

# KAJIAN GENETIC ALGORITHM

## DALAM PENYELESAIAN TSP

Arief Widhiyasa

13505126

Program Studi Teknik Informatika

Institut Teknologi Bandung

Jl. Ganesha 10, Bandung

E-mail : arief.clawford@gmail.com

### Abstrak

Makalah ini membahas tentang Genetic Algorithm (GA). Genetic Algorithm merupakan suatu algoritma yang pendekatannya dilakukan melalui kajian sistem genetika kehidupan. Dimana berbagai kegiatan genetik seperti persilangan dan mutasi merupakan salah satu operasi pada Genetic Algorithm ini. Memanfaatkan teknik randomisasi dan persilangan serta mutasi, genetika algoritma menjadi suatu teknik heuristik yang dapat menemukan suatu solusi dengan cukup cepat. Walaupun ada kemungkinan dimana solusi yang ditemukan terjebak pada suatu optimum lokal, bukan merupakan optimum global, namun algoritma ini masih merupakan salah satu algoritma yang baik dalam menyelesaikan masalah-masalah NP dimana waktu eksekusinya yang cukup singkat sangat membantu. Dalam makalah ini akan dikaji mengenai Genetik Algoritma beserta implementasinya dalam menyelesaikan TSP yang merupakan masalah NP, dalam waktu eksekusi yang cukup singkat.

**Kata-kata kunci:** *Genetic, TSP, Selection, Crossover, Mutation, Population.*

### 1. PENDAHULUAN

Sampai saat ini TSP masih merupakan masalah yang belum dapat diselesaikan dengan benar-benar optimal. Berbagai pendekatan telah dilakukan dari berbagai sudut. Namun masih juga belum dapat menyelesaikan masalah tersebut dengan optimal.

TSP yang merupakan singkatan dari Travelling Salesman/Salesperson Problem yang merupakan suatu masalah yang cukup terkenal pada teori grafika. Secara umum deskripsi permasalahannya adalah sebagai berikut :

‘Diberikan sejumlah kota dan jarak antar kota. Tentukan sirkuit terpendek yang harus dilalui oleh seorang pedagang bila pedagang itu berangkat dari sebuah kota asal dan menyinggahi setiap kota tepat satu kali dan kembali lagi ke kota asal keberangkatan. Kota dapat dinyatakan sebagai simpul graf, sedangkan sisi menyatakan jalan yang menghubungkan antar dua buah kota. Bobot pada sisi menyatakan jarak antara dua buah kota.’ (Dikutip dari Diktat Kuliah Matematika Diskrit, Ir. Rinaldi Munir, M.T. halaman VIII-48)

Masalah ini tidak lain adalah menentukan sirkuit hamilton yang memiliki bobot minimum pada sebuah graf terhubung. Sehingga peraturan ‘harus kembali ke tempat semula’

sama sekali tidak mengubah kompleksitas dari masalah ini.

Bila ditinjau dari segi sejarah, masalah ini pertama kali dikemukakan sejak jaman komputer belum ada. Sekitar abad ke-19 Sir William Rowan Hamilton, seorang ahli Matematika Irlandia dan Thomas Penyngton Kirkman, seorang ahli Matematika Inggris mengemukakan masalah ini. Diskusi dari hasil pekerjaan mereka dapat dilihat pada buku Graph Theory (1736-1936) karangan N.L. Biggs, E.K. Lloyd, dan R.J. Wilson: Clarendon Press, Oxford, 1976. Bentuk umum dari permasalahan TSP ini muncul dan dipelajari mulai tahun 1930 oleh Karl Menger di Vienna and Harvard. Masalah itu kemudian lebih lanjut lagi dibahas oleh Hassler Whitney dan Merrill Flood di Princeton. Pembahasan lebih detail mengenai hubungan kedua pembelajaran diatas dan perkembangan TSP dapat ditemukan di *paper* karya Alexander Schrijver, yang berjudul “On The History of Combinatorial Optimization”.

Dilihat dari persoalannya solusi paling pasti adalah dengan mencoba semua permutasinya (ordered combinations) kemudian melihat yang mana yang paling pendek jaraknya. Jadi intinya menggunakan teknik *brute force* dimana semua kemungkinan dicoba. Jadi jika ada  $n$  buah kota,

akan terjadi  $n!$  kali proses perhitungan atau  $O(n!)$  dan ini sangat tidak mangkus (efisien).

Jika menggunakan Dynamic Programming, masalah ini dapat diselesaikan kira-kira dalam  $O(2^n)$ . Walaupun ini eksponensial, tapi masih jauh lebih baik dari  $O(n!)$ . Secara sederhana, Dynamic Programming merupakan teknik dimana dilakukan penyimpanan data untuk sesuatu yang sudah pernah dihitung dan digunakan lebih dari satu kali.

Melihat hasil dari 2 buah cara diatas yang masih menghasilkan runtime yang sangat lama, masalah ini digolongkan ke dalam NP-Hard. NP adalah singkatan dari 'Non-deterministic Polynomial time' merupakan suatu kumpulan problem yang bisa diselesaikan secara sempurna hanya dengan runtime polynomial tergantung terhadap banyaknya input pada sebuah non-deterministic Turing machine. Bila masalah ini merupakan masalah memutuskan, misalnya 'diberikan rute dengan cost  $x$ , apakah ada suatu rute yang lebih singkat dari  $x$ ?', masalah baru ini merupakan NP-Complete. NP-Complete adalah suatu kumpulan masalah-masalah NP-Hard dimana solusi reduksinya juga masih merupakan NP.

