Neural Network Evolution dengan *Genetic Algorithm* dalam Menyelesaikan Video Game

Shaffira Alya Mevia - 13519083[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13519083@std.stei.itb.ac.id

***Abstract*—Video game adalah sebuah media hiburan dan juga bisa masuk ke dalam kategori professional. Banyak orang yang menghabiskan waktunya untuk menguasai sebuah video game. Akan tetapi, tidak hanya orang namun juga sebuah kecerdasan buatan. Dewasa ini banyak video pada media sosial yang menunjukkan bagaimana suatu kecerdasan buatan dapat mengajarkan dirinya sendiri atau berevolusi untuk menyelesaikan suatu video game sederhana seperti Super Mario World dan Flappy Bird. Dibalik kata 'kecerdasan buatan' tersebut sebenarnya adalah aplikasi dari teori graf pada *Neural Network* yang digabungkan dengan *Genetic Algorithm* untuk membuat suatu *NeuroEvolution*.**

***Keywords*—Video Game, Graf, Neural Network, Genetic Algorithm, NeuroEvolution of Augmenting Topologies (NEAT).**

I. Pendahuluan

Video game awalnya adalah sebuah media untuk hiburan semata, sekarang bisa menjadi salah satu sumber pendapatan stabil untuk beberapa kalangan. Mulai banyak orang menghabiskan waktunya untuk menyelesaikan game yang mereka sukai. Dampak daripada video game tidak mesti selalu negatif. Dalam industri pembuatan video game, banyak individu yang mengekspresikan kreativitasnya dalam media game.

Dewasa ini, sering muncul video pada media sosial bagaimana intelegensi buatan dapat menyelesaikan sebuah video game. Game-game yang susah untuk diselesaikan bagi manusia biasa secara normal dapat diselesaikan dengan mudah bagi intelegensi buatan yang dimaksud. Apabila dilihat lebih lanjut, intelegensi buatan ini sebenarnya hanyalah mengaplikasikan dari sebuah algoritma yang dapat berevolusi, yaitu *genetic algorithm*.

Genetic algorithm pada dasarnya terinspirasi dari teori evolusi oleh Charles Darwin, algoritma ini seperti mensimulasikan seleksi natural dimana individu yang paling cocok adalah yang akan dipilih untuk reproduksi generasi selanjutnya. Selain itu, ada juga yang disebut dengan *artificial neural network* atau lebih sering disebut dengan *neural network* saja, adalah sebuah bentuk aplikasi dari teori graf yang sederhana.

Apabila algoritma yang dapat berevolusi ini digunakan dalam *neural network* maka bisa dibuat sebuah *NeuroEvolution* atau *neural network* yang dapat berevolusi. Salah satu *neuroevolution* yang sering kali dipakai untuk menyelesaikan suatu video game adalah aplikasi dari NEAT atau *NeuroEvolution of Augmenting Topologies*. Pada makalah ini akan dijelaskan bagaimana aplikasi NEAT untuk menyelesaikan sebuah game yang sederhana seperti, Super Mario World dan Flappy Bird.

II. Landasan Teori

A. Graf

Graf adalah sebuah struktur yang terdiri dari simpul (*vertex/nodes*) dan sisi (*edges*) yang dapat merepresentasikan hubungan antara objek-objek secara diskrit. Salah satu aplikasi umumnya adalah dalam peta atau *travelling salesman problem*.

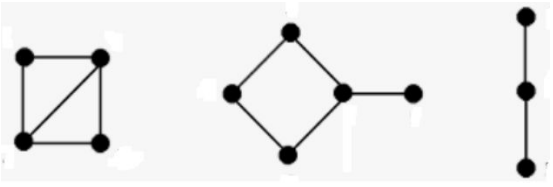
Graf dapat didefinisikan sebagai $G=(V, E)$ dengan V adalah sebuah himpunan tidak kosong dari simpul-simpul, contohnya adalah $\{v_1, v_2, ..., v_n\}$. Sedangkan E adalah sebuah himpunan sisi yang menghubungkan sepasang simpul, contohnya adalah $\{e_1, e_2, ..., e_n\}$.

Pada sisi dan simpul graf dapat diberikan label-label. Label pada sisi disebut sebagai bobot graf. Umumnya, label pada sisi dan simpul tidak mesti harus berupa angka, bisa juga berupa tulisan tergantung kepada penggunaannya untuk apa. Label juga tidak mesti unik. Salah satu aplikasi lain dari graf adalah diagram molekul kimia.

Graf dapat dikategorikan menjadi beberapa jenis berdasarkan ada atau tidaknya gelang atau sisi ganda pada suatu graf. Kategori tersebut adalah sebagai berikut.

1. Graf Sederhana (*Simple Graph*)

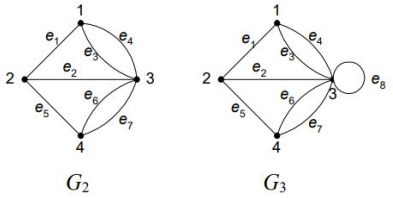
   Graf sederhana adalah graf yang tidak mengandung gelang ataupun sisi ganda. Pada graf sederhana, sisi merupakan pasangan yang tidak terurut sehingga (u, v) memiliki arti yang sama dengan (v, u).



