PENDEKATAN BARU PENYELESAIAN TRAVELING SALESMEN PROBLEM DENGAN JARINGAN SYARAF TIRUAN

Julian Supardi [1)] , Rani Famela[2)]

[1,2)] *Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Sriwijaya*

e-mail: Julian@unsri.ac.id

ABSTRACT

Problems in Solving Traveling Salesman Problem (TSP) include the slow turnaround time and the small number of cities that can be solved optimally. This study aimed to try a new approach to solve TSP with Artificial Neural Networks. This new approach was performed hybridizing Hopfield algorithm with Djikstra algorithm. Thus the city need to be divided into several zones. Each zone sought the shortest route using Hopfield, then algorithms Djikstrab is used. From the test results, the accuracy of the approach developed different for each number of cities in a single zone. The ideal number for one zone were 1 or 3 cities, but when 2,4, or 5 cities in one zone showed bad result. However these results have improved settlement outcomes TSP with Hopfield algorithm.

Key Word : *Traveling Salesman Problem, TSP, Hopfiled, Djikstra*

I. Latar Belakang

Traveling Salesman Problem (TSP) dikenal sebagai salah satu masalah optimisasi yang cukup banyak aplikasinya di dunia nyata dan banyak menarik perhatian pada peneliti sejak beberapa dekade terdahulu (Eisele et.al, 1994, Reinelt, G, 1994). Secara sederhana, TSP dapat dideskripsikan sebagai berikut: misalkan seorang sales ingin menyelesaikan urusan bisnisnya ke beberapa kota yang ingin ditujunya. Ia mulai dari kota ia berasal dan ingin mengunjungi kota-kota yang ingin ditujunya tepat sekali kemudian kembali lagi ke kota dia berasal tanpa melalui kota yang sudah dilaluinya. Selesman tersebut harus menentukan lintasan yang harus ia lalui hingga diperoleh jarak tempuh terpendek.

Meskipun TSP sepertinya mudah dipaparkan, akan tetapi sangat sulit untuk diselesaikan. Bahkan hingga saat ini belum ada satupun metode eksak yang dapat menjamin menghasilkan nilai optimal untuk sebarang masalah dalam waktu secara polynomial. Oleh sebab itulah, TSP dikarakteristikan ke dalam kelas NP-*complete* (Aarth, E. H. L. dan Lenstra, J. K , 1997). Berdasarkan hal tersebut, banyak peneliti lebih memusatkan kepada pengembangan metode-metode *heuristic* seperti *simulated annealing (*Kickpatrick, 1983), *Ant Algorithm* (Dorigo M. dan Gambardella, L. M, 1997) dan *Tabu seach* (Aarth, E. H. L. dan Lenstra, J. K , 1997) untuk menyelesaikan permasalahan TSP.

JST pada dasarnya adalah sebuah model pemrosesan yang mekanisme kerjanya menyerupai prinsif kerja sistem syaraf otak manusia (Schalkoff, Robert,1992). Secara umum prinsif kerja dari JST tersebut adalah melakukan pembelajaran untuk memperoleh pengetahuan, selanjutnya pengetahuan tersebut digunakan untuk menyelesaikan permasalahan serupa. Permasalahan yang dapat diselesaikan dengan JST pada umumnya adalah permasalahan-permasahan yang memiliki pola tertentu (Supardi, Julian 2005). Karena itu, JST sering diaplikasikan pada pengenalan pola seperti

pola huruf, pola kalimat, predeksi harga saham, dan lain sebagainya (Schalkoff, Robert,1992).

TSP merupakan sebuah permasalahan yang memiliki pola, yakni *pertama* setiap kota hanya dilalui satu kali, *kedua* keluaran dari satu kota merupakan masukan bagi kota berikutnya, dan *ketiga* setiap kali perjalanan akan selalu kembali ke tempat semula. Dengan demikian dapat dipredeksi bahwa JST mampu diaplikasikan untuk menyelesaikan permasalahan TSP. Berkaitan dengan itu, maka beberapa peneliti telah mencoba melakukannya. Namun, hasil yang diperoleh masih belum mencapai nilai optimal. Permasalahan yang timbul dapat diklasifikasikan menjadi dua tipe, yakni *pertama* waktu penyelesaian yang lambat dan *kedua* masih sedikitnya jumlah kota yang dikunjungi dan apabila jumlah kota diperbesar, maka tingkat kesalahan perhitungan semakin besar.