Untuk memecahkan masalah ini, berbagai pendekatan harus dilakukan. Pendekatan standar yang dilakukan ketika menemui suatu NP problem adalah :

1. Membuat suatu algoritma untuk mencari solusi yang optimal global. (cara ini akan bekerja cepat untuk ukuran masalah yang kecil).
2. Mencari suatu 'suboptimal' atau algoritma heuristic. Algoritma yang akan menghasilkan suatu solusi yang baik, walaupun tidak bisa dibuktikan optimum global.
3. Mencari kasus-kasus spesial dimana dapat diimplementasikan heuristic yang lebih baik atau lebih pasti.

Melihat dari hasil pendekatan di atas, pada akhirnya algoritma yang digunakan adalah heuristic. Heuristic merupakan suatu metoda pendekatan yang untuk memecahkan suatu masalah dan tidak peduli apakah solusinya bisa dibuktikan benar atau tidak. Tetapi secara umum bisa menghasilkan solusi yang baik atau menyelesaikan masalah yang lebih sederhana yang mengandung atau berhubungan dengan masalah yang lebih kompleks.

Saat ini, Genetic Algorithm merupakan salah satu algoritma yang dicoba untuk mencapai solusi optimal TSP, walaupun sampai saat ini masih belum merupakan optimum global, algoritma ini sudah menghasilkan solusi yang sangat mendekati solusi optimum global.

## 2. PENGENALAN GENETIC ALGORITHM

Fisika, Biologi, Ekonomi atau Sosiologi seringkali harus berurusan dengan masalah optimisasi klasik. Ekonomi sepertinya merupakan salah satu spesialis dalam hal tersebut<sup>1</sup>. Secara umum, dapat juga dikatakan sebagian besar pengembangan-pengembangan di bidang Matematika pada abad ke-18 berhadapan dengan masalah tersebut.

Metoda analisis murni telah membuktikan effisensinya. Walaupun begitu, metoda tersebut tetap memiliki kelemahan yang tidak dapat dihilangkan, contohnya : Kenyataan jarangkali mematuhi fungsi-fungsi diferensial tingkat tinggi yang dipertunjukkan para profesor<sup>2</sup>.

Metode lainnya, penggabungan analisis matematika dengan pencarian acak telah muncul. Dapat dibayangkan ketika kita menyebarkan robot-robot kecil di sebuah pada daerah pegunungan. Robot-robot tersebut dapat mengikuti jalan-jalan setapak yang mereka temui. Ketika sebuah robot mencapai puncak, dia akan menyatakan bahwa dia telah menemukan puncak optimum. Metoda ini akan sangat efisien, tapi tidak ada bukti bahwa optimum yang ditemukan adalah optimum untuk seluruh pegunungan (optimum global), setial robot bisa saja menemukan hanya optimum lokal. Metoda jenis ini hanya bekerja pada ruang lingkup pencarian yang diperkecil.

Lalu apakah hubungan antara metoda optimasi tersebut dengan dunia buatan (*artificial*) dan bahkan dengan TSP?

### 2.1. Evolusi dan optimisasi.

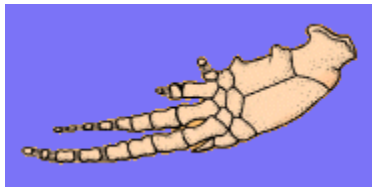
Sekarang kita akan mencoba kembali ke 45 juta tahun yang lalu mengamati seekor Basilosaurus :



Basilosaurus

Basilosaurus dapat dikatakan prototipe dari seekor paus. Panjangnya sekitar 15 meter dan beratnya kurang lebih 5 ton. Ia juga memiliki kepala yang setengah bebas (*quasi-independent*) dan kaki belakang. Ia bergerak dalam air menggunakan gerakan yang bergelombang dan berburu mangsa-mangsa kecil<sup>3</sup>.

Pergerakan di dalam elemen yang kental (air) sangatlah susah dan memerlukan usaha yang cukup keras. Ia harus memiliki banyak energy untuk bergerak dan mengontrol lengkungan gerakannya. Namun bagian depan anggota tubuh basilosaurus tidak benar-benar teradaptasi untuk berenang<sup>4</sup>. Untuk itu, dua buah fenomena evolusi harus muncul, yaitu : memendeknya lengan serta pengontrolan pergerakan oleh bahu, dan pemanjangan jari-jari yang menjadi dasar struktur *flipper*-nya.



Tursiops flipper

Gambar diatas menunjukkan bahwa 2 jari dari lumba-lumba standar yang memanjang dibandingkan jari-jari lainnya.

Basilosaurus merupakan hewan pemburu, oleh karena itu, ia harus cepat dan tepat sasaran. Seiring dengan waktu, subyek-subyek yang diteliti mulai muncul dengan lengan yang lebih pendek dan jari yang memanjang. Sehingga mereka bisa bergerak lebih cepat dan tepat dari sebelumnya, akibatnya mereka bisa hidup lebih lama dan memiliki banyak keturunan.

Sementara itu, peningkatan-peningkatan lainnya juga muncul menuju bentuk aerodinamis umum seperti integrasi kepala dengan badan, perubahan pada wajah, penguatan sirip belakang dan lain-lain. Sehingga pada akhirnya menghasilkan sebuah individu yang beradaptasi dengan

sempurna dengan berbagai tekanan pada lingkungan perairan.

Proses adaptasi ini, proses optimasi morfologi ini sangat sempurna sehingga jaman sekarang, kesamaan antara paus, lumba-lumba dan hiu sangat banyak. Tetapi kemunculan ikan *cartilaginous* pertama (Chondrichthyen) berasal pada jaman Devonian (400 juta tahun lalu), jauh sebelum pembentukan mamalia pertama dari perubahan bentuk Cetacean<sup>5</sup>.

Oleh karena itu, mekanisme Darwin juga merupakan proses optimisasi<sup>6</sup>, optimasi hidrodinamik untuk ikan dan binatang perairan lainnya, tubuh aerodinamis untuk pterodactyls, burung atau kelelawar. Observasi ini merupakan basis dari *Genetic Algorithm*.

