

Perbandingan Algoritma *Exhaustive*, Algoritma Genetika Dan Algoritma Jaringan Syaraf Tiruan *Hopfield* Untuk Pencarian Rute Terpendek

Rudy Adipranata¹, Felicia Soedjianto², Wahyudi Tjondro
Teknik Informatika, Fakultas Teknologi Industri, Universitas Kristen Petra
Jl. Siwalankerto 121-131, Surabaya
Telp. 8439040
Email : rudya@petra.ac.id¹, felis@petra.ac.id²

Abstract

Nowadays, there are many algorithms to explore the shortest path. Each algorithm has its own advantages and disadvantages, besides the advantages of each algorithm depend on its case. In this paper, we compare three of shortest path algorithms based on travelling salesman problem.

The algorithms which are compared in optimum pathfinding are exhaustive algorithm, genetic algorithm and neural network hopfield, which these three algorithms have different advantages and disadvantages. Here, the cases which are used can be created by the user by using the application facility.

The result from the examination of the comparison between three algorithms can be concluded that for general, genetic algorithm using the correct parameter input can give the most optimum results in shortest path and process time. Exhaustive algorithm gives the optimum result only in small case, while neural network hopfield gives disappointed result in big scales case.

Keywords : shortest path algorithm, exhaustive, genetic algorithm, hopfield neural network

1. Pendahuluan

Terdapat banyak algoritma untuk melakukan pencarian rute terpendek. Pemilihan algoritma yang paling optimum selalu menjadi permasalahan dalam pencarian rute terpendek, dimana masing-masing algoritma memiliki kelebihan dan kekurangannya masing-masing.

Dalam lingkup pencarian rute terpendek ini tidak dapat dikatakan secara langsung algoritma mana yang paling optimum untuk keseluruhan kasus, karena belum tentu suatu algoritma yang memiliki optimasi yang tinggi untuk suatu kasus memiliki optimasi yang tinggi pula untuk kasus yang lain. Optimasi yang mencakup efisiensi waktu proses kerja algoritma, waktu tempuh yang diperlukan untuk mencapai tujuan akhir dan jarak tempuh yang paling pendek ini selalu tergantung dari setiap kondisi permasalahan yang ada, dan kondisi yang paling mempengaruhi adalah banyak titik.

Dalam penelitian ini akan dilakukan perbandingan diantara beberapa algoritma yang ada untuk kasus TSP (*Travelling Salesman Problem*). Pada *Travelling Salesman Problem* ini terdapat n buah kota yang harus dilalui semuanya oleh seorang *salesman* dan setelah melalui semua kota harus kembali ke kota pertama kali dia berangkat. Dalam perjalanannya tersebut, *salesman* haruslah memilih rute yang terpendek. Dalam skala kecil (jumlah kota < 5), dapat digunakan metode *exhaustive algorithm* untuk mencari rute terbaik dari setiap jarak tempuh yang ada. Pendapat tersebut didapat secara mudah ditinjau dari rumus pencarian jumlah kemungkinan rute yang ada dari metode *exhaustive algorithm*, yaitu nilai faktorial dari jumlah kota-1 atau $(n-1)!$, namun kasus akan menjadi berbeda jika jumlah titik menjadi semakin banyak. Pada makalah ini akan dibandingkan antara metode *Exhaustive Combinational*, *Genetic Algorithm* dan *Neural Network Hopfield*.

2. Algoritma *Exhaustive Combinational*

Pada prinsipnya metode ini merupakan penggabungan antara *depth first search* dengan pelacakan mundur (*backtracking*), yaitu pelacakan yang bergerak ke belakang menuju pada suatu keadaan awal. Nilai dari pengujian yang dilakukan adalah ‘ya’ atau ‘tidak’. Dengan kata lain metode ini adalah pencarian buta yang mencari semua kemungkinan yang ada dari permasalahan yang diproses. (Kristanto, 2004)

Secara garis besar cara kerja algoritma ini adalah sebagai berikut:

1. Membangkitkan semua kemungkinan solusi (membangkitkan suatu titik tertentu atau lintasan tertentu dari keadaan awal).
2. Melakukan pengujian untuk melihat apakah titik tersebut benar-benar merupakan solusi yang diharapkan dengan cara membandingkan titik tersebut atau titik akhir dari suatu lintasan yang dipilih dengan kumpulan tujuan yang diharapkan.
3. Jika solusi tersebut memenuhi syarat yang diinginkan, keluar. Jika tidak, ulangi kembali langkah yang pertama untuk mencari kemungkinan yang lain.