Gambar 1 Graf sederhana
Sumber : [1]

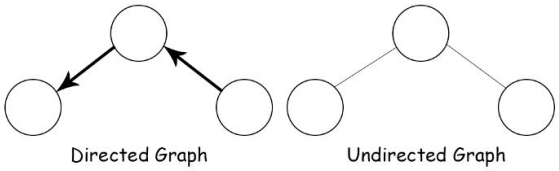
2. Graf Tidak Sederhana (*Unsimple Graph*)

Graf tidak sederhana adalah graf yang mengandung sisi ganda dan/atau gelang papa sisinya. Graf tidak sederhana dapat dikategorikan lagi tergantung kepada adanya sisi ganda dan gelang. Graf Ganda atau *Multigraph* adalah graf yang memiliki sisi ganda. Sedangkan Graf Semu atau *Pseudograph* adalah graf yang mengandung sisi yang menghubungkan simoul dengan simpul itu sendiri.



Gambar 2 Graf Ganda (G_2) dan Graf Semu (G_3)
Sumber : [1]

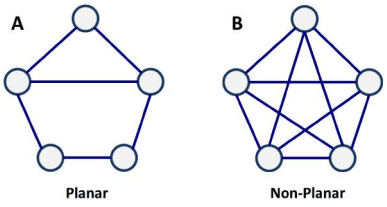
Berdasarkan orientasi arah pada sisi, graf dapat dikategorikan menjadi dua jenis, yaitu graf tak berarah dan graf berarah. Graf Tak Berarah atau *Undirected Graph* adalah graf yang sisinya tidak memiliki orientasi arah. Pada graf tak berarah, urutan pasangan simpul tidak diperhatikan sehingga $(u, v) = (v, u)$. Sementara, Graf Berarah atau *Directed Graph* adalah graf yang setiap sisinya diberikan orientasi arah. Dalam graf berarah urutan pasangan simpul justru diperhatikan, maka $(u, v) \neq (v, u)$.



Gambar 3 Graf Berarah dan Tak Berarah
Sumber : [2]

Graf memiliki beberapa sifat tertentu, beberapa diantaranya adalah berdasarkan jenis simpul dan interaksi dengan struktur data graf. Apabila graf memiliki sifat *heterogeneous* artinya graf terdiri dari simpul-simpul yang berbeda. Kebalikannya adalah *homogeneous*, yang berarti simpul-simpul pada graf adalah sama. Kemudian graf juga bisa bersifat statis, yaitu simpul dan sisi dari graf tidak berubah, dalam kata lain tidak ditambah ataupun dikurangi. Graf juga dapat bersifat dinamis, yaitu simpul dan sisi dari graf dapat berubah, dalam kata lain simpul dan sisi dapat ditambah, dikurangi, ataupun digerakkan.

Bentuk dari graf terkadang sangat abstrak dan susah untuk dimengerti karena banyak sisi yang saling bersilangan. Beberapa graf tertentu dapat diubah menjadi bentuk planar, yaitu sisi-sisi dari graf tidak ada yang bersilangan sama sekali.

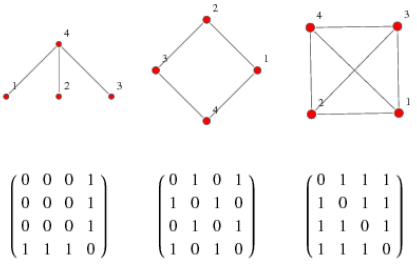


Gambar 4 Graf Planar dan Tidak Planar
Sumber : [2]

Bentuk daripada sebuah graf dapat direpresentasikan menjadi matriks untuk memudahkan proses komputasi. Berikut adalah beberapa cara untuk merepresentasikan graf menjadi sebuah matriks.

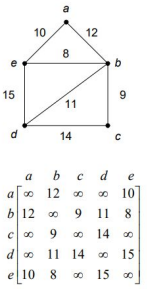
1. Matriks Ketetanggaan (*Adjacency Matrix*)

Matriks ketetanggaan adalah matriks yang terdiri dari simpul sebagai indeks baris dan kolomnya. Jika pada suatu matriks $A=[a_{ij}]$ sehingga a_{ij} akan berisi 1 jika simpul i dan j bertetangga, yaitu ketika dua buah simpul pada graf tidak berarah terhubung secara langsung dengan sebuah sisi, dan 0 jika tidak bertetangga. Pada kasus tertentu, apabila sisi adalah sebuah *loop* maka akan bernilai 2.



Gambar 5 Matriks Ketetanggan
Sumber : [2]

Matriks ketetanggaan dapat berbobot, yang berarti sisi pada graf memiliki suatu nilai tertentu. Sehingga pada matriks, akan berisi dengan bobot tersebut daripada angka 1 atau 2.

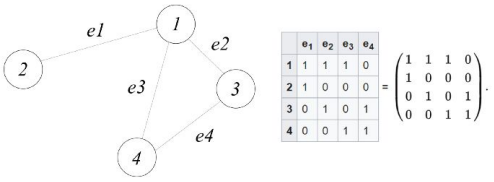


Gambar 6 Matriks Ketetanggaan Berbobot
Sumber : [3]

Pada prakteknya, bobot tersebut dapat merepresentasikan jenis ikatan pada suatu molekul ataupun pada media sosial seperti LinkedIn dapat merepresentasikan urutan hubungan antara dua simpul atau orang. Konsep bobot pada sisi adalah yang membuat graf menjadi salah satu struktur data yang menarik karena bisa menerima informasi yang bersifat struktural dan singular. Dalam kata lain, dapat menerima informasi eksternal dan internal.