## 2.2. Evolusi dan Genetic Algorithms

John Holland, dari University of Michigan memulai karyanya pada genetic algorithm pada awal tahun 1960. Hasil yang pertama dikemukakan ke publik adalah *Adaptation in Natural and Artificial System*<sup>7</sup> pada 1975.

Holland mempunyai tujuan ganda, yaitu untuk meningkatkan pengertian di bidang proses adaptasi alami dan untuk mendesain sistem buatan yang memiliki ciri-ciri mirip dengan sistem alami<sup>8</sup>.

Ide dasarnya seperti ini. Kumpulan gen dari sebuah populasi yg diberikan akan memberikan suatu solusi, atau solusi yang lebih baik, atau memberikan masalah adaptasi. Solusi ini tidak "active" karena kombinasi genetic dimana solusi itu bergantung terbagi menjadi beberapa subjek. Hanya dengan asosiasi dari berbagai elemen gen yang bisa menuju ke solusi. Untuk lebih mudahnya, kita bisa mengambil contoh dari perubahan bentuk tubuh basilosaurus yaitu pemendekan lengan dan pemanjangan jari. Perubahan tersebut dikontrol oleh 2 gen. Sebelumnya tidak ada basilosaurus yang memiliki 2 buah gen tersebut namun seiring dengan reproduksi dan persilangan (kawin), muncul suatu kombinasi genetic baru, dan akhirnya muncul makhluk baru yang mewarisi gen-gen terbaik dari induknya. Lengannya kini bisa digunakan sebagai 'flipper'.


Metoda Holland ini cukup efektif karena ia tidak hanya mendeklarasikan peran sebuah mutasi (mutasi sangat jarang meningkatkan ke'mangkus'an algoritma), tetapi ia juga

menggunakan rekombinasi genetik (persilangan).<sup>2</sup> Dengan menggunakan teknik persilangan ini pada solusi parsial, akan meningkatkan kapabilitas suatu algoritma untuk mendekati atau bahkan menemukan solusi optimum globalnya.





### 2.3. Penggunaan Genetic Algorithm

Sebagai contoh, kita akan mencoba memasuki sebuah dunia genetik yang disederhanakan. Kromosom merepresentasikan sebuah grup dari fungsi-fungsi yang saling berhubungan. Gen merepresentasikan aktivasi atau deaktivasi dari sebuah fungsi.

Sekarang kita coba mengamati kumpulan global genetik dari 4 basilosaurus yang ada di dunia ini. Kita akan menganggap kromosom yang merepresentasikan panjang dari tubuh bagian depan. Panjang dari lengan dan panjang dari jari akan direpresentasikan dengan 4 buah gen. Dua buah gen pertama akan merepresentasikan panjang lengan dan sisanya merepresentasikan panjang jari.

Dalam bentuk representasi genom kita, gambar lingkaran pada kotak berwarna latar biru menyatakan aktivasi fungsi pemanjangan dan gambar silang pada kotak berwarna latar hijau menyatakan deaktivasinya. Sehingga genom yang ideal (Lengan pendek dan jari panjang) akan berbentuk seperti : .

Misalkan kumpulan genetik pada populasi kita adalah seperti ini :

Subject	Genome
A	
B	
C	
D	

Bisa dilihat jikalau subyek A dan B sangat menyerupai sifat-sifat nenek moyangnya. Dan sebaliknya, D sangat mendekati optimum, ia hanya butuh pemanjangan jari sedikit lagi.













Ini adalah dunia yang sangat kejam sehingga kemampuan untuk bergerak dan berpindah tempat merupakan modal pertama untuk bertahan hidup dan bereproduksi. Tidak akan ada betina yang dengan mudah mau mengawini basilosaurus yang terlihat seperti A. Tetapi mereka akan tetap memimpikan untuk bertemu dengan D suatu hari.

Nilai kecocokannya (*fitness*) sangat mudah dihitung. Kita hanya perlu memberikan satu poin untuk setiap gen yang berkorespondensi dengan gen idealnya. Gen yang sempurna akan memperoleh nilai 4. Kemungkinan bereproduksi pada suatu subyek juga sangat tergantung pada nilai ini. Pada kasus tadi, kita akan memperoleh nilai seperti yang ditunjukkan tabel di bawah ini :

Subject	Fitness	Reproduction probability
A	1	$1/7 = 0.143$
B	1	$1/7 = 0.143$
C	2	$2/7 = 0.286$
D	3	$3/7 = 0.428$
Total	7	$7/7=1$




Sekarang kita akan membuat suatu siklus reproduksi dengan 4 keturunan. Subyek D akan dipilih 4 kali dan akan memperoleh 4 keturunan. Subyek C akan dipilih 2 kali sehingga A dan B masing-masing akan dipilih hanya satu kali saja.

Pola reproduksinya seperti yang ditunjukkan pada tabel di bawah ini :

Sbj	Received genes	Genome	Fts	Reproduction probability
A'	A :  D : 		2	$2/10=0.2$
B'	B :  D : 		2	$2/10=0.2$
C'	D :  C : 		3	$3/10=0.3$
D'	C :  D : 		3	$3/10=0.3$
Total			10	$10/10=1$

Selama reproduksi silang yang bermunculan pada tempat-tempat yang acak. Sehingga akan memunculkan suatu populasi baru berisi A', B', C' dan D'. Populasi ini memiliki hubungan ke gen ideal lebih tinggi dari sebelumnya. Dalam contoh kita, total nilai kecocokan dari seluruh populasi melonjak dari 7 ke 10.

Dalam siklus-siklus reproduksi berikutnya, C' dan D' akan memiliki keturunan sebagai berikut :

D' :  + C' :  = 

Subyek baru ini telah memiliki gen ideal, lengannya telah menjadi 'flipper'.