3. Algoritma Genetika

Algoritma genetika adalah algoritma pencarian heuristik yang didasarkan pada mekanisme evolusi biologis. Keberagaman pada evolusi biologis adalah variasi dari kromosom dalam individu organisme. Variasi kromosom ini akan mempengaruhi laju reproduksi dan tingkat kemampuan organisme untuk tetap hidup (Kristanto, 2004). Pada dasarnya ada 4 kondisi yang sangat mempengaruhi proses evaluasi, yaitu:

1. Kemampuan organisme untuk melakukan reproduksi.
2. Keberadaan populasi organisme yang bisa melakukan reproduksi.
3. Keberagaman organisme dalam suatu populasi.
4. Perbedaan kekuatan dan kemampuan organisme untuk bertahan hidup.

Individu yang lebih kuat (*fit*) akan memiliki tingkat *survival* atau tingkat daya bertahan hidup yang lebih tinggi. Selain itu individu yang semakin kuat akan memiliki tingkat reproduksi yang lebih tinggi jika dibandingkan dengan individu yang kurang *fit*. Pada kurun waktu tertentu (sering dikenal dengan istilah generasi), populasi secara keseluruhan akan memuat lebih banyak organisme yang *fit*.

Pada algoritma ini teknik pencarian dilakukan sekaligus atas sejumlah solusi yang mungkin, dikenal dengan istilah populasi. Di dalam populasi tersebut terdapat individu yang disebut dengan istilah kromosom. Kromosom-kromosom tersebut merupakan suatu solusi yang masih berbentuk simbol, biasanya adalah bilangan biner. Kromosom-kromosom ini akan mengalami evolusi melalui sejumlah iterasi yang disebut dengan generasi. Dalam setiap generasi kromosom akan mengalami proses evaluasi dengan menggunakan alat ukur yang disebut dengan fungsi *fitness*. Dalam algoritma genetik, istilah kromosom merujuk pada kandidat solusi dari suatu masalah, sering dilambangkan sebagai sebuah string yang terdiri dari bit. Gen adalah sebuah bit tunggal atau sebuah blok yang terdiri dari bit-bit yang berdampingan yang melambangkan elemen tertentu dari kandidat solusi. *Crossover* adalah pertukaran material genetik antara kromosom dari parent. Mutasi adalah menukar sebuah *allele* pada *locus* (posisi) random dengan dengan *allele* lainnya, misal 0 menjadi 1.

Setiap generasi akan menghasilkan kromosom-kromosom baru yang dibentuk dari generasi sebelumnya dengan menggunakan operator reproduksi (*reproduction*), kawin silang (*crossover*), dan juga mutasi (*mutation*). Nilai *fitness* dalam suatu kromosom akan menunjukkan kualitas kromosom dalam populasi tersebut. Generasi berikutnya dikenal dengan istilah anak (*offspring*) yang terbentuk dari gabungan 2 kromosom generasi sekarang yang bertindak sebagai induk (*parent*) dengan menggunakan operator penyilangan (*crossover*).

Generasi-generasi baru dibentuk dengan cara:

- Melakukan proses seleksi sesuai dengan nilai obyektif dari kromosom *parent* dan juga kromosom *offspring*.
- Membuang beberapa kromosom sehingga jumlah populasi akan kembali menjadi kromosom.

Demikian generasi yang baru terus dihasilkan sesuai dengan besar generasi yang ditentukan dan setelah melalui beberapa generasi maka algoritma ini akan konvergen ke kromosom terbaik.

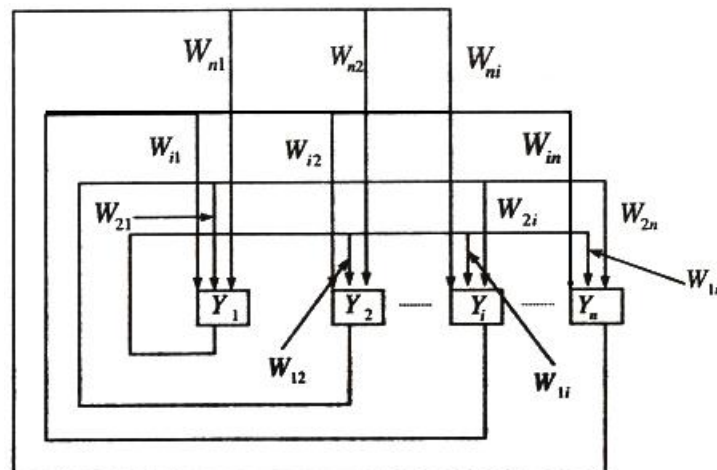
5. Algoritma Neural Network Hopfield

Algoritma *Neural Network Hopfield* pertama kali dikenalkan oleh John Hopfield dari *California Institute of Technology* pada tahun 1982. John Hopfield merancang sebuah jaringan syaraf tiruan yang kemudian dikenal dengan nama jaringan *Hopfield*. Dalam jaringan *Hopfield* semua *neuron* berhubungan penuh. *Neuron* yang satu mengeluarkan *output* dan kemudian menjadi *input* untuk semua *neuron* yang lain. Proses penerimaan sinyal antara *neuron* ini secara *feedback* tertutup terus menerus sampai dicapai kondisi stabil (Kristanto 2004)

Dalam model diskritnya, jaringan *Hopfield* bobot sinaptiknya menggunakan vektor biner berdimensi n atau $\{0,1\}^n$. Model semacam ini berisi n *neuron* dan jaringannya terdiri dari $n(n-1)$ interkoneksi dua jalur. Secara matematis konsep di atas dapat disajikan dalam matriks simetris $n \times n$ dengan diagonal utamanya bernilai 0. Dengan kata lain, setiap *neuron* tidak memberi *input* kepada dirinya sendiri.