2. Matriks Bersisian (*Incidence Matrix*)

Matriks bersisian adalah matriks yang terdiri dari simpul sebagai indeks baris dan kolomnya. Jika pada suatu matriks $A=[a_{ij}]$ sehingga a_{ij} akan berisi 1 jika simpul i dan j bersisian, yaitu sebuah simpul terhubung secara langsung dengan sebuah sisi tertentu, dan 0 jika tidak bersisian.



Gambar 7 Matriks Bersisian
Sumber : [2]

3. Matriks Derajat (*Degree Matrix*)

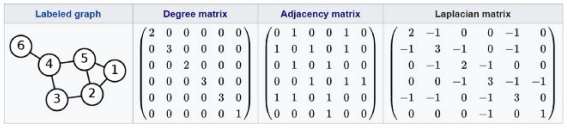
Matriks derajat adalah matriks yang terbentuk dari derajat setiap simpulnya. Derajat adalah jumlah sisi yang bersisian dengan simpul pada graf tak berarah maupun graf berarah.

4. *Laplacian Matrix*

Laplacian Matrix adalah matriks yang merepresentasikan sebuah graf yang didapatkan dari hasil pengurangan matriks ketetanggaan dari matriks derajat.

$$L = D - A \qquad (1)$$

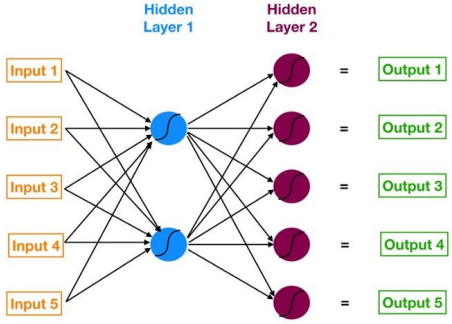
Dengan L adalah *Laplacian Matrix*, D adalah matriks derajat, dan A adalah matriks ketetangaan.



Gambar 8 Tabel hubungan graf, matriks derajat, matriks ketetanggaan, dan matriks *laplacian*.
Sumber : [2]

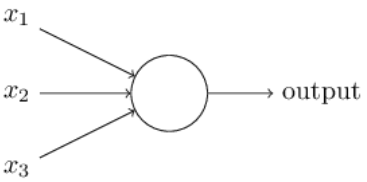
B. Neural Network

Neural Network adalah sebuah struktur data dalam bentuk jaringan atau hubungan multilayer antara neuron atau simpul yang dapat digunakan untuk mengklasifikasikan hal atau membuat prediksi dan bermodelkan kepada jaringan saraf manusia.*Neural Network* (NN) adalah sistem yang adaptif karena dapat mengubah strukturnya tergantung informasi yang diterimanya, bersifat eksternal maupun internal.



Gambar 9 *Neural Network* dengan dua layer tersembunyi
Sumber : [4]

Secara umum, sebuah NN terdiri dari layer input (berwarna oranye), beberapa layer tersembunyi (berwarna biru dan ungu) yang berfungsi sebagai pemrosesan data, dan sebuah layer output atau prediksi (berwarna hijau). Kemudian untuk setiap garis, yang merupakan hubungan antar neuron, memiliki bobotnya masing-masing.



Gambar 10 Sebuah *perceptron* sederhana
Sumber : [5]

Cara kerja NN dapat dijelaskan dengan aplikasi dari salah satu NN yaitu *perceptron*. Sebuah *perceptron* menerima beberapa input biner yang kemudian akan mengeluarkan satu output biner. Setiap hubungan atau *connection* yang dimiliki antara input dengan *perceptron* diberikan bobot, sebuah angka riil, yang merepresentasikan tingkat kepentingan input tersebut terhadap output.

$$output = \begin{cases} 0 \ jika \sum_j w_j x_j \leqslant konstanta \\ 1 \ jika \sum_j w_j x_j > konstanta \end{cases} \qquad (2)$$

Hasil dari output tergantung kepada hasil penjumlahan bobot yang dibandingkan terhadap suatu konstanta tertentu. Konstanta, yang juga merupakan bilangan riil, ini ditentukan sendiri oleh programmer. Secara dasar, demikianlah cara kerja *perceptron* atau NN. Akan tetapi, cara rumusan untuk menentukan output tersebut lebih sering ditemukan dengan memanfaatkan sebuah variabel b yaitu *bias* yang memiliki hubungan $b \equiv -konstanta$. Dengan demikian, perumusan dapat ditulis ulang menjadi *dot product* sebagai berikut.

$$output = \begin{cases} 0 \ jika \ w \cdot x + b \leqslant 0 \\ 1 \ jika \ w \cdot x + b > 0 \end{cases} \qquad (3)$$