Sekarang bisa kita lihat bahwa prinsip-prinsip dasar dari Genetic Algorithm cukup simpel, yaitu :

1. Mengkodekan permasalahan dalam string biner.
2. Membuat suatu populasi acak. Bagian ini termasuk membuat sebuah kumpulan genetik yang merepresen-tasikan kelompok-kelompok solusi yang mungkin.
3. Menghitung nilai kecocokan (fitness) dari masing-masing subyek. Nilai ini akan langsung bergantung dengan jarak dari solusi optimumnya.
4. Seleksi subyek yang akan berpasangan sesuai dengan bagiannya pada populasi kecocokan global.
5. Perkawinan silang dan mutasi genom.
6. Kemudian diulangi lagi langkah demi langkah dari langkah ke-3 sampai menemukan optimum.

Penggunaan dari Genetic Algorithm juga bisa dimodelkan dengan menggunakan notasi Genotip (Genotype-GTYPE) dan fenotip (Phenotype-PTYPE)<sup>10</sup>.

1. Memilih pasangan GTYPE menurut kecocokan PTYPE mereka.
2. Mengaplikasikan operator genetik seperti perkawinan silang dan mutasi, untuk menciptakan GTYPE baru.
3. Mengembangkan GTYPE baru tersebut dan mendapatkan PTYPE untuk generasi baru, kemudian diulangi lagi dari langkah ke-1.

Persilangan (kawin silang) merupakan basis dari Genetic Algorithm, walaupun selain itu masih ada operator genetik lain yaitu mutasi. Kenyataannya, solusi yang diinginkan mungkin terjadi tidak pada kumpulan genetik yang diberikan, walaupun pada kumpulan genetik yang besar. Mutasi memungkinkan penggabungan sebuah konfigurasi genetik baru, dimana akan memperlebar kemungkinan menemukan solusi optimal. Operasi lain seperti inversi juga memungkinkan terjadi, tetapi kita tidak akan berhubungan dengan operasi itu saat ini.

#### 2.4. Masalah Scaling (Pemberian Skala)

Kita telah melihat sebelumnya bahwa pada genetic algorithm, kemungkinan reproduksi sangat tergantung pada nilai kecocokan setiap subyek. Kita akan mensimulasi hal tersebut dengan memberikan tekanan adaptasi pada lingkungannya.

Penggunaan dari metode ini mau tidak mau akan membentuk dua buah masalah :

1. Sebuah "super-subject" akan menjadi terlalu sering dipilih dalam sebuah populasi dan selalu mencoba untuk mengkonvergenkan genomnya. Berbagai macam hal pada kumpulan genetik harus dikurangi untuk melanjutkan proses Genetic Algorithm.
2. Dalam eksekusi Genetic Algorithm, perbedaan antara nilai kecocokan (fitness) dapat dikurangi. Nilai yg terbaik akan mendapatkan nilai kemungkinan seleksi yang sama dengan yang lain, sehingga Genetic Algorithm akan berhenti berfungsi.

Untuk meringankan masalah ini, sangatlah mungkin untuk mengubah nilai kecocokan (*fitness*). Berikut ini adalah 4 metode utama yang dapat digunakan :

1. Windowing : untuk setiap subjek, kurangi nilai fitness dengan nilai fitness subjek yang lebih buruk. Hal ini memungkinkan untuk memperkuat subjek terkuat dan untuk memperoleh distribusi berbasis nol.
2. Exponential : metode ini dikemukakan oleh S.R Ladd<sup>11</sup>, dengan mengambil nilai akar dari fitness plus one (maksudnya ditambah 1 ya?). Hal ini memungkinkan untuk mengurangi pengaruh subjek terkuat.
3. Linear Transformation: mengaplikasikan sebuah perubahan linear pada masing2 fitness, contoh:  $f' = a.f + b$ . Subjek terkuat sekali lagi dikurangi.
4. Linear normalization: fitness dilinierisasikan. Contoh: dalam sebuah populasi dengan 10 subjek, subjek pertaman akan mendapat 100, subjek kedua 90, kemudia 80...dst, yang terakhir akan mendapat 10. kemudian hindarilah ketegangan dari perhitungan langsung. Bahkan jika perbedaan diantara subjek terbilang cukup kuat, maka perbedaan di antara kemungkinan dalam mereproduksi hanya bergantung pada ranking dari subjek.

Untuk mengilustrasikan metode ini, masi kita anggap ada sebuah populasi yang terdiri dari empat subyek untuk memeriksa akibat dari *scaling* ini. Untuk setiap subyek, kita akan memberikan nilai kecocokan (*fitness*) dan kemungkinan dipilih yang bersesuaian. Hasilnya bisa dilihat pada tabel di bawah ini :

Subjects	1	2	3	4
Rough Fitness	50/50%	25/25%	15/15%	10/10%
Windowing	40/66.7%	15/25%	5/8.3%	0/0%
Exponential	7.14/36.5%	5.1/26.1%	4.0/20.5%	3.32/16.9%
Linear transfo.	53.3/44.4%	33.3/27.8%	20/16.7	13.3/11.1%
Linear normalization	40/40%	30/30%	20/20%	10/10%

Windowing telah mengeliminasi subyek-subyek yang paling lemah, dimana kemungkinannya menjadi 0 dan kemudian akan meningkatkan kemungkinan subyek terkuat. (subyek terkuat kemungkinannya bertambah dari 50% menjadi 67%).

Proses exponential akan meratakan distribusinya. Ini sangat berguna ketika sebuah super-subject menginduksi secara cepat dan konvergen.

Proses linear transformation secara umum memiliki peran yang mirip dengan proses exponential.