Kemampuan jaringan syaraf tiruan untuk mengenali pola ditentukan dengan rumus $0,15 \times n$. n adalah banyaknya dimensi yang digunakan. Misalkan dimensi yang digunakan adalah 100 maka pola yang akan dikenali sebanyak 15 buah. Dimensi adalah suatu vektor yang mewakili banyaknya baris dan kolom yang digunakan. Dari contoh di atas berarti baris yang digunakan berjumlah 100 dan kolom yang digunakan berjumlah 100 juga. Jaringan syaraf tiruan digunakan untuk mengenali berbagai input atau masukan yang diberikan kepadanya. Misalnya, jaringan tersebut dilatih untuk mengenali berbagai versi angka 2. Setelah mengalami cukup latihan, maka jaringan itu dapat menghasilkan angka 2 secara sempurna dari masukan angka 2 yang rusak atau kena *distorsi*.

Model atau arsitektur dari jaringan *Hopfield* ditunjukkan pada gambar 1, dimana dalam gambar tersebut terdapat suatu jaringan yang terdiri dari N buah simpul yang mengandung *nonlinearitas hard limiter* dan *input* serta *output* biner yang dapat bernilai 1 atau -1.



Gambar 1. Arsitektur Jaringan Hopfield

Dari gambar diatas, dapat dijelaskan bahwa *output* setiap simpul diumpan balikkan kepada *input* dari simpul-simpul lainnya melalui bobot koneksi W_{ij} yang tetap. Nilai W_{ij} mula-mula diinisialisasi menggunakan algoritma *Hopfield* untuk sebuah pola yang masuk.

Pola yang tidak dikenal dimasukkan ke dalam jaringan sekali saja pada waktu kondisi nol. Kemudian jaringan akan beriterasi sampai *output* yang dihasilkan bersifat *konvergen*. *Konvergen* adalah pola *output* yang tidak berubah sampai iterasi selesai. Pola yang dinyatakan oleh simpul-simpul *output* setelah jaringan konvergen adalah *output* jaringan syaraf tiruan.

Pada kasus *Travelling Salesman Problem* dengan menggunakan jaringan syaraf tiruan *Hopfield*, langkah pertama adalah melakukan pemetaan antara permasalahan ke dalam struktur jaringan (Freeman and Skapura, 1991). Hal ini dilakukan dengan merepresentasikan n *node* (*neuron*) jaringan menjadi n kemungkinan urutan posisi yang dilalui pada sebuah kota. Untuk n kota maka dibutuhkan n *node*. Sebagai contoh terdapat 5 kota, maka lima node yang berisi : 0 0 0 1 0 menunjukkan sebuah kota yang mempunyai urutan keempat untuk dikunjungi. Dengan cara ini maka terdapat 5 *group* yang masing-masing berisi 5 *node* seperti contoh di atas dimana masing-masing *group* merepresentasikan sebuah kota. Untuk lebih jelas dapat dilihat pada gambar di bawah dimana pada gambar tersebut terdapat matriks 5x5 yang menunjukkan urutan posisi kota saat dilalui oleh *salesman*. Dari gambar tersebut dapat dilihat bahwa urutan kota yang dilalui adalah B-A-E-C-D.

1	2	3	4	5	
0	1	0	0	0	A
1	0	0	0	0	B
0	0	0	1	0	C
0	0	0	0	1	D
0	0	1	0	0	E

Gambar 2. Matrik Posisi 5 Kota

Untuk menentukan apakah sebuah rute merupakan rute terpendek pada jaringan *Hopfield* ini digunakan sebuah fungsi energi, dimana jika nilai energi yang dihasilkan kecil, menunjukkan bahwa rute tersebut minimum. Berikut adalah formula yang digunakan dalam pendekatan untuk mencari nilai fungsi energi E yang minimum:

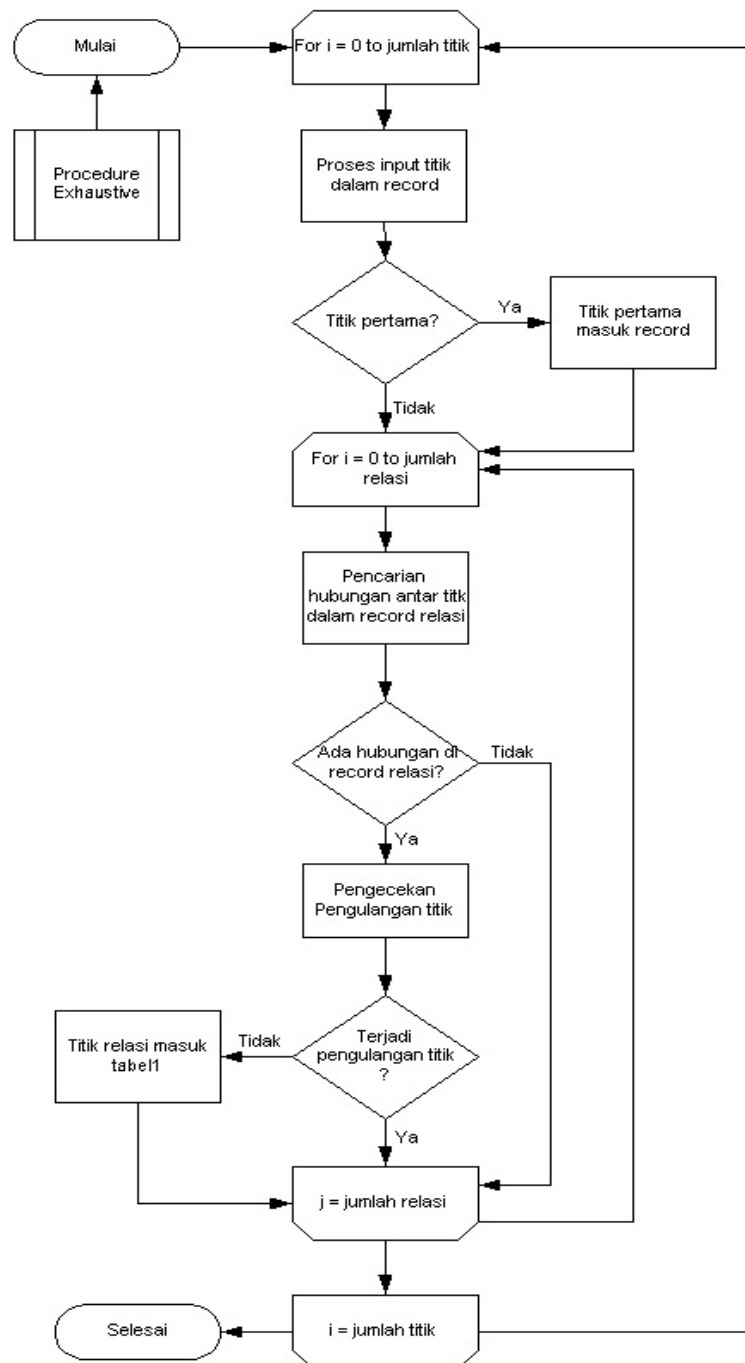
$$E = \frac{A}{2} \sum_{X=1}^n \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n v_{Xi} v_{Xj} + \frac{B}{2} \sum_{i=1}^n \sum_{X=1}^n \sum_{\substack{Y=1 \\ Y \neq X}}^n v_{Xi} v_{Yi} + \frac{C}{2} \left(\sum_{X=1}^n \sum_{i=1}^n v_{Xi} - n \right)^2 + \frac{D}{2} \sum_{X=1}^n \sum_{\substack{Y=1 \\ Y \neq X}}^n \sum_{i=1}^n d_{XY} v_{Xi} (v_{Y,i+1} + v_{Y,i-1}) \quad (1)$$

Pada formula di atas, notasi X dan Y menunjukkan kota, sedangkan notasi i dan j menunjukkan posisi kota tersebut pada rute. n adalah jumlah kota dan d_{XY} adalah jarak antara kota X dan Y . Konstanta A , B , C dan D adalah konstanta yang harus ditentukan tergantung jumlah kota. Dari formula di atas dilakukan perhitungan terhadap nilai fungsi energi E hingga didapatkan nilai E yang minimum. Apabila telah didapatkan nilai fungsi energi yang minimum, diharapkan hasil dari proses algoritma ini memberikan hasil berupa rute yang terpendek.

3. Desain Sistem

Secara keseluruhan, sistem terdiri dari tiga bagian yaitu untuk algoritma *Exhaustive*, Algoritma Genetika dan *Hopfield*. Untuk algoritma *Exhaustive*, terdiri dari dua bagian yaitu pencarian semua kombinasi dan perhitungan bobot rute yang didapat. Untuk pencarian

kombinasi, dapat digambarkan dengan menggunakan *flowchart* seperti terdapat pada gambar di bawah ini.