Perceptron sendiri sebenarnya tidak jauh berbeda dengan sebuah rangkaian NAND gate. Oleh karena itu, jenis NN yang lazim digunakan sekarang mengimplementasikan *sigmoid neuron* sebagai jenis neuronnya untuk melakukan proses *learning* pada NN. Pada umumnya sebenarnya NN memerlukan suatu fungsi aktivasi untuk bekerja, yang paling umum digunakan adalah fungsi sigmoid namun beberapa kasus juga bisa menggunakan fungsi TanH. Fungsi-fungsi ini dibebaskan terhadap kebutuhan programmer.

$$a_i^n = \sigma\left(w_{i,0}\, a_0^{n-1} + w_{i,1}\, a_1^{n-1} + ... + w_{i,m}\, a_m^{n-1} + b_i\right) \qquad (4)$$

Dalam hal ini, perumusan sudah memasuki NN multilayer dengan *a* adalah representasi neuron, *n* adalah notasi untuk menunjukkan pada layer berapa *a* berada, *m* adalah jumlah neuron yang ada pada layer tersebut, dan *i* adalah penanda untuk setiap neuron yang ada pada layer *n*. Sementara itu, *w* adalah bobot dari *connection*-nya dan b_i adalah *bias*. Kemudian, secara garis besar, keseluruhan dari NN dapat direpresentasikan menjadi matriks seperti berikut dengan *p* adalah jumlah neuron pada layer selanjutnya.

$$\begin{bmatrix} w_{0,0} & \dots & w_{0,m} \\ \dots & \dots & \dots \\ w_{p,0} & \dots & w_{p,m} \end{bmatrix} \times \begin{bmatrix} a_0^n \\ \dots \\ a_m^n \end{bmatrix} + \begin{bmatrix} b_0 \\ \dots \\ b_p \end{bmatrix} = \begin{bmatrix} z_0 \\ \dots \\ z_p \end{bmatrix} \qquad (5)$$

Dari persamaan matriks di atas, hasil matriks *Z* kemudian dimasukkan ke dalam fungsi aktivasi.

C. Genetic Algorithm

Genetic Algorithm adalah sebuah algoritma yang terinspirasi oleh teori evolusi milik Charles Darwin dan mensimulasikan proses tersebut untuk menentukan individu yang paling cocok untuk disilangkan menjadi generasi selanjutnya.

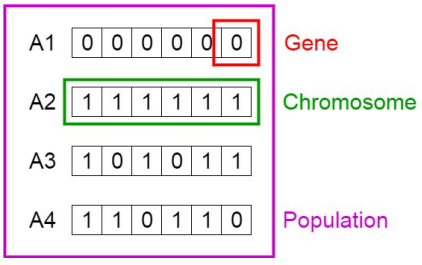
Proses dari seleksi natural dimulai dengan memilih individu yang paling cocok dari sebuah populasi. Kemudian, akan dihasilkan keturunan yang menurunkan karakteristik dari *parent* dan akan ditambahkan pada populasi di generasi selanjutnya. Jika *parent* memiliki *fitness* yang baik, maka keturunannya juga akan memiliki *fitness yang* lebih baik dan kemungkinan yang lebih tinggi untuk melewati proses seleksi natural. Proses ini terus akan berulang sampai ditemukan generasi yang paling tepat.

Pada *genetic algorithm* terdapat lima tahapan sebagai berikut.

1. Populasi Awal (*Initial Population*)

   Proses seleksi natural diawali dengan himpunan individu yang disebut sebagai populasi. Setiap individu ini bisa dibilang sebagai kemungkinan solusi untuk masalah yang ingin diselesaikan. Individu dapat disebut sebagai gen atau *gene* yang merupakan himpunan parameter atau variabel. *Genes* yang disatukan menjadi string untuk membentuk suatu kromosom atau *chromosome* yang merupakan solusi. Ukuran dari populasi adalah selalu sama untuk setiap generasinya.

   Pada algoritma genetika atau *genetic algorithm*, representasi dari setiap gen menggunakan nilai biner, yaitu 1 dan 0.



Gambar 11 Populasi, Kromosom, dan Gen
Sumber : [6]

2. Fungsi *Fitness (Fitness Function)*

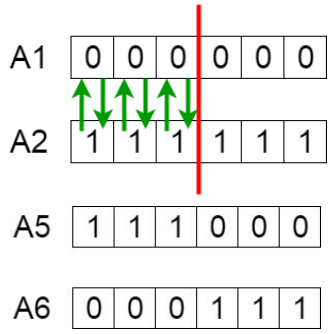
   Fungsi *fitness* menentukan seberapa cocok sebuah individu dalam mengerjakan masalah yang diberikan. Hasilnya adalah sebuah nilai *fitness* yang kemudian akan dibandingkan dengan individu lainnya untuk dipilih sebagai *parent*.

3. Seleksi (*Selection)*

   Seleksi adalah proses untuk memilih individu yang paling cocok dan akan disilangkan agar menghasilkan keturunan untuk generasi selanjutnya. *Parent* atau sepasang individu dipilih berdasarkan nilai *fitness* yang ia miliki. Semakin tinggi nilai tersebut, semakin besar kemungkinan untuk terpilih.