Pada akhirnya, proses linear normalization akan menetralkan nilai distribusi sesuai dengan nilai kecocokannya dan hanya tergantung pada rankingnya. Proses ini akan menghindari suatu super-subject yang baik menjadi distribusi yang terlalu homogen.

## 2.5. Pengertian Umum

Genetic algorithm adalah sistem orisinal yang diharapkan bekerja seperti kehidupan<sup>12</sup>. Metodenya sangat berbeda dengan kebanyakan algoritma optimasi<sup>13</sup>. Ciri-cirinya sebagai berikut :

1. Menggunakan hasil pengkodean dari parameter, bukan parameter itu sendiri.
2. Bekerja pada populasi bukan pada sesuatu yang unik.
3. Menggunakan nilai satu-satunya pada fungsi dalam prosesnya. Tidak menggunakan fungsi luar atau pengetahuan luar lainnya.
4. Menggunakan fungsi transisi probabilitas, bukan yang pasti.

Sangatlah penting untuk memahami bahwa memfungsikan algoritma seperti itu tidak menjamin kesuksesan. Kita yang ada di dalam system stochastic dan kumpulan genetik

mungkin terlalu jauh dari solusi, contohnya, pengkonvergenan yang terlalu cepat bisa menghambat proses evolusi. Algoritma ini adalah algoritma yang benar-benar mangkus, dan digunakan dalam takaran yang berbeda-beda dari pertukaran stock, penjadwalan produksi atau pemograman robot perakitan dalam industri otomotif.

## 3. PROSEDUR GENETIC ALGORITHM

Suatu Genetic Algorithm standar membutuhkan dua hal untuk didefinisikan, yaitu :

1. Sebuah [genetic representation](#) dari sebuah [solution domain](#) (domain solusi),
2. Sebuah [fitness function](#) untuk mengevaluasi sebuah domain solusi.

Representasi standar dari solusinya adalah sebuah [array of bits](#) (Larik bit). Larik dari tipe lain atau struktur lain juga bisa digunakan. Properti utama yang membuat representasi genetic ini baik adalah bagian-bagiannya yang bisa diakses dengan mudah karena ukuran yang pasti (*fixed*), yang memudahkan suatu operasi persilangan yang sederhana. Representasi panjang variabel juga digunakan disini, tetapi implementasi persilangan jauh lebih sulit pada kasus ini. Representasi dalam bentuk tree dibahas lebih lanjut pada [Genetic programming](#) dan representasi bebas dibahas lebih lanjut pada [HBGA](#).

Fungsi penghitung nilai kecocokan (fitness) didefinisikan pada representasi genetic dan digunakan untuk mengukur kualitas (quality) pada solusi yang direpresentasikan. Fungsi penghitung ini selalu tergantung pada masalah yang ada (problem dependent). Sebagai contoh, pada [knapsack problem](#), kita ingin memaksimalkan nilai total objek yang bisa dimasukkan ke knapsack (karung) yang memiliki



kapasitas tertentu. Representasi solusinya mungkin bisa sebuah larik bit, dimana setiap bitnya merepresentasikan obyek yang berbeda, dan nilai dari bitnya (0 atau 1) merepresentasikan apakah obyek itu ada di knapsack atau tidak. Tidak setiap representasi solusi valid, karena bisa saja jumlah total obyek-obyeknya melebihi kapasitas dari knapsack itu sendiri. Nilai kecocokan (fitness) solusi adalah jumlah total nilai-nilai dari obyek-obyek di dalam knapsack jika representasinya valid atau nilainya adalah 0 jika representasinya tidak valid. Pada suatu kasus, sangat susah atau bahkan tidak mungkin untuk menemukan representasi dari *fitness*-nya; pada kasus ini, [interactive genetic algorithms](#) digunakan.

Setelah kita memiliki representasi genetik dan sebuah fungsi untuk mencari nilai kecocokan (fitness) terdefinisi, maka Genetic Algorithm akan melanjutkan untuk membentuk suatu populasi acak, kemudian meningkatkannya melalui aplikasi yang berulang-ulang dari mutasi, persilangan, dan operator seleksi.

### Initialization

Pada awalnya solusi individual akan secara acak dibuat dalam bentuk sebuah inisial populasi. Besar populasinya sangat tergantung pada keadaan masalah itu sendiri, tapi biasanya populasi mengandung sekitar beberapa ratus atau bahkan ribuan solusi yang mungkin. Secara sederhana, populasinya dibuat secara acak, dengan mengcover seluruh kemungkinan solusi (*search space*). Cara lainnya, solusinya mungkin bisa di "seeded" pada area dimana kemungkinan besar ditemukan solusi optimalnya.

### Selection

Seiring dengan berjalannya algoritma, suatu bagian pada populasi akan dipilih ([selected](#)) untuk membuat suatu generasi baru. Solusi individual tersebut dipilih melalui suatu *fitness-based* process, dimana solusi pencocok ([fitter](#), yang diukur oleh suatu [fitness function](#)) akan menyatakan kemungkinan terpilih. Beberapa metode seleksi menggunakan nilai kecocokan tersebut dan kemudian memilih solusi terbaik dari situ. Metode lain hanya menggunakan solusi acak dari populasi, sehingga proses ini mungkin akan memakan waktu sangat lama.

Sebagian besar fungsi bersifat [stochastic](#) dan didesain agar sebagian kecil dari solusi yang baik terpilih. Hal ini menolong untuk menjaga keanekaragaman pada suatu populasi cukup besar, mengurangi terjadinya prematur

konvergen solusi pada populasi. Metode seleksi yang populer dan telah teruji antara lain [roulette wheel selection](#) dan [tournament selection](#).

### Reproduction

Langkah selanjutnya adalah dengan membuat generasi kedua dari populasi yang ada melalui [genetic-operator](#): [crossover](#) (persilangan), dan atau [mutation](#) (mutasi).