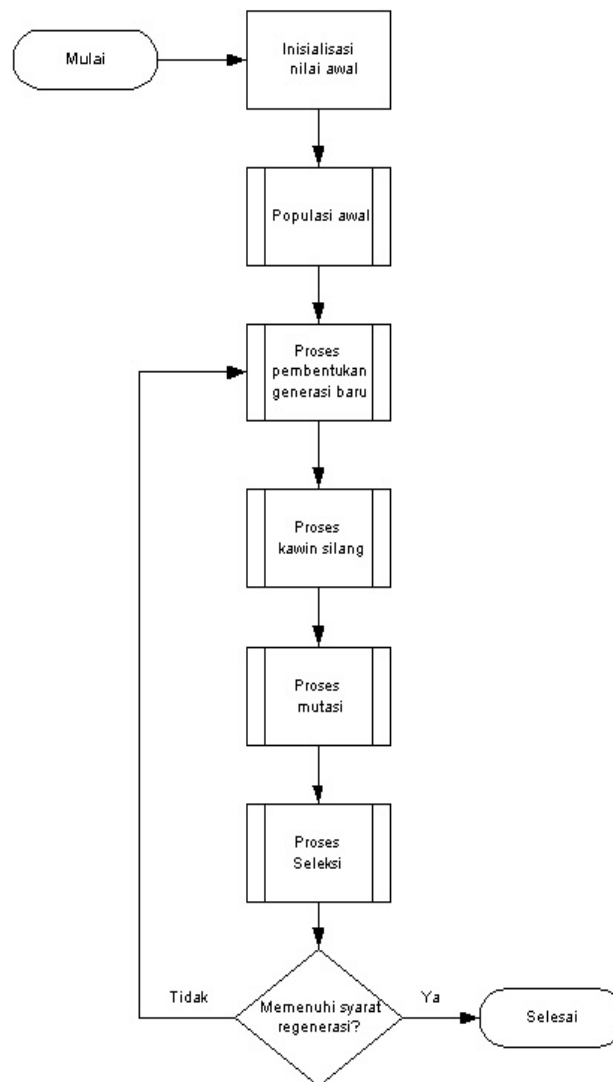


Gambar 3. *Flowchart* Pencarian Kombinasi Pada Algoritma *Exhaustive*

Langkah berikut setelah pencarian kombinasi adalah perhitungan bobot atau jarak total dari titik-titik yang telah terbentuk dalam suatu rute. Cara penghitungan bobot ini dilakukan dengan membandingkan dua titik dalam rute yang ditemukan dengan *record* relasi satu per satu, apabila kedua titik dalam rute tersebut sesuai dengan kedua titik yang terdapat dalam

record relasi, maka dilakukan penjumlahan terhadap bobot dalam *record* relasi dengan bobot total. Demikian seterusnya dilakukan pengulangan hingga titik terakhir dalam rute.

Untuk algoritma Genetika, *flowchart* secara keseluruhan dapat dilihat pada gambar 4.



Gambar 4. *Flowchart* Algoritma Genetika

Pada algoritma Genetika, langkah pertama adalah melakukan penentuan nilai awal (inisialisasi). Bagian penentuan nilai awal ini merupakan *input* yang dilakukan oleh pengguna sendiri. *Input* yang perlu dimasukkan dalam algoritma Genetika ini meliputi:

- Penentuan besar populasi dalam satu generasi.
- Penentuan banyak generasi yang akan dilakukan.
- Penentuan besar *crossover probability* (peluang terjadinya kawin silang).
- Penentuan besar *mutation probability* (peluang mutasi).

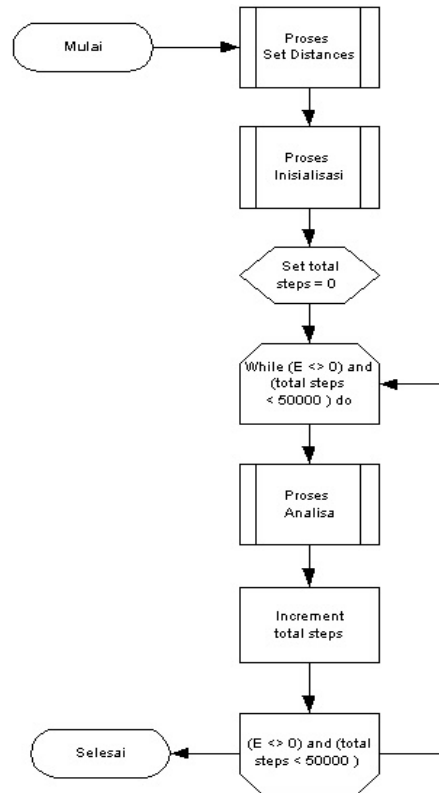
Setelah melakukan inisialisasi, proses berikutnya adalah proses pembentukan populasi awal. Proses ini berfungsi untuk membentuk populasi generasi pertama. Pembentukan populasi generasi pertama ini dilakukan dengan cara mengisi kromosom-kromosom yang ada secara *random*/acak dari semua titik yang ada. Pada saat pembentukan ini perlu dilakukan pengecekan agar bentuk populasi awal tersebut *valid*. Bentuk kromosom yang *valid* adalah memiliki panjang yang sesuai dengan jumlah titik dan melalui semua titik.

