) ) ) N
) (/ N (+ (* (- (- N N ) (/ N N ) ) (* N (* N
N ) ) ) (* (/ N (* (/ (- N N ) (- N N ) ) (*
N (/ N N ) ) ) ) (- N N ) ) ) ) ) N )
```

Artinya solusi dengan nilai kecocokan = 1 ditemukan pada generasi ke-55 dengan rumus:

```
(/
   (/
      (-
         (/
            N
            (-
               N
               (*
                  (/
                     N
                     (*
                        (/
                           (- N N )
                           (- N N )
                        )
                        (*
                           N
                           (/ N N )
                        )
                     )
                  )
                  (- N N )
               )
            )
         )
         N
      )
      (/
         N
         (+
            (*
               (-
                  (- N N )
                  (/ N N )
```

```
                    )
                    (*
                       N
                       (* N N )
                    )
                 )
                 (*
                    (/
                       N
                       (*
                          (/
                             (- N N )
                             (- N N )
                          )
                          (*
                             N
                             (/ N N )
                          )
                       )
                    )
                    (- N N )
                 )
              )
           )
        )
     )
     N
)
```

Yang sebenarnya setara dengan

```
(- (* N N) N)
```

(Perlu diperhatikan bahwa pada program ini pembagian dengan angka 0 akan menghasilkan nilai 1 supaya tidak terjadi kesalahan *division by zero*).

Berdasarkan hasil ini, dapat disimpulkan bahwa *genetic programming* memang berhasil menemukan rumus deretnya, tetapi dengan bentuk yang sangat rumit dibandingkan rumus sebenarnya meskipun jika disederhanakan, kedua rumus tersebut identik. Hal ini dikarenakan fakta bahwa bagaimana pun juga komputer tidak berpikir, dia hanya mengerjakan apa yang disuruh. Sehingga kalau tidak diberitahu cara menyederhanakan persamaan, maka dia tidak akan menyederhanakannya.

5. KESIMPULAN

Genetic programming adalah sebuah terobosan di dalam bidang komputasi. *Genetic programming* diciptakan untuk menjawab pertanyaan "*How can computers be made to do what needs to be done, without being told exactly how to do it?*" yang dulu hanya didengar melalui cerita-cerita fiksi. Namun, dengan adanya *genetic programming* maka bukan hal yang mustahil lagi bagi komputer untuk "berpikir" sendiri.

Penerapan *genetic programming* pun banyak, mulai dari hal yang sederhana seperti mencari suatu rumus Matematika sampai membuat sebuah program yang kompleks. Salah satu penerapan *genetic programming* yang dibahas makalah ini adalah mencari rumus suatu deret bilangan.

Untuk merealisasikan *genetic programming* dapat digunakan bahasa pemrograman apa pun, baik fungsional maupun prosedural. Pada makalah ini disertakan *source code* dalam bahasa C yang merupakan implementasi sederhana dari *genetic programming* untuk memecahkan persoalan mencari rumus deret bilangan.

Setelah menjalankan contoh *input* dan mendapatkan contoh *output*, dapat dilihat bahwa komputer berhasil menebak apa rumus deret tersebut tanpa diberitahu rumus yang sebenarnya. Di sinilah terlihat peran dari *genetic programming*, yaitu membuat komputer mengambil keputusan sendiri. Sudah dijelaskan di atas bahwa *genetic programming* tidak menjamin akan mendapatkan hasil yang terbaik, hanya sangat baik, seperti hasil yang diperoleh dalam percobaan ini.

Penggunaan *genetic programming* yang dibahas pada makalah ini merupakan salah satu yang paling dasar. Penerapan *genetic programming* yang sesungguhnya jauh lebih kompleks. Contohnya "*Software That Writes Software*"[9].

DAFTAR REFERENSI

[1] Darrell Whitley, "*A Genetic Algorithm Tutorial*", Statistics and Computing 4 (1994) 65-85.

[2] John R. Koza, "*Survey of Genetic Algorithms and Genetic Programming*", Proceedings of the Wescon 1995.

[3] John R. Koza, "*Genetic Programming*", Encyclopedia of Computer Science and Technology 39 (1998) 29-43.

[4] John R. Koza, *Genetic Programming On the Programming of Computers by Means of Natural Selection*, MIT Press, 1998.

[5] Mitchell Melanie, *An Introduction to Genetic Algorithms*, MIT Press, 1999.

[6] Introduction to genetic algorithms with Java applets. http://cs.felk.cvut.cz/~xobitko/ga/. Tanggal akses: 26 Desember 2007.

[7] Wikipedia | Genetic Algorithm. http://en.wikipedia.org/wiki/Genetic_algorithm. Tanggal akses: 27 Desember 2007.

[8] Wikipedia | Genetic Programming. http://en.wikipedia.org/wiki/Genetic_programming. Tanggal akses: 27 Desember 2007.

[9] Salon Technology | Software That Writes Software. http://www.salon.com/tech/feature/1999/08/10/genetic_programming. Tanggal akses: 27 Desember 2007.

LAMPIRAN

Source code yang penulis buat dapat diunduh dari pranala berikut:
http://students.if.itb.ac.id/~if16022/genprog.c
http://captainkuro.freehostia.com/genprog.c