Berangkat dari kondisi tersebut, penelitian ini ingin mencoba menerapkan pendekatan baru dalam menyelesaikan TSP dengan hopfiled. Pendekatan yang digunakan adalah mengkombinasikan Algoritma hopfield dan Algoritma Djikstra dalam menentukan solusi TSP.

II. Traveling Salesman Problem

Traveling *Salesman Problem* (TSP) dikenal sebagai salah satu masalah optimisasi yang cukup banyak aplikasinya di dunia nyata dan banyak menarik perhatian pada peneliti sejak beberapa dekade terdahulu (Eisele JG et all., 1994 dan Reinelt, 1994). Secara sederhana, TSP dapat dideskripsikan sebagai berikut: misalkan seorang sales ingin menyelesaikan urusan bisnisnya ke beberapa kota yang ingin ditujunya. Ia mulai dari kota ia berasal dan ingin mengunjungi kota-kota yang ingin ditujunya tepat sekali, kemudian kembali lagi ke kota dia berasal tanpa melalui kota yang sudah dilaluinya. Salesman tersebut harus menentukan lintasan yang harus ia lalui hingga diperoleh jarak tempuh terpendek.

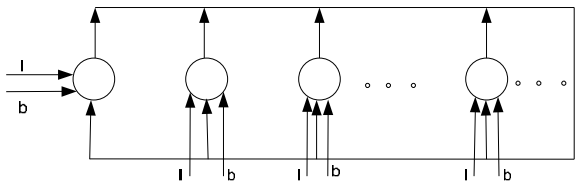
III. Algoritma Hopfield

Jaringan Hopfield merupakan salah satu model jaringan syaraf tiruan, yang terdiri dari beberapa neuron yang saling terhubung satu sama lain yang kemudian dapat diterapkan suatu *associative memory* dengan memperhitungkan fungsi energi ke dalam jaringan. Konfigurasi jaringan Hopfield dikenal sebagai *recurrent network.* Dalam model diskritnya, bobot sinaptik jaringan hopfield menggunakan vektor biner dimensi n atau $a\{0,1\}^n$. Model semacam ini berisi N neuron dan jaringannya terdiri dari n (n-1) interkoneksi dua jalur. Secara matematis konsep tersebut dapat disajikan dengan matriks simetris n x n dengan diagonal utamanya bernilai 0. Dengan kata lain setiap neuron tidak memberi input kepada dirinya sendiri. Gambar 3-1 berikut menunjukkan matrik koneksi antar neuron yang ada di jaringan *hopfield.*

$$A = \begin{vmatrix} 0 & X_{12} & X_{13} & \dots & X_{1N} \\ X_{21} & 0 & X_{23} & \dots & X_{2N} \\ X_{31} & X_{32} & 0 & \dots & X_{3N} \\ \dots & \dots & \dots & \dots & \dots \\ X_{n1} & X_{n2} & X_{n3} & \dots & 0 \end{vmatrix}$$

Gambar 3-1 Matriks Koneksi *Neuron* di Jaringan *Hopfield*

Sementara itu, topologi dari jaringan hopfield dapat dilihat pada Gambar 3-2 berikut ini :



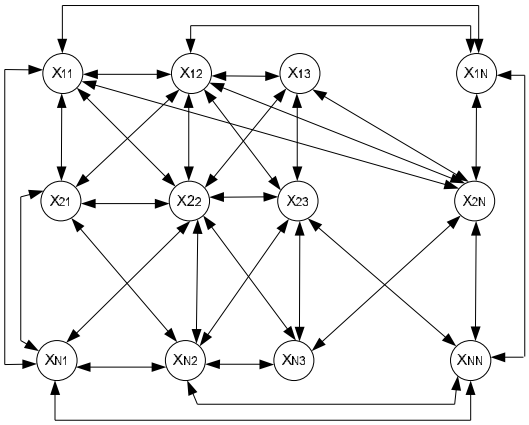
Ket:
i: external Input
b: Bias

Gambar 3-2 Jaringan *Hopfield* (Hopfield dan Tank, 1985)

Dari Gambar 3-2 tersebut tampak bahwa neuron yang satu mengeluarkan output dan kemudian menjadi input bagi neuron yang lain. Output setiap simpul diumpanbalikan ke input neuron yang lainnya melalui bobot koneksi w_{ij} yang tetap. Nilai w_{ij} mula-mula diinisalisasi menggunakan rumus yang diberikan hopfield untuk seluruh pola-pola.