Untuk setiap solusi baru yang dibentuk, sebuah pasangan "parent" atau orang tua solusi dipilih dari kumpulan populasi sebelumnya. Dengan membuat sebuah "child" atau anak solusi menggunakan metoda diatas, yaitu persilangan dan mutasi, sebuah solusi baru telah dibuat, dimana pada umumnya akan mewarisi bagian-bagian dari orang tuanya. Orang tua baru dipilih lagi dan membuat suatu anak solusi lagi, dan berlanjut sampai suatu populasi solusi baru dengan ukuran yang cukup terbentuk.

Proses ini akan menghasilkan suatu generasi baru dimana kromosomnya berbeda dengan generasi sebelumnya. Secara umum rata-rata nilai kecocokannya (fitness) akan meningkat melalui prosedur ini, karena hanya organisme-organisme terbaik yang dipilih dalam pembentukan populasi selanjutnya, bersama dengan beberapa yang agak cocok dengan solusinya, alasannya sudah disebutkan di atas.

### Termination

Proses tersebut diatas akan terus dilakukan sampai suatu kondisi terminasi/berhenti ditemukan. Kondisi terminasi/berhenti yang umum dipergunakan yaitu :

- Suatu solusi ditemukan yang memenuhi kriteria minimum
- Generasi telah mencapai suatu tingkat tertentu
- Budget yang dialokasikan (misalnya waktu komputasi) telah dicapai
- Solusi dengan nilai kecocokan tertinggi akan mencapai atau telah mencapai suatu batas dimana proses selanjutnya yang akan dilakukan tidak akan menghasilkan hasil yang lebih baik
- Inspeksi secara manual dan berkala
- Kombinasi dari berbagai macam cara terminasi di atas

### Pseudo-code algorithm

1. Memilih atau membuat suatu [populasi](#) inisial.

2. Menghitung nilai kecocokan (fitness) untuk setiap individu pada populasi tersebut.
3. Proses pengulangan
  1. Memilih beberapa individu yang memiliki nilai kecocokan tertinggi untuk melakukan proses reproduksi.
  2. Membuat suatu generasi baru melalui proses persilangan dan mutasi (operasi genetika) sehingga akan memberikan kelahiran pada beberapa bibit unggul.
  3. Menghitung nilai kecocokan individual pada bibit unggul tersebut.
  4. Mengganti individu dengan ranking terburuk pada populasi sebelumnya dengan bibit unggul hasil operasi genetika tadi.
4. Sampai mencapai suatu kondisi terminasi yang sesuai.

### Observations

Ada beberapa observasi umum tentang pembatasan solusi melalui Generic Algorithm, antara lain:

- Pada banyak kasus dengan kompleksitas yang cukup rumit, Genetic Algorithm memiliki kecondongan untuk menuju ke optimum lokal daripada mendapatkan suatu optimum global dari problem tersebut. Kasus seperti ini muncul sangat tergantung pada bentuk dari fungsi pengambilan nilai kecocokannya. Beberapa masalah mungkin menyediakan akses yang mudah ke solusi optimum global, sebaliknya yang lain akan memudahkan menemukan solusi optimum lokal. Masalah ini bisa dihindari dengan menggunakan fungsi yang berbeda, memperbanyak jumlah mutasi yang terjadi, atau menggunakan metode seleksi yang menjaga keanekaragaman populasi. Teknik yang biasa digunakan untuk menjaga keanekaragaman adalah "niche penalty", dimana, setiap grup individe memiliki suatu nilai pinalti (niche radius), yang mana akan mengurangi kemunculan dari grup tersebut pada generasi selanjutnya, sehingga memungkinkan individu lain untuk tetap berada pada populasi.
- Melakukan operasi pada set data yang dinamik cukup sulit, karena genom mulai untuk berkonvergen lebih awal menuju solusi dimana pada akhirnya bisa menjadi solusi yang tidak valid. Beberapa metode telah dikeluarkan untuk menghindari proses meningkatnya keanekaragaman genetik dan mengurangi terjadinya proses konvergen yang terlalu dini, antara lain dengan meningkatkan kemungkinan terjadinya mutasi jika kualitas solusinya menurun (*triggered hypermutation*), atau dengan memperkenalkan suatu yang benar-benar baru, elemen baru yang dibuat secara acak ke dalam kumpulan gen (*random immigrants*). Hasil penelitian juga menyatakan bahwa ada keuntungan jika menggunakan suatu biological *exaptation* dalam menyelesaikan masalah-masalah tersebut.
- Genetic Algorithm tidak bisa menyelesaikan secara efektif suatu masalah dimana nilai kecocokannya hanya berupa benar/salah (0 dan 1), sehingga tidak bisa mengambil solusinya (Seperti tidak ada bukit untuk didaki). Dalam kasus seperti ini, pencarian random akan menghasilkan suatu solusi secepat jika menggunakan Genetic Algorithm.
- Selection jelas merupakan operator genetik yang sangat penting, tetapi pendapat kebanyakan orang terbagi atas kepentingan persilangan lawan mutasi. Beberapa orang berpendapat bahwa persilangan merupakan yang paling penting, sementara mutasi hanya digunakan untuk meyakinkan bahwa solusi yang berpotensi tidak hilang. Sementara yang lain berpendapat bahwa persilangan pada suatu populasi yang besar dan seragam hanya memberikan suatu inovasi yang aslinya ditemukan oleh mutasi, dan pada suatu populasi yang tidak terlalu seragam, persilangan selalu sangat mirip pada mutasi skala besar.
- Seringkali Genetic Algorithm dapat menemukan suatu solusi yang baik walaupun pada ruang pencarian yang sangat sulit.
- Untuk masalah optimasi yang spesifik, beberapa algoritma optimasi mungkin akan menemukan solusi yang lebih baik



daripada Genetic Algorithm (sama-sama diberikan waktu perhitungan yang sama). Seperti algoritma alternatif berikut ini : [simulated annealing](#), [hill climbing](#), dan [swarm intelligence](#) ([ant colony optimization](#), [particle swarm optimization](#)).