Proses ketiga adalah proses seleksi, dimana setelah terbentuk populasi awal, maka selanjutnya hasil populasi awal itu akan diseleksi. Metode seleksi yang digunakan dalam algoritma Genetika ini adalah *roulette wheel selection*. Cara kerja *roulette wheel selection* adalah seperti berikut: pertama-tama secara acak (*random*) akan dipilih sebuah nilai diantara 0 dan 1, kemudian nilai tersebut akan dikalikan dengan nilai total *fitness* seluruh individu dalam generasi tersebut. Nilai hasil perkalian antara bilangan *random* dan total *fitness* tersebut dianggap sebagai titik yang ditunjuk oleh jarum pada *roulette wheel*. Untuk mendapatkan individu yang akan dipilih, maka *fitness* dari individu anggota-anggota generasi tersebut akan dijumlahkan mulai dari individu pertama sampai jumlah *fitness* tersebut melebihi nilai hasil perkalian antara bilangan *random* dan total *fitness*. Dengan demikian individu yang memiliki *fitness* yang tinggi akan mempunyai kesempatan lebih besar untuk terpilih, karena semakin besar nilai *fitness* suatu individu, semakin besar pula pengaruhnya terhadap penjumlahan *fitness* individu-individu dalam populasi tersebut.

Setelah melakukan proses seleksi, maka hasil dari proses tersebut akan digunakan dalam proses *crossover*. Sebelum melakukan proses *crossover* dilakukan pengundian dengan bilangan *random* untuk setiap kromosom, apakah kromosom tersebut mengalami *crossover* atau tidak. Jika dari proses pengundian menunjukkan bahwa terjadi *crossover* maka akan dibuat bilangan *random* lainnya untuk menentukan dimana *crossover* akan terjadi. Posisi *crossover* dipilih dengan batasan antara 0 hingga panjang kromosom dan pada titik itulah maka *crossover* akan dilakukan antara dua kromosom. Proses *crossover* sendiri dilakukan dengan menyalin isi kromosom *parent* 1 ke dalam kromosom *offspring* 1 dan isi kromosom *parent* 2 ke dalam kromosom *offspring* 2, mulai dari kromosom ke-2 sampai pada posisi dimana *crossover* akan terjadi. Selanjutnya akan dilakukan penyalinan isi kromosom *parent* 2 ke dalam kromosom *offspring* 1 dan isi kromosom *parent* 1 ke dalam kromosom *offspring* 2, mulai dari kromosom yang letaknya tepat setelah posisi *crossover* sampai dengan panjang kromosom terpendek. Setelah proses *crossover* dijalankan selalu dilakukan pengecekan apakah kromosom yang terkena *crossover* tersebut merupakan kromosom yang valid, dalam arti kromosom hasil *crossover* tersebut membentuk suatu rute. Jika terjadi pengulangan individu dalam kromosom, maka dilakukan proses normalisasi untuk membuat kromosom tersebut menjadi valid. Proses normalisasi menghapus individu yang berulang dan mengganti dengan individu yang terhilang dalam kromosom tersebut sehingga terbentuk suatu rute baru yang valid.

Pada saat penyalinan kromosom dilakukan, kromosom tersebut dapat mengalami mutasi, yaitu perubahan isi kromosom, dimana isi dari kromosom tersebut digantikan dengan suatu nilai yang dipilih secara acak dari titik-titik yang ada. Proses pertama yang dilakukan adalah membangkitkan bilangan acak antara 0 hingga 1 sejumlah kromosom yang ada. Setelah itu untuk melakukan mutasi dilakukan penelusuran setiap bilangan *random* yang telah dibangkitkan tersebut dan dibandingkan antara nilai *random* kromosom dengan peluang mutasi yang ditentukan dari awal dalam inisialisasi nilai awal, jika nilai *random* yang ada lebih kecil atau sama dengan nilai peluang mutasi, maka kromosom tersebut akan mengalami mutasi. Setelah proses mutasi, maka akan dilakukan lagi proses seleksi dan dilakukan pengecekan apakah proses keseluruhan telah selesai atau belum.