3.1. Arsitektur Jaringan Hopfield untuk TSP

Misalkan terdapat N kota, maka akan terdapat N * N neuron di dalam jaringan hopfield, yakni X11 hingga XNN. Keluaran dari masing-masing Neuron sebut saja dengan nama X'ij yang menunjukan hubungan elemen di dalam matriks V. Masing-masing Neuron akan kembali kepada dirinya sendiri dan menjadi input bagi neuron yang lain. Lebih detail hal tersebut diberikan dalam Gambar 3-3 berikut :



Gambar 3-3 Jaringan Hopfield untuk TSP

3.2. Algoritma Hopfield untuk TSP

Berikut algoritma hopfield yang digunakan untuk menyelesaikan permasalahan TSP :

- Langkah 0 :
  - Inisialisasi aktifasi dari semua unit neuron;
  - inisialisasi Δt dengan nilai acak kecil
- Langkah 1 :
  jika kondisi penghentian belum terpenuhi, lakukan langkah 2 - 6
- Langkah 2 :
  lakukan langkah 3-5 sebanyak n^2 kali.
- Langkah 3 :
  Pilih unit neuron secara acak
- Langkah 4 :
  Lakukan pembaharuan unit neuron yang terpilih dengan rumus :

$$\Delta U_{xi} = \Delta t\left(-U_{ij(lama)} - A\sum_{j\neq i} V_{xj} - B\sum_{y\neq x} V_{yi} - C\left(n - \sum_{x}\sum_{j} V_{xj}\right) - D\sum_{y\neq x} d_{xy}\left(V_{y,i+1} + V_{y,i-1}\right)\right)$$

$$U_{xi\,(baru)} = U_{xi\,(lama)} + \Delta U_{xi}$$

- Langkah 5:
  Hitung keluaran masing-masing aktifasi dengan fungsi aktivasi:

$$V_{xi} = 0,5\left(1 + \tanh\left(\alpha U_{xi}\right)\right)$$

- Langkah 6 : Periksa kondisi berhenti, jika kondisi perhentian belum terpenuhi, kembali ke langkah 2 (Fausett, Laurene,1994)

IV. Algoritma Djikstra

Salah satu algoritma mencari lintasan terpendek yang paling terkenal adalah algoritma *djikstra* yang ditemukan oleh matematikawan Belanda, Edsger Wybe Djikstra pada tahun 1959. Algoritma *djikstra* menggunakan prinsip *greedy* dalam menentukan lintasan terpendeknya. Pada setiap langkah, ambil sisi yang berbobot minimum yang menghubungkan sebuah simpul yang sudah terpilih dengan sebuah simpul lain yang belum terpilih. Lintasan dari simpul asal ke simpul yang baru haruslah merupakan lintasan terpendek diantara semua lintasannya yang belum terpilih.

Algoritma ini termasuk algoritma pencarian graf yang digunakan untuk menyelesaikan masalah lintasan terpendek dengan satu sumber pada sebuah graf yang tidak memiliki cost sisi negatif, dan menghasilkan sebuah pohon lintasan terpendek. Algoritma dijkstra mencari lintasan terpendek dalam sejumlah langkah. Algoritma ini juga dapat digunakan untuk mencari total biaya(cost) dari lintasan terpendek yang dibentuk dari sebuah simpul ke sebuah simpul tujuan. Sebagai contoh, bila simpul pada graf merepresentasikan kota dan bobot sisi merepresentasikan jarak antara 2 kota yang mengapitnya, maka algoritma dijkstra dapat digunakan untuk mencari rute terpendek antara sebuah kota dengan kota lainnya. Berikut adalah skema umum dari algoritma dijkstra pada pencarian *shortest path* :

1. Buatlah 3 buah list, yaitu list jarak (list1), list simpul-simpul sebelumnya (list 2), dan list simpul yang sudah dikunjungi (list3), serta sebuah variable yang menampung simpul saat ini (current vertex).
2. Isi semua nilai dalam list jarak dengan tak hingga, kecuali simpul awal yang diisi dengan 0.
3. Isi semua nilai dalam list 2 dengan false
4. Isi semua nilai dalam list 3 dengan null