- Dengan semua mesin mempelajari masalah, akan sangat baik untuk meningkatkan parameter seperti kemungkinan mutasi, kemungkinan terjadinya rekombinasi dan ukuran populasi untuk menemukan suatu pengaturan yang baik pada masalah yang sedang dikerjakan. Angka mutasi yang terlalu kecil akan mengakibatkan suatu *genetic drift* (yang mana sangat jarang terjadi pada alam) atau konvergensi prematur pada optimum lokal sehingga Genetic Algorithm tidak bekerja dengan baik. Angka mutasi yang terlalu besar bisa menyebabkan kemungkinan menghilangnya solusi-solusi yang baik. Saat ini sudah ada basis teori untuk batas atas dan batas bawah untuk parameter-parameter ini yang mungkin bisa membantu pada proses seleksi, namun masih belum terlalu sering dipraktekkan.
- Implementasi dan evaluasi fungsi nilai kecocokan (fitness) merupakan suatu faktor yang sangat penting demi kecepatan dan ke'mangkus'an (efisiensi) algoritma.

#### 4. IMPLEMENTASI PADA TSP

Seusai dengan judul makalah ini, sekarang akan dibahas mengenai implementasi Genetic Algorithm ini untuk menyelesaikan TSP.

##### Implementasi Standar

Berikut adalah Implementasi Standar dalam implementasi Genetic Algorithm untuk menyelesaikan TSP. Implementasi ini bekerja dengan cukup baik, namun ada kemungkinan cukup besar akan ditemukannya local optimum dibanding global optimum.

##### Genom

Dalam membuat genom untuk TSP, kita tidak bisa menggunakan model representasi standar. Karena setiap kota haruslah unik dalam gen dan tidak bisa diduplikasi.

Oleh karena itu digunakan model representasi sekuensial pada genom dimana setiap kota di list pada urutan yang ke berapa kota itu dikunjungi. Ini adalah cara yang paling biasa dalam representasi Genom TSP. Contohnya :

[9 3 4 0 1 2 5 7 6 8]
-----------------------

##### Persilangan

Untuk operasi persilangannya, pada umumnya digunakan Greedy Crossover yang dibuat oleh J. Grefenstette. Kutipan dari Sushil J. Louis tentang Greedy Crossover ini adalah :

" Greedy crossover selects the first city of one parent, compares the cities leaving that city in both parents, and chooses the closer one to extend the tour. If one city has already appeared in the tour, we choose the other city. If both cities have already appeared, we randomly select a non-selected city"

Atau dalam bahasa indonesia dapat dikatakan bahwa Greedy crossover akan memilih kota pertama dari salah satu induknya, kemudian membandingkan kota berikutnya pada kedua orang tuanya, lalu melanjutkan perjalanan. Jika suatu kota telah dikunjungi, akan dilanjutkan ke kota lainya. Jika kedua kota orang tua selanjutnya telah ditemui, akan dipilih suatu kota yang belum dikunjungi secara acak.

##### Mutasi

Dalam operasi mutasi tradisional, kita bisa langsung mengganti bit-bit pada gen. Namun untuk TSP kita tidak dapat melakukannya. Karenanya, dilakukan suatu pertukaran pada urutan kota yang tercantum di genom. Contohnya :

Sebelum mutasi : [0 1 2 3 4 5 6]
Setelah mutasi : [0 1 3 2 4 5 6]

Banyak cara untuk melakukan operasi pertukaran tersebut. Cara paling sederhana dan mudah adalah dengan menggunakan pertukaran acap (*random swap tech*). Namun cara seperti ini tidak akan dapat menemukan solusi optimum dengan cepat namun dapat mengurangi kemungkinan terjadinya konvergensi pada optimum lokal. Cara yang umum digunakan adalah mutasi *greedy-swap*. Satu lagi kutipan dari Sushil J. Louis :

"The basic idea of greedy-swap is to randomly select two cities from one chromosome and swap them if the new (swapped) tour length is shorter than the old one"

Atau dapat dikatakan ide dasarnya sedikit mirip dengan random swap tech, hanya saja sebelum dilakukan pertukaran dilakukan pengecekan apakah rute terbaru lebih singkat daripada yang lama atau tidak. Jika iya, akan dilakukan pertukaran tersebut, jika tidak akan dilakukan lagi pengambilan acak untuk dilakukan pertukaran.

### Seleksi (*Selection*)

Untuk proses seleksi ada beberapa metode yang bisa dipakai (dipakai salah satu atau dikombinasikan), antara lain :

#### 1. Roulette Wheel Selection

Definisi dari situs Marek Obitko :

*Cost Selection* : Orang tua dipilih berdasarkan nilai kecocokannya (fitness). Kromosom yang lebih baik memiliki persentase dipilih yang lebih besar. Bisa dibayangkan sebuah roda roulette dimana diletakkan semua kromosom pada suatu populasi, setiap kromosom memiliki ukuran tempat bola yang berbeda-beda sesuai dengan nilai kecocokannya.

*Rank Selection* : Metode seleksi sebelumnya memiliki masalah ketika nilai kecocokannya berbeda sangat jauh. Misalkan saja kromosom terbaik memiliki nilai kecocokan (fitness) sebesar 90 %. Maka kemungkinan kromosom lain dipilih akan menjadi sangat kecil. Dengan metoda *Rank Selection*, pertama-tama akan dilakukan perankingan untuk populasi. Dan setiap kromosom akan mendapat nilai kecocokan berdasarkan rankingnya pada populasi. Jadi kromosom terburuk akan mendapat nilai 1, kedua terburuk mendapat nilai 2, demikian seterusnya, sehingga yang terbaik akan mendapat nilai N, dimana N adalah jumlah total kromosom pada populasi.