4. Persilangan (*Crossover*)

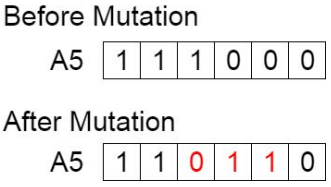
   Persilangan adalah proses yang paling penting dalam algoritma genetika. Untuk setiap kromosom masing-masing *parent* akan ditentukan sebuah titik persilangan atau *crossover point* yang ditentukan secara acak. Keturunan yang dihasilkan akan memiliki gen dari kedua *parent* yang telah ditukar tergantung kepada titik persilangan. Garis merah pada gambar dibawah menandakan titik persilangan.



Gambar 12 Proses persilangan gen antara *parent* dan hasil keturunannya
Sumber : [6]

5. Mutasi (*Mutation*)

   Mutasi adalah proses perubahan urutan gen pada keturunan secara acak. Hal ini dilakukan untuk memastikan keragaman dalam populasi.



Gambar 13 Proses mutasi
Sumber : [6]

Secara umum algoritma genetika akan berhenti apabila nilai *fitness* antargenerasi tidak menimbulkan perbedaan yang signifikan atau programmer bisa melihat sendiri apakah hasil dari generasi sudah cukup untuk menyelesaikan problematika yang diberikan.

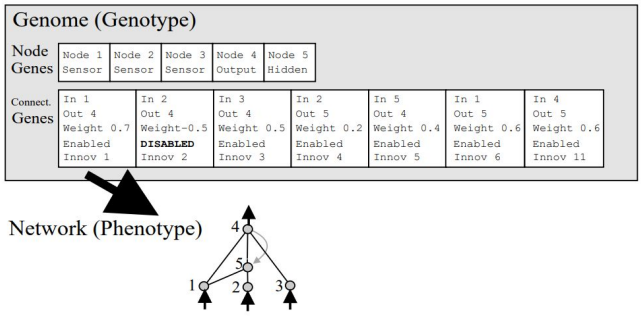
III. *Neural Network Evolution* dengan *Genetic Algorithm*

Walaupun membantu banyak menyelesaikan permasalahan, *Neural Network* masih belum sempurna. Oleh karena itu, *neural network* sering digabungkan dengan *genetic algorithm* untuk menutup ketidaksempurnaan tersebut. Pendekatan ini terkadang disebut sebagai *NeuroEvolution* (NE). NN adalah struktur data yang tepat untuk diimplementasikan evolusi karena memiliki kemampuan untuk merepresentasikan solusi untuk berbagai macam permasalahan. Ada beberapa jenis NE, namun pada makalah ini akan dibahas NEAT atau *NeuroEvolution of Augmenting Topologies*.

Apabila dilihat dari namanya, *Neuro* bermakna kepada *neural network*, *Evolution* bermakna kepada pengaplikasian *genetic algorithm*, *topologies* bermakna kepada banyaknya layer atau node pada *neural network* yang digunakan.

Kemudian *augmenting topologies* bermakna topologi yang bersifat dinamis. Dalam pendekatan NE secara tradisional, sebuah topologi sudah ditentukan sebelum proses evolusi untuk NN dimulai. Biasanya jaringan topologinya adalah satu layer tersembunyi berisi neuron dengan setiap neuron tersembunyi terhubung ke semua jaringan input dan semua jaringan output. Untuk input dan output layer akan statis namun *hidden layer* akan mengalami perubahan-perubahan tertentu.

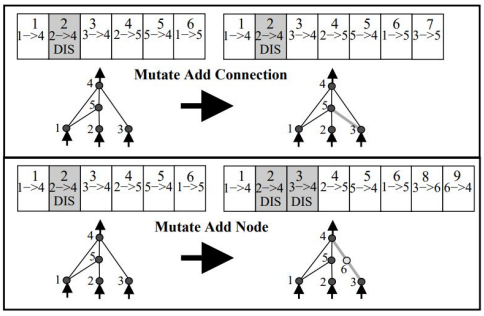
Pada NEAT dikenalkan istilah baru yaitu *genome* atau genotipe yang berisi *node genes* dan *connection genes*. *Node genes* adalah gen-gen untuk simpul sementara *connection genes* adalah gen-gen untuk sisi.



Gambar 14 *Encoding* NEAT pada arsitektur jaringan
Sumber : [7]

Untuk setiap *node gene* atau gen simpul terdiri dari nama gen itu sendiri, status simpul itu sendiri apakah sebuah input, output, atau *hidden*. Kemudian untuk *connection gene* atau gen sisi terdiri dari informasi input dan output node, bobotnya, status *connection* tersebut apakah *enabled* atau *disabled*, dan *innovation number*. *Innovation number* adalah seperti label untuk setiap *connection* secara global untuk seluruh generasi. Hal ini mengakibatkan setiap generasi selalu memiliki data *connection* yang sama dan dapat memanfaatkannya.

Di dalam NEAT juga dikenalkan istilah spesiasi yaitu sebuah proses dalam membuat spesies-spesies sehingga jaringan atau *network* hanya akan disaingkan dengan struktur yang mirip dengan spesies tersebut untuk mengoptimasi bobot dalam membuat topologi yang kemungkinan besar lebih kompleks. Dengan spesiasi ini, *innovation number* akan tetap terjaga karena pada umumnya TWEANN (*Topology and Weight Evolving Artificial Neural Networks)* melakukan evolusi secara paralel atau asinkron, yang tidak memiliki tabel global seperti pada NEAT.