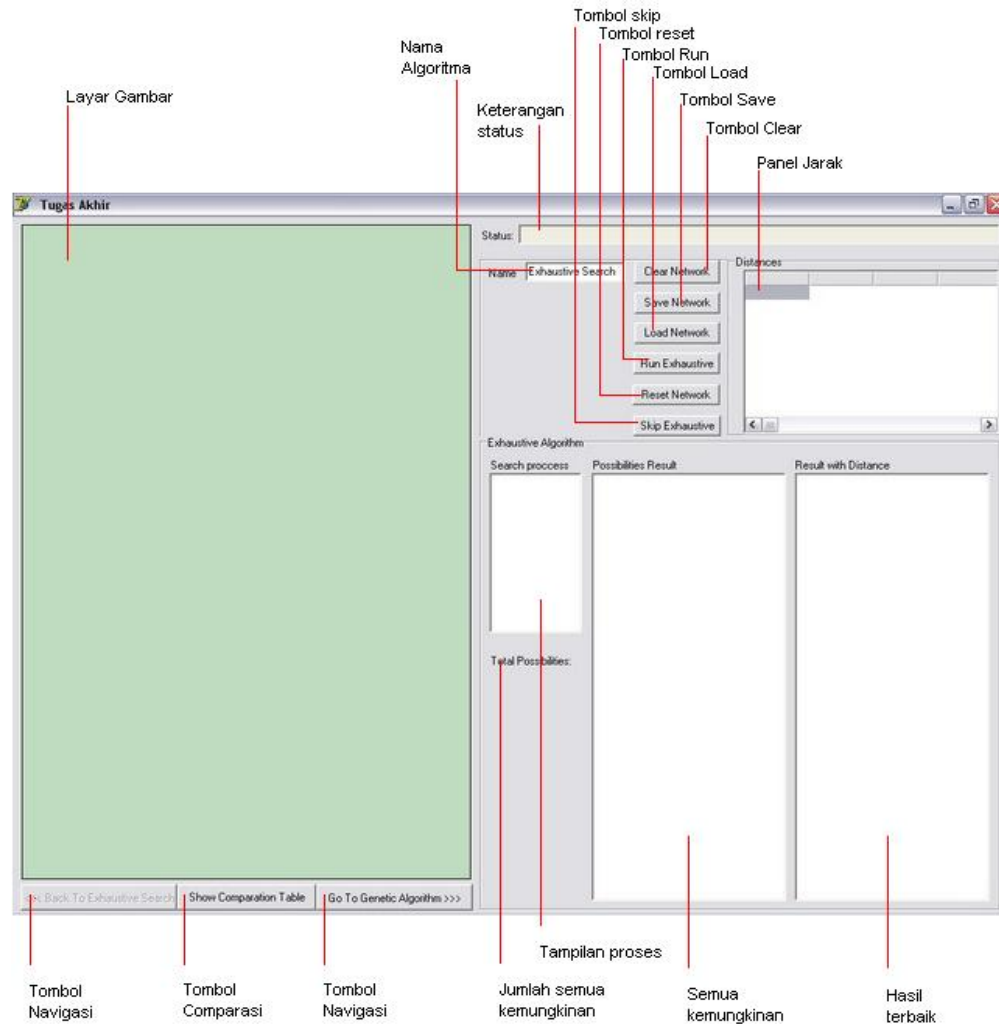
Untuk algoritma *Hopfield*, *flowchart* keseluruhan sistem dapat dilihat pada gambar 5. Algoritma *Hopfield* terdiri atas beberapa proses yang saling terkait satu sama lain. Proses yang pertama adalah penentuan jarak antara titik-titik yang ada. Pada proses ini dilakukan perhitungan secara matematis jarak antara dua titik yang diketahui. Setelah proses pencarian jarak dilakukan, maka proses selanjutnya yang dilakukan adalah proses inisialisasi. Proses inisialisasi ini merupakan proses untuk menentukan nilai-nilai awal yang diperlukan dalam proses analisa. Sedangkan proses terakhir adalah proses analisa yang digunakan untuk menganalisa atau melakukan penghitungan formula yang terdapat pada algoritma *Hopfield*



Gambar 5. *Flowchart Hopfield*

4. Implementasi dan Hasil Pengujian

Untuk implementasi sistem, digunakan bahasa pemrograman Borland Delphi 7. Sistem ini dikembangkan sehingga pengguna dapat melakukan penentuan kota sesuai dengan yang dikehendaki. Adapun hasil *interface* sistem dapat dilihat pada gambar di bawah ini. Pada *interface* tersebut terbagi menjadi dua bagian besar yaitu bagian kiri dan kanan. Pada bagian kiri adalah tempat pengguna menempatkan kota-kota yang hendak dilakukan pengujian. Sedangkan pada bagian kanan adalah tempat untuk memasukkan jarak antar kota, memilih algoritma yang hendak diuji, memasukkan parameter yang bersesuaian dengan algoritma yang dipilih serta melihat hasil rute yang terpendek beserta dengan jaraknya.



Gambar 6. Interface Aplikasi

Untuk pengujian, dilakukan perbandingan jarak yang dihasilkan serta waktu yang dibutuhkan oleh sistem untuk menjalankan algoritma tersebut. Jumlah kota yang dilakukan pengujian adalah 3, 5, 6, 8, 10 dan 15 kota. Sedangkan untuk algoritma Genetika dan *Hopfield*, masing-masing dilakukan 3 macam kombinasi parameter.