#### 2. Tournament Selection dan Elitism

Definisi dari W. B. Langdon, University College, London :

“A mechanism for choosing individuals from a population. A group (typically between 2 and 7 individuals) are selected at random from the population and the best (normally only one, but possibly more) is chosen. An elitist genetic algorithm is one that always retains in the population the best individual found so far. Tournament Selection is naturally elitist”

Terjemahannya, seleksi ini merupakan suatu mekanisme pemilihan individu dalam suatu populasi, dimana sebuah grup (biasanya terdiri dari 2 sampai 7 individu) dipilih secara acak dari populasi dan yang terbaik (biasanya cuma satu, namun mungkin lebih) dipilih dari salah satu golongan elit yang merupakan individu terbaik yang ditemukan sejauh ini.

### *Co-Evolutions. Migrations*

Genetic Algorithm merupakan algoritma yang cukup rapi. Namun dia datang dengan kumpulan persoalannya sendiri. Jika mendapat suatu masalah skala besar, ada kemungkinan besar Genetic Algorithm akan tertahan pada optimum local. Atau dengan kata lain, algoritma ini akan menemukan solusi yang cukup baik, namun bukan merupakan solusi yang terbaik. Ada beberapa cara untuk mengatasi masalah ini, dan salah satunya adalah dengan co-evolution.

Dapat dikatakan jikalau co-evolution ini memanfaatkan fitur SMP pada WinNT/2k, yang dioperasikan pada banyak CPU sekaligus. Kita bisa dengan mudah menjalankan beberapa Genetic Algorithm secara terpisah tanpa ada penalti. Untuk pertukaran data antar CPU kita bisa memigrasikan (migrate) gen-gen terbaik pada masing-masing populasi.

### Basis Implementasi

Genetic Algorithm class mengimplementasikan basis logika dari Genetic Algorithm, yaitu : rekombinasi dan mutasi. User bisa mengatur populasi gen dan isi gen, metode seleksi dan metode randomisasi. User juga bisa menspesifikasi tingkah laku class ini dengan menggunakan template. Berikut contoh templatennya dalam bahasa C :

```
template <typename Traits,
typename Selection> class GA {...}
```

Parameter template Traits harus mendefinisikan typedef untuk class gen, class random, container populasi, dan sinkronisasi class.

Parameter template Selection harus menyediakan metode seleksi Genetic Algorithm. Sekarang ada 3 jenis class, yaitu : `selection_tournament<>`, `selection_roulette_cost<>`, dan `selection_roulette_rank<>`.

### Interface GA<>

- init – menginisialisasi populasi
- update – menghitung nilai kecocokan (fitness) dan mengembalikan gen optimal atau end()
- find\_best – mencari gen dengan nilai kecocokan terbaik
- epoch – membuat populasi berikutnya (via selection, persilangan, mutasi dsb)
- recombine – membuat seleksi, menghasilkan gen-gen baru, menghapus orang tua yang tidak elit, dan menghilangkan gen-gen yang kembar
- mutate – melaksanakan proses mutasi gen
- migration – menukarkan gen-gen terbaik antar populasi
- sort – mengurutkan gen-gen pada populasi berdasarkan nilai kecocokannya (fitness), meletakkan yang terbaik pada urutan paling pertama populasi.
- begin – mengembalikan iterator pada gen terbaik pertama di populasi
- end – mengembalikan iterator pada posisi setelah akhir dari populasi
- size – mengembalikan nilai ukuran dari populasi

Contoh penggunaan dari class GA<>, adalah sebagai berikut (masih dalam bahasa C) :

```
typedef ga_traits<RandomCRT> traits;
typedef GA<traits,
selection_tournament<traits> > tGA;
traits::Gene::Context context;
tGA ga(50, &context);
tGA::iterator it = ga.update();
while(it == ga.end())
{
    ga.epoch();
    it = ga.update();
}
traits::Gene* gnp = (*it);
```

## 5. KESIMPULAN

Genetic Algorithm yang merupakan algoritma heuristic yang dikembangkan dari teori genetika kehidupan, merupakan salah satu algoritma yang sangat bermanfaat. Walaupun solusi yang didapatkannya tidak pasti merupakan solusi yang optimal, namun algoritma ini masih menghasilkan solusi yang cukup baik. Dalam menyelesaikan TSP, Genetic Algorithm dapat menyelesaikan TSP dengan cukup baik untuk

kasus sekitar 200 kota. Walaupun solusinya tidak selalu merupakan optimal global.

## 6. DAFTAR PUSTAKA

- [http://en.wikipedia.org/wiki/Genetic\\_algorithm](http://en.wikipedia.org/wiki/Genetic_algorithm) (2 Januari 2007)
- <http://www.google.com> (1 Januari 2007)
- <http://www.generation5.org/content/2001/tspapp.asp> (3 Januari 2007)
- <http://www-cse.uta.edu/~cook/ai1/lectures/applets/gatsp/TSP.html> (3 Januari 2007)
- <http://www.gcd.org/sengoku/docs/arob98.pdf> (3 Januari 2007)
- [http://www.rennard.org/alife/english/ga\\_vintrgb.html](http://www.rennard.org/alife/english/ga_vintrgb.html) (2 Januari 2007)
- Crosby, Jack L. (1973), *Computer Simulation in Genetics*, John Wiley & Sons, London.
- Fraser, Alex S. (1957), *Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction*. Australian Journal of Biological Sciences vol. 10 484-491.
- Goldberg, David E (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Kluwer Academic Publishers, Boston, MA.