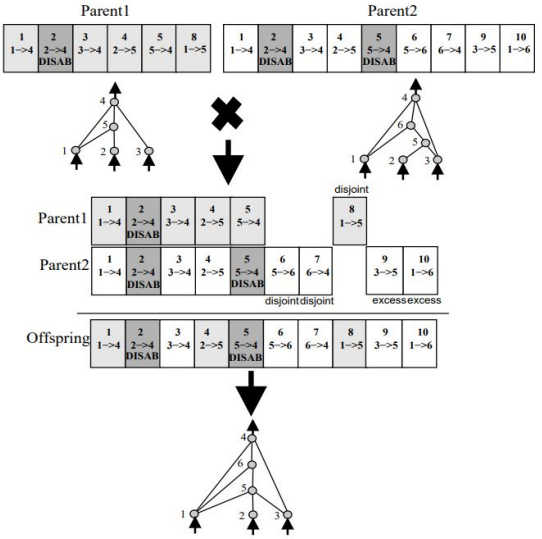


Gambar 15 Proses Mutasi pada NEAT
Sumber : [7]

Secara struktural ada dua jenis mutasi pada NEAT, yaitu menambah *connection* atau menambah *node*. Pada setiap mutasi yang terjadi harus selalu memperhatikan *innovation number* karena *innovation number* seperti penanda bersejarah atau *historical marker* yang mengidentifikasi leluhur gen tersebut. Ketika menambahkan *connection*, sebuah *connection gene* ditambahkan ke dalam *genotype* dengan *innovation number* selanjutnya pada list tersebut. Ketika menambahkan *node*, *connection gene* dimana *node* tersebut ditambahkan akan diset menjadi *disabled* dan ditambahkan dua *connection gene* baru yang menghubungkan ketiga *node* tersebut.

Pada umumnya dalam TWEANN dapat terjadi persilangan yang menyebabkan hilangnya informasi. Misalnya ada kromosom 1 dengan gen-gen [A, B, C] dan kromosom 2 dengan gen-gen [C, B, A]. Dengan algoritma genetika, titik persilangan akan dipilih secara acak, namun pada pasangan kromosom ini dimanapun titik persilangan berada akan selalu menghasilkan jawaban yang sama. Masalahnya jawaban tersebut adalah kromosom 3 dengan gen-gen [A, B, A] dan kromosom 4 dengan gen-gen [C, B, C]. Kromosom 3 dan 4 sama-sama mengalami kehilangan gen, baik gen A atau C.

Masalah yang dijelaskan diatas dapat diselesaikan dengan NEAT yaitu dengan menggunakan *historical marking* untuk melakukan *alignment* gen-gen seperti yang ada pada persilangan biologis.



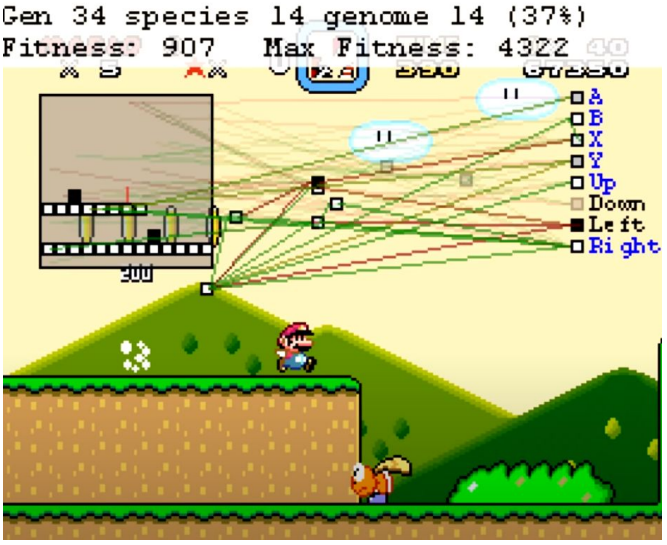
Gambar 16 Persilangan NEAT dengan *Gene Alignment*
Sumber : [7]

Berbeda dengan proses persilangan pada algoritma genetika umumnya, untuk melakukan persilangan hanya bisa terjadi terhadap gen dengan *innovation number* yang sama. Untuk setiap gen yang sesuai akan dipilih secara acak dari kedua *parent*. Apabila gen tersebut hanya ada pada satu *parent*, misalnya gen 6-8, maka akan dimasukkan ke dalam keturunannya. Gen tersebut dinamakan *disjoint genes*. Akan tetapi apabila gen tersebut merupakan *excess genes*, yaitu gen yang tidak memiliki pasangannya dan berada di akhir, gen tersebut hanya akan masuk ke dalam keturunannya apabila *parent* asalnya lebih cocok, dalam kata lain memiliki nilai *fitness* yang lebih tinggi.

Populasi awal pada TWEANN biasanya dibuat secara acak yang memungkinkan untuk membuat kompleksitas algoritma menjadi sangat besar. Untuk menangani hal ini, NEAT melakukan inisiasi awal populasi dengan minimal agar jaringan yang dibuat menjadi lebih sederhana.