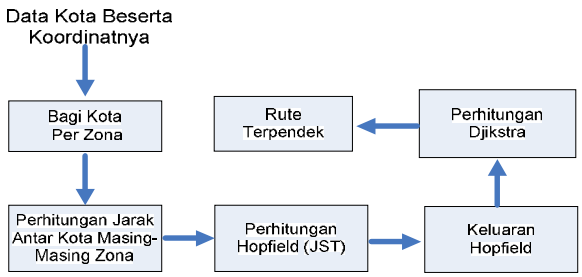
5. Current Vertex diisi dengan simpul awal(start).
6. Tandai current vertex sebagai simpul yang telah dikunjungi.
7. Update list 1 dan 2 berdasarkan simpul-simpul yang dapat langsung dicapai dari current vertex
8. Update current vertex dengan simpul yang paling dekat dengan simpul awal.
9. Ulangi langkah 6 sampai semua simpul sudah dikunjungi.

V. Metodologi Penelitian

Dalam penelitian ini, untuk menyelesaikan permasalahan digunakan dua Algoritma, yakni algoritma Hopfield dan Algoritma Djikstra. Langkah-langkah yang digunakan adalah:

1. Bagi masing-masing bidang menjadi empat zona
2. Inputkan koordinat masing-masing kota pada masing-masing zona
3. Hitung jarak terpendek antar kota pada setiap zona dengan algoritma hopfield
4. Hitung jarak terpendek untuk masing-masing zona dengan algoritma djikstra

Secara diagram langkah-langkah tersebut dapat disajikan sebagai berikut:

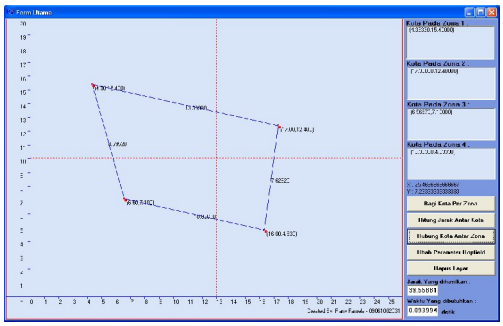


Gambar 5.1. Skema Pemrosesan Penyelesaian TSP

VI. Hasil dan Pembahasan

Dalam implementasi, Parameter untuk Hopfield dipilih sedemikian sehingga mendapat solusi yang optimal, pemilihan ini dilakukan dengan beberapa kali percobaan. Dalam hal inim pemilihan parameter yang kurang tepat sangat berpengaruh terhadap hasil yang diberikan. Pada proses pengujian ini, parameter yang digunakan untuk fungsi energi berturut-turut adalah A = 300, B = 300, C = 100, D = 10, Uo = 0.02, Uoo = -0.016, deltaT = 0.01. Sedangkan untuk iterasi maksimum yang digunakan adalah 8000 iterasi dan energi maksimumnya 100.

Pengujian dilakukan untuk beberapa kasus jumlah kota. Pertama, pengujian dilakukan untuk kasus dimana terdapat 1 kota pada masing-masing zona atau kelompok kota, ini berarti terdapat 4 kota yang akan dicari rute nya. Hasil yang diberikan dari 10 kali percobaan untuk kasus pertama ini dapat dilihat pada sebagai mana diberikan dalam Tabel 5.1.



Gambar 6.1. Hasil Pengujian dengan 4 Kota

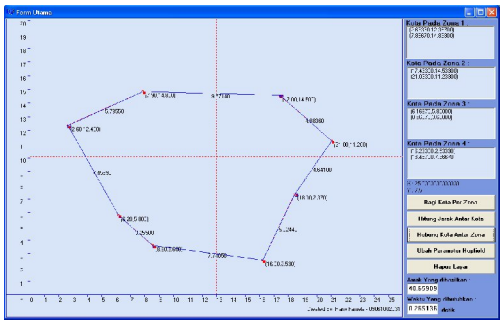
Tabel 6.1. Hasil Pengujian 4 Kota

No	Jumlah Kota	Jarak	Waktu (detik)	Rute Terpendek
1	4	39.5588	0.093994	Ya
2	4	31.6507	0.078125	Ya
3	4	46.7038	0.093994	Ya
4	4	27.03746	0.094	Ya
5	4	39.3689	0.062	Ya
6	4	36.0547	0.093994	Ya
7	4	23.74262	0.062011	Ya
8	4	30.69283	0.093994	Ya
9	4	32.33629	0.093994	Tidak
10	4	33.6042	0.093017	Ya

Dari Tabel 6.1 dapat dilihat bahwa keakuratan perangkat lunak dalam menemukan rute untuk kasus uji pertama adalah 90%. Hal ini berarti untuk jumlah kota = 4, perangkat lunak dapat memberikan solusi yang sangat baik dengan rata-rata waktu yang dibutuhkan adalah 0.085 detik.