Tabel 1. Pengujian Tiga Algoritma

	Jumlah kota					
	3	5	6	8	10	15
<i>Exhaustive</i>	Jarak: 692 Waktu: < 0.001 dt	Jarak: 1011 Waktu: 0.02 dt	Jarak: 1086 Waktu: 0.11 dt	Jarak: 1267 Waktu: 10.91 dt	Jarak: 1267 Waktu: 782.626 dt	- terlalu banyak kombinasi
Genetika 1*)	Jarak: 692 Waktu: 0.941 dt	Jarak: 1011 Waktu: 0.922 dt	Jarak: 1086 Waktu: 0.991 dt	Jarak: 1267 Waktu: 1.427 dt	Jarak: 1353 Waktu: 1.532 dt	Jarak: 1903 Waktu: 2.053 dt
Genetika 2*)	Jarak: 692 Waktu: 0.16 dt	Jarak: 1011 Waktu: 0.17 dt	Jarak: 1086 Waktu: 0.21 dt	Jarak: 1267 Waktu: 0.291 dt	Jarak: 1816 Waktu: 0.31 dt	Jarak: 1680 Waktu: 3.985 dt

	Jumlah kota					
	3	5	6	8	10	15
Genetika 3*)	Jarak: 692 Waktu: 0.14 dt	Jarak: 1011 Waktu: 0.14 dt	Jarak: 1086 Waktu: 0.15 dt	Jarak: 1267 Waktu: 0.251 dt	Jarak: 1346 Waktu: 2.303 dt	Jarak: 1553 Waktu: 11.827 dt
<i>Hopfield</i> 1*)	Jarak: 692 Waktu: 0.01 dt	Jarak: 1143 Waktu: 0.01 dt	Jarak: 1396 Waktu: 0.03 dt	Jarak: 2101 Waktu: 0.06 dt	Jarak: 1981 Waktu: 0.1 dt	Jarak: 3118 Waktu: 0.651 dt
<i>Hopfield</i> 2*)	Jarak: 692 Waktu: < 0.001 dt	Jarak: 1011 Waktu: < 0.001 dt	Jarak: 1496 Waktu: 0.01 dt	Jarak: 1497 Waktu: 0.02 dt	Jarak: 2431 Waktu: 2.043 dt	Jarak: 2612 Waktu: 0.1 dt
<i>Hopfield</i> 3*)	Jarak: 692 Waktu: 0.631 dt	Jarak: 1143 Waktu: 1.592 dt	Jarak: 1518 Waktu: 2.774 dt	Jarak: 1712 Waktu: 0.01 dt	Jarak: 1991 Waktu: 0.01 dt	Jarak: 2511 Waktu: 1.111 dt

*)Parameter:

Genetika 1: population size: 10, max generation: 50, crossover probability: 25%, mutation probability: 10%

Genetika 2: population size: 10, max generation: 10, crossover probability: 25%, mutation probability: 10%

Genetika 3: population size: 5, max generation: 10, crossover probability: 25%, mutation probability: 10%

Hopfield 1: du0: 0.001, Dt: 0.0001, A: 1.2, B: 1.2, D: 1.5

Hopfield 2: du0: 0.1, Dt: 0.01, A: 1.2, B: 1.2, D: 1.5

Hopfield 3: du0: 0.0001, Dt: 0.000001, A: 1.2, B: 1.2, D: 1.5

5. Kesimpulan

Dari pengujian yang telah dilakukan, dapat ditarik kesimpulan yaitu secara umum algoritma Genetika bekerja lebih baik daripada kedua algoritma yang lain ditinjau dari jarak yang dihasilkan serta waktu yang dibutuhkan untuk melakukan perhitungan pada algoritma. Untuk jumlah kota yang banyak, agar algoritma Genetika dapat menghasilkan rute paling optimum, perlu dilakukan pemilihan parameter input yang tepat.

Secara umum, algoritma *Hopfield* tidak dapat memberikan rute yang optimum dibanding kedua algoritma yang lain, bahkan dengan kasus jumlah kota yang sedikit. Seperti algoritma Genetika, pada algoritma *Hopfield* juga diperlukan pemilihan parameter input yang tepat agar rute yang dihasilkan adalah yang paling optimum.

Ditinjau dari waktu proses secara umum, algoritma *Exhaustive* adalah algoritma yang membutuhkan waktu terlama, diikuti oleh algoritma Genetika serta proses paling cepat adalah algoritma *Hopfield*.

6. Daftar Pustaka

- Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization And Machine Learning*. Massachusetts: Addison-Wesley Publishing Company Inc.
- Kristanto, Andi. (2004). *Jaringan Syaraf Tiruan*. Yogyakarta: Gava Media.
- Mitchell, M. (1996). *An Introduction to Genetic Algorithms (2nd ed.)*. London : The MIT Press.
- Freeman, J. A and Skapura, D. M. (1991). *Neural Networks Algorithms, Applications, and Programming Techniques*. Massachusetts: Addison-Wesley Publishing Company Inc.