IV. Aplikasi NEAT untuk Menyelesaikan Video Game

Dewasa ini, NEAT sering digunakan untuk membuat program yang dapat belajar dan menyelesaikan sebuah game. Berbagai jenis game dapat dikalahkan atau dipelajari oleh NEAT. Dalam contoh kali ini diambil game Super Mario World.

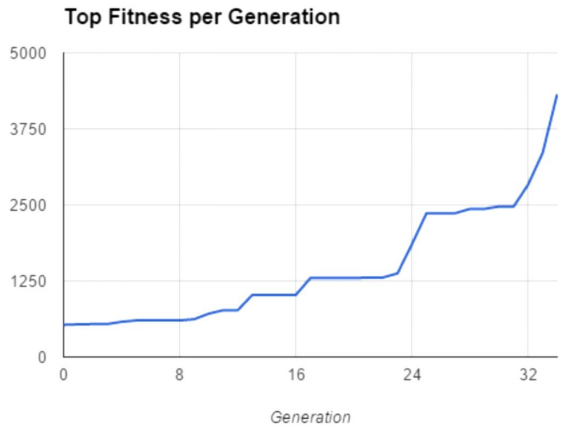


Gambar 17 Simulasi NEAT pada Super Mario World
Sumber : [8]

Pada gambar diatas terlihat hasil simulasi NEAT untuk Super Mario World pada level Donut Plains 1. Pada simulasi tersebut, terlihat banyak garis hijau dan merah yang menghubungkan antara kotak kiri dengan *node* pada bagian kanan.

Kotak pada bagian kiri adalah peta dari dunia di level Donut Plains 1 secara sederhana dengan kotak putih adalah kotak dimana mario bisa berdiri sementara kotak hitam adalah kotak yang bisa bergerak, merepresentasikan musuh-musuh seperti Goomba atau *mushrooms* pada dunia Super Mario World. Dari kotak putih dan hitam ini akan diambil menjadi input *nodes* atau layer untuk NEAT. Sementara itu tulisan pada bagian kanan adalah list kemungkinan output. Apabila menekan *Right* akan maju atau menekan A akan loncat dan seterusnya. Kemudian, pada bagian *neural network* untuk *connection* berwarna hijau akan membaca kotak putih dan kotak hitam dan mengeluarkan warna yang sama, putih tetap putih. Akan tetapi, untuk yang warna merah ajab membaca kotak hitam dan putih dan mengeluarkan warna yang berkebalikan, kotak putih akan menjadi kotak hitam.

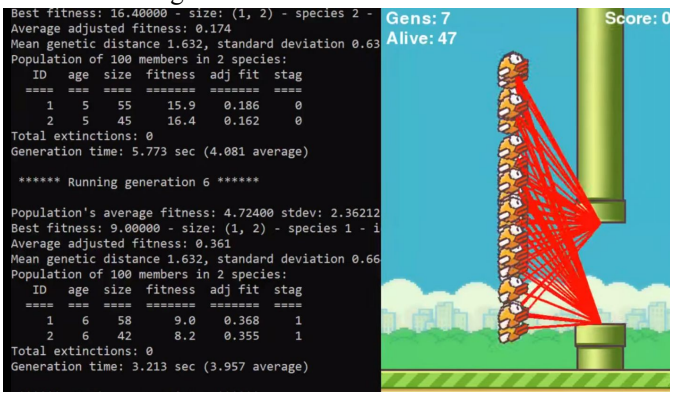
Pada simulasi ini, nilai *fitness* berisi dengan jarak terjauh dan seberapa cepat spesies tersebut dalam menempuh di dalam level tersebut. Berdasarkan dari implementasi NEAT, maka spesies-spesies dengan nilai *fitness* tertinggilah yang akan dipilih untuk melakukan persilangan yang menghasilkan keturunan untuk generasi berikutnya.



Gambar 18 Grafik *Fitness* Tertinggi per Generasi
Sumber : [8]

Diperlukan sekitar 34 generasi untuk sampai akhirnya Mario dapat menyelesaikan level Donut Plains 1 tanpa mati sekalipun dengan nilai *fitness* sebesar 4322.

Dalam video game, NEAT dapat diaplikasikan dengan berbagai cara. Dengan setiap video game tersebut diperlakukan hal yang unik untuk game tersebut. Berbeda dengan video game Super Mario World sebelumnya, di dalam Flappy Bird cara untuk memenangkan gamenya lebih sederhana. Tujuan dari pemain adalah hanya untuk menghindari pipa hijau dengan menekan satu tombol saja, yaitu loncat. Burung Flappy Bird akan bergerak ke arah kanan secara otomatis, namun tidak bergerak ke atas atau ke bawah.



Gambar 19 Simulasi NEAT pada Flappy Bird
Sumber : [9]

Dalam implementasi pada game Flappy Bird ini, input layernya adalah jarak antara burung Flappy Bird dengan pipa atas dan bawah serta posisi burung itu sendiri. Sehingga dalam kasus ini, kita memiliki tiga input node. Kemudian, output layernya adalah command untuk loncat atau tidak. Umumnya fungsi aktivasi menggunakan sigmoid, namun pada kasus ini digunakan fungsi TanH.