Kasus uji kedua adalah jumlah kota yang terdapat pada masing-masing zona adalah 2 kota, hal ini berarti terdapat 8 kota yang akan dicari rute nya. Pengujian juga dilakukan sebanyak 10 kali sama

seperti kasus uji pertama. Hasil pengujian untuk kasus uji kedua ini dapat dilihat pada Gambar 6.2 dan Tabel. 6.2



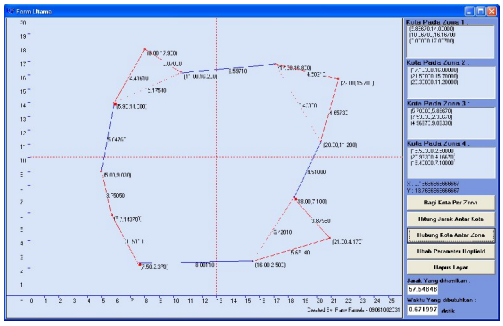
Gambar 6-2. Hasil Pengujian Kasus Uji Kedua

Tabel 6-2. Hasil Pengujian Kasus Uji Kedua

No	Jumlah Kota	Jarak	Waktu (detik)	Rute Terpendek
1	8	48.659	0.265136	Ya
2	8	-	-	Gagal
3	8	-	-	Gagal
4	8	-	-	Gagal
5	8	39.95918	0.391	Ya
6	8	-	-	Gagal
7	8	45.7985	0.329	Ya
8	8	47.61576	0.296997	Ya
9	8	-	-	Gagal
10	8	-	-	Gagal

Dari tabel 6.2 dapat dilihat dari 10 kali pengujian yang dilakukan untuk kasus uji kedua ini, perangkat lunak hanya mampu menemukan rute sebanyak 4 kali. Hal ini menunjukkan bahwa keakuratan perangkat lunak dalam menemukan rute untuk kasus uji kedua sangat kecil, hanya 40%. Sedangkan rata-rata waktu yang dibutuhkan adalah 0.320533 detik untuk masing-masing rute yang valid.

Kasus uji selanjutnya adalah kasus uji ketiga yaitu mengelompokkan 3 kota dalam satu zona atau kelompok kota. Sehingga pada masing-masing zona terdapat 3 buah kota. Berarti terdapat 12 kota yang akan dihitung rute nya. Gambar 6.3 dan Tabel 6.3 menunjukkan hasil pengujian kasus ketiga ini.



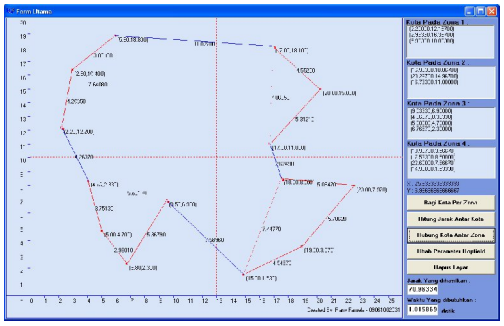
Gambar 6.3. Hasil Pengujian Kasus Uji Ketiga

Tabel 6.3. Hasil Pengujian Kasus Uji Ketiga

No	Jumlah Kota	Jarak	Waktu (detik)	Rute Terpendek
1	12	57.54848	0.671997	Ya
2	12	60.77401	0.718994	Ya
3	12	65.32976	0.733999	Ya
4	12	65.79063	0.843994	Ya
5	12	70.26989	0.672	Ya
6	12	67.7737	0.704	Ya
7	12	76.02676	0.70397	Ya
8	12	70.11869	0.687988	Ya
9	12	64.39	0.687988	Ya
10	12	-	-	Gagal

Dari Tabel 6.3 dapat dilihat bahwa dari 10 kali pengujian hanya 1 kali perangkat lunak gagal dalam menenemukan rute. Hal ini berarti keakuratan perangkat lunak dalam menyelesaikan kasus ketiga adalah 90%. Perangkat lunak dapat berjalan dengan baik untuk pengelempokan 3 kota untuk tiap-tiap zona. Dari tabel tersebut juga dapat diketahu bahwa rata-rata waktu yang dibutuhkan perangkat lunak menyelesaikan kasus ketiga untuk 12 kota ini adalah 0.713881.

Kasus uji terakhir adalah untuk jumlah kota keseluruhan 5,6,7,8,9,10, dan 11 dengan jumlah kota di masing-masing zona nya bervariasi dan lebih kecil atau sama dengan 3. pengujian dilakukan 10 kali untuk masing-masing jumlah kota dan di ambil rata-rata waktu yang dibutuhkan. Hasil yang didapat dilihat pada pada Gambar 6.4 dan rekapitulasi hasil pengujian disajikan dalam Tabel 6.4.



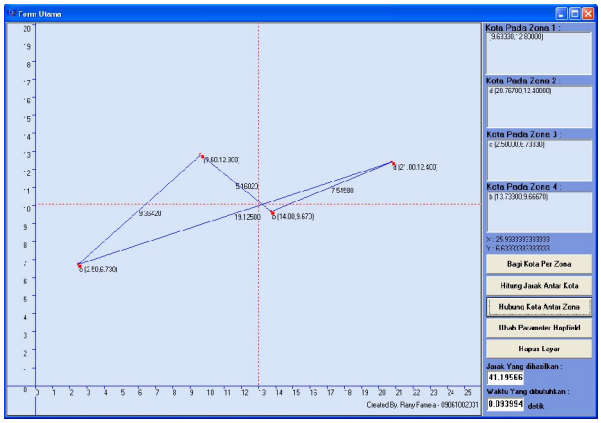
Gambar 6.4. Hasil Pengujian Kasus kota 5,6,7,8,9,10, dan 11 yang terdistribusi pada masing-masing zona.

Tabel 6.4. Rekapitulasi Hasil Pengujian Kasus kota 5,6,7,8,9,10, dan 11 yang terdistribusi pada masing-masing zona.

No	Jml Kota	Rata-rata waktu (detik)	Jumlah Rute	Tk Akurasi (%)
1	5	0.2273	9	90%
2	6	0.2778	9	90%
3	7	0.3688	7	70%
4	8	0.3962	8	80%
5	9	0.4487	7	70%
6	10	0.5441	6	60%
7	11	0.6161	7	70%

Nilai keakuratan untuk masing-masing jumlah kota pada Tabel 6.4 didapat dari hasil bagi antara jumlah rute terpendek yang ditemukan dengan banyaknya percobaan yang dilakukan. Dalam penelitian ini, pengujian dilakukan sebanyak 10 kali untuk masing-masing jumlah kota. Sedangkan rata-rata waktu nya didapat dengan mencari rata-rata waktu dari jumlah rute terpendek yang ditemukan. Kombinasi jumlah masing-masing kota dalam suatu zona juga menentukan rute yang dihasilkan.

Dari sejumlah kasus uji yang dilakukan, selain kombinasi dari jumlah masing-masing kota dalam suatu zona, rute terpendek yang ditemukan juga bergantung pada letak kota yang berada dekat dengan garis pemisah antar zona. Ketika kota terdekat antar zona yang ditemukan berada di zona persilangan dari zona sebelumnya, maka hasil yang diberikan bukan merupakan rute terpendek. Kondisi ini diperlihatkan dalam Gambar 6.5.



Gambar 6.5.Kondisi Kota di dekat Persilangan zona persilangan dari zona sebelumnya.

Pada Gambar 6.5. dapat dilihat, ketika kota terdekat yang ditemukan dari kota a yang berada di zona pertama adalah kota b yang berada di zona ke empat, maka urutan kota setelah kota a adalah kota b, bukan kota c atau d. padahal jika urutan kota setelah kota a adalah c atau d, maka rute yang dihasilkan akan lebih kecil dari pada 41,19566 sebagaimana diperlihatkan Gambar 6.5 tersebut.