Perbedaan simulasi diantara game Flappy Bird ini dengan simulasi Super Mario World adalah saat di Super Mario World hanya ada satu *sprite* atau karakter yang berjalan per simulasi karena hanya melakukan tes per spesies dan tidak langsung per generasi. Namun ternyata simulasi langsung per generasi seperti pada simulasi dengan game Flappy Bird.

NEAT dapat diaplikasikan kepada banyak video game sederhana untuk mensimulasikan bagaimana sebuah kecerdasan buatan dapat berevolusi. Cara penggunaannya juga dapat menggunakan berbagai bahasa namun ada *library* khusus NEAT untuk bahasa pemrograman Python.

V. Kesimpulan

Sebuah game sederhana, seperti Super Mario World dan Flappy Bird ternyata dapat dikalahkan dengan sebuah algoritma canggih yang berdasarkan pada konsep matematika diskrit yang sederhana, yaitu graf. Dari graf yang sederhana bisa membuat suatu simulasi sederhana otak manusia yang sangat kompleks dengan aplikasi *Neural Network* digabung dengan *Genetic Algorithm* yang sering kali juga disebut dengan *NeuroEvolution*. Dalam makalah ini digunakan aplikasi dari NEAT atau *NeuroEvolution of Augmenting Topologies*. Secara sederhana cara kerjanya adalah dengan menginisiasi populasi lalu melakukan simulasi yang menghasilkan nilai *fitness* untuk setiap spesiesnya. Setelah itu, terjadi proses persilangan yang berbeda dengan *genetic algorithm* pada umumnya, yaitu dengan memperhatikan *historical marking* pada *connection genes*. Setelah itu terjadi mutasi yang bisa dibagi menjadi dua yaitu menambahkan simpul atau sisi yang juga memperhatikan *innovation number* pada *connection genes*. Terakhir dilakukan spesiasi untuk membuat populasi baru dan proses diiterasi sampai akhirnya ditemukan generasi yang paling cocok untuk menyelesaikan suatu permasalahan.

Dalam aplikasinya pada game Super Mario World, peta dunia pada game-lah yang menjadi input layer sementara pada game Flappy Bird jarak antara burung dengan pipa serta posisi burung adalah input layer. Kemudian, keduanya secara umum memiliki output layer yang bersifat sama yaitu pergerakan selanjutnya yang akan dilakukan oleh karakter, yaitu maju, loncat, mundur, dan lain sebagainya. Pada simulasi Super Mario World diperlukan 34 generasi untuk menyelesaikan perta Donut Plains 1 dan pada simulasi Flappy Bird hanya diperlukan 13 generasi untuk mencetak skor lebih dari 1000, yang sangat susah dilakukan oleh manusia. Hal ini dapat bervariasi karena kompleksitas dari permasalahan.

VI. Ucapan Terima Kasih

Puji syukur penulis panjatkan kepada Allah SWT karena atas pertolongan, izin, berkat, rahmat, dan hidayah-Nya, penulis dapat menyelesaikan tugas makalah mata kuliah IF2120 Matematika Diskrit dengan lancar dan mendapatkan ilmu baru yang bermanfaat. Selain itu juga, penulis ingin menyampaikan ucapan terima kasih kepada Ibu Fariska Zakhralativa Ruskanda S.T.,M.T. selaku dosen pengajar Kelas 03 atas segala ilmu dan bimbingannya. Penulis juga ingin menyampaikan terima kasih kepada kedua orang tua dan juga kerabat yang telah memberikan dukungan dan doa kepada penulis.

Referensi

[1] Rinaldi Munir. 2020. *Graf (Bag.1)*. Diakses pada 2 Desember 2020 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf.

[2] Flawnson Tong. 2019, April 23. *Everything you need to know about Graph Theory for Deep Learning*. Diakses pada 2 Desember 2020 dari https://towardsdatascience.com/graph-theory-and-deep-learning-know-hows-6556b0e9891b.

[3] Rinaldi Munir. 2020. *Graf (Bag.2)*. Diakses pada 2 Desember 2020 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf.

[4] Tony Yiu. 2019, Juni 2. *Understanding Neural Networks*. Diakses pada 5 Desember 2020 dari https://towardsdatascience.com/understanding-neural-networks-19020b758230.

[5] Nielsen, Michael A. 2015. *Neural Networks and Deep Learning*. United States of America: Determination Press.

[6] Vijini Mallawaarachchi. 2017, Juli 8. *Introduction to Genetic Algorithms - Including Example Code*. Diakses pada 5 Desember 2020 dari https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3.

[7] Stanley, K.O., & Miikkulainen, R. 2002. *Evolving Neural Networks through Augmenting Topologies*. Evolutionary Computation, 10(2), 99-127. doi:10.1162/106365602320169811

[8] SethBling. 2015, Juni 13. *MarI/O - Machine Learning for Video Games*. Diakses pada 8 Desember 2020 pada https://www.youtube.com/watch?v=qv6UVOQ0F44.

[9] Tech With Tim. 2019, Agustus 3. *AI Teaches Itself to Play Flappy Bird - Using NEAT Python!* Diakses pada 9 Desember 2020 pada https://www.youtube.com/watch?v=OGHA-elMrxI.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 3 Desember 2020

Shaffira Alya Mevia 13519083