Untuk menguji efektifitas dari hasil yang telah dijelaskan sebelumnya, maka hasil-hasil yang telah didapat akan dibandingkan dengan beberapa data dari literatur yang ada.
Tabel 6.6 memperlihatkan waktu Penyelesaian TSP dengan Hopfield (Gunadi, Kartika dan Iksan, peter 2001)

Jumlah Kota	Algoritma Exhaustive (1/100detik)	Jst Hopfield (1/100detik)
6	0	88
7	50	130
8	170	165
9	2470	270
10	25480	314
11	280280	410
12	3363360	470

Dari Tabel 6.5 dapat dilihat waktu yang dibutuhkan untuk menenemukan rute untuk jumlah kota mulai dari 6 hingga 12 kota adalah lebih besar dari 0,87 detik. Sedangkan waktu yang dihasilkan dari penelitian ini untuk jumlah kota yang sama adalah sekitar 0.2778 detik hingga 0.713881 detik.

Sementara itu Tabel 6.6 memperlihatkan perbandingan waktu Pengujian Tiga Algoritma (Adipranata, Rudy, et all)

Tabel 6.6 : Perbandingan Waktu dari Tiga Metode Untuk Penyelesaian TSP

	Jumlah Kota					
	3	**5**	**6**	**8**	**10**	**15**
Exhaustive	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: 782.626dt	- 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
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
Hopfield 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
Hopfield 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: 0.1dt
Hopfield 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

Dari tabel 6.6 dapat dilihat jarak yang dihasilkan dari perhitungan dengan algoritma hopfield untuk jumlah kota dari 5 hingga 15 kota sudah lebih besar dari algoritma yang lainnya (Genetika dan Exhaustive). Walaupun demikian, 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 (Adipranata, Rudy, et all). Sementar itu hasil dari penelitian ini, algoritma hopfield telah mampu menemukan rute hingga untuk 12 kota dengan parameter hopfield yang telah ditentukan sebelumnya.

VII. Kesimpulan

Kesimpulan yang dapat diperoleh dari penelitian ini adalah Hybrid Algoritma Hopfield dan algoritma Djikstra memberikan hasil yang lebih baik dalam penyelesaian permasalahan TSP jika dibandingkan dengan Algoritma Hopfield biasa.

Daftar Pustaka

Aarth, E. H. L dan Lenstra, J. K. 1997. Local *Search in Combinatorial Optimization.* London : John and Wiley.

Dorigo, M dan Gambardella, L. M. 1997. *Ant colonies for the traveling salesman problem*. *Biosystem*.

Eisele, J.G., Castaneda R, dan Galindo, O.1994. *Usefulness of Solution of the Travelling Salesman Problem in Typing of Biological Sequnces in a Clinical Laboratory Setting.*

Fausett, Laurene.1994. *Fundamentals of Neural Networks : Architectures, Algorithms, and Applications.* New Jersey: Prentice-Hall Inc.

Gunadi, Kartika. 2001. Jurnal Informatika. Jaringan Saraf Tiruan Sebagai Alternatif Untuk Penyelesaian Travelling Salesperson Problem. Vol. 2, No. 1, Mei 2001: 30 – 32

Hykin, Simon. *1999. Neural Network : A Comprehensive Foundation.* New Jersey: Prentice-Hall Inc.

J.J. Hopfield and D.W.Tank. 1985. *Neural Computation of decisions in Optimaziation Problems. Biol.Cyebern*, vol.52.

Kickpatrick, S., Gellat, C. D. dan Vecchi, P. 1983. *Optimization by Simulated Annealing*. *Science*.

Kusumadewi, S. 2004. *Membangun Jaringan Syaraf Tiruan Menggunakan Matlab & Excel Link*. Yogyakarta: Penerbit Graha Ilmu.

Larman, C. 2004. *Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition.* Addison Wesley Professional.

Reinelt, G.1994. *The Travelling Salesman : Computational Solutions for TSP Applications*. Berlin : Springer-Verlag.

Schalkoff, Robert. 1992. *Pattern Recognition: Statistical, Structural, and Neural Approaches*. Jhon Wiley & Sons, Inc.

Siang, JJ. 2005. *Jaringan Syaraf Tiruan dan Pemprogamannya Menggunakan Matlab.* Yogyakarta : Penerbit ANDI.

Supardi, Julian. 2005. Neural Network: *Feed Forward Training & Testing.*

TeknoInfo.Vol. 3 No. 1 Januari 2005.ISSN.1693-0010

Qu, Hong, Zhang Yi, and Huajin Tang. 2006. *Improving Local Minima of Columnar Competitive Models For TSPs*.IEEE Transacations on Sircuits and Systems vol 53. No.6 June 2006