STUDI PERBANDINGAN ALGORITMA *CHEAPEST INSERTION HEURISTIC* DAN *ANT COLONY SYSTEM* DALAM PEMECAHAN *TRAVELLING SALESMAN PROBLEM*

Rusdi Efendi, Siti Maulinda
Program Studi Teknik Informatika, Fakultas Teknik, Universitas Bengkulu
Jalan W.R Supratman Kandang Limun Bengkulu
Bengkulu 38371 A
Telepon : (0736) 344087, 21170 – 227
E-mail: r_efendi@yahoo.com, lindsay.maulinda@gmail.com

ABSTRAK
Travelling Salesman Problem (TSP) merupakan salah satu kasus graf klasik yang memecahkan suatu masalah perjalanan yang berangkat dari suatu titik awal dan kembali lagi ke titik awal tersebut. Permasalahannya adalah mencari suatu rute tertentu dengan perhitungan rute terpendek. Dengan kata lain, permasalahan TSP ini sama halnya dengan mencari sirkuit Hamilton dengan bobot total terkecil pada teorema graf. Studi ini membahas perbandingan algoritma Cheapest Insertion Heuristic dengan Ant Colony System dalam menyelesaikan kasus TSP berdasarkan analisis hasil waktu proses yang telah dilakukan.

Kata Kunci: TSP, Cheapest Insertion Heuristic, Ant Colony System

1. PENDAHULUAN

Travelling Salesman Problem (TSP) merupakan masalah untuk mencari rute atau lintasan terpendek yang bisa dilalui *salesman* yang ingin mengunjungi beberapa titik kota tanpa harus mendatangi kota yang sama lebih dari satu kali, dan kemudian akan kembali ke kota awal. Permasalahan TSP merupakan salah satu topik teori graf yang menarik bagi banyak peneliti.

Secara matematis, Graf *G* didefinisikan sebagai pasangan himpunan (V,E) yang dalam hal ini [1]:

V = himpunan tidak kosong dari simpul - simpul (*vertices* atau *node*): $\{v_1, v_2, ..., v_n\}$

E = himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul: $\{e_1, e_2, ..., e_n\}$

atau dapat ditulis singkat notasi $G = (V,E)$

Dalam teorema graf, permasalah TSP dapat digambarkan sebagai berikut : Bila diketahui suatu graf berbobot penuh dan lengkap (dimana tiap simpulnya merepresentasikan titik-titik kota, tiap sisi merepresentasikan tiap jalan dari satu kota ke kota lainnya, dan tiap bobot sisi merepresentasikan jarak atau biaya yang dibutuhkan untuk menempuh jalan dari satu kota ke kota lainnya), maka tujuan penyelesaian TSP adalah mencari sirkuit Hamilton yang bobot totalnya paling kecil.

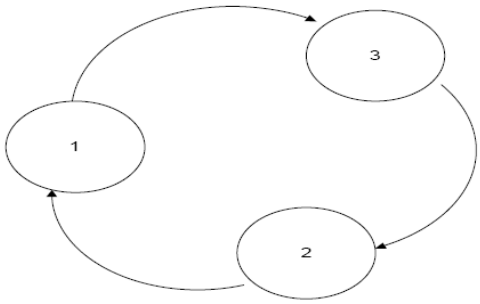
Ada banyak algoritma untuk memecahkan kasus TSP ini. Namun hingga saat ini, belum ditemukan algoritma yang benar-benar mangkus, karena penelitian yang telah dilakukan rata-rata hanya mencari pendekatan penyelesaian TSP dengan waktu yang relatif singkat, meskipun hasilnya tidak dijamin pasti optimal.

Dalam bahasan ini akan dikemukakan perbandingan antara dua algoritma dalam penyelesaian masalah TSP secara teori, yaitu perbandingan algoritma *Cheapest Insertion Heuristics (CIH)* dan *Ant Colony System* (ACS). Algoritma *Cheapest Insertion Heuristic* adalah algoritma yang membangun suatu tour dari sikel-sikel kecil dengan bobot minimal dan secara berturut-turut ditambah dengan titik baru sampai semua titik berhasil dilalui [2]. Sedangkan *Ant Colony System* (ACS) adalah bagian optimasi dari algoritma semut. Permasalahan TSP yang dibahas terbatas pada TSP *complete graph* dimana semua kota memiliki akses langsung ke kota lain dan diasumsikan seluruh kota terhubung satu sama lain dengan biaya/jarak antara 2 buah kota memiliki nilai yang sama meskipun dengan arah berbeda. Pembahasan ini dimaksudkan sebagai pembanding terhadap hasil penelitian terhadap Cheapest Insertion Heuristic (CIH)[3] dan Ant Colony System [4] yang telah dilakukan. Dengan demikian, penelitian ini diharapkan dapat menjadi salah satu referensi dalam pengembangan penyelesaian kasus TSP selanjutnya.

2. TEORI CHEAPEST INSERTION HEURISTIC (CIH)

Berikut ini adalah tata urutan algoritma CIH[5]:

1. Penelusuran dimulai dari sebuah kota pertama yang dihubungkan dengan sebuah kota terakhir.
2. Dibuat sebuah hubungan *subtour* antara 2 kota tersebut. Yang dimaksud *subtour* adalah perjalanan dari kota pertama dan berakhir di kota pertama, misal (1,3) → (3,2) → (2,1) seperti tergambar dalam Gambar 3.



Gambar 1. *Subtour* CIH

3. Ganti salah satu arah hubungan (*arc*) dari dua kota dengan kombinasi dua *arc*, yaitu *arc* (i,j) dengan *arc* (i,k) dan *arc* (k,j), dengan k diambil dari kota yang belum masuk *subtour* dan dengan tambahan jarak terkecil.
   Jarak diperoleh dari: cik + ckj – cij
   cik adalah jarak dari kota i ke kota k,
   ckj adalah jarak dari kota k ke kota j dan
   cij adalah jarak dari kota i ke kota j
4. Ulangi langkah 3 sampai seluruh kota masuk dalam *subtour*.

Sebagai contoh diberikan 5 kota dengan jarak antar kota seperti tertera dalam Tabel 1.

Tabel 1. Jarak Antarkota

Kota Asal	*Kota Tujuan*	*Jarak*
1	2	132
1	3	217
1	4	164
1	5	58
2	3	290
2	4	201
2	5	79
3	4	113
3	5	303
4	5	196

Untuk mencari jarak terpendek melalui ke 5 kota tersebut sebagaimana terdapat dalam Tabel 1, ambil langkah-langkah sebagai berikut:

1. Ambil perjalanan dari kota 1 ke 5
2. Buat *subtour* → (1,5) → (5,1)
3. Buat tabel yang menyimpan kota yang bisa disisipkan dalam *subtour* beserta tambahan jaraknya, seperti ditampilkan dalam Tabel 2.

Tabel 2. *Arc* Penambah *Subtour* ke 1

Arc yang akan diganti	*Arc* yang ditambahkan ke *subtour*	Tambahan Jarak
(1,5)	(1,2) – (2,5)	$c_{12} + c_{25} – c_{15} = 153$
(1,5)	(1,3) – (3,5)	$c_{13} + c_{35} – c_{15} = 462$
(1,5)	(1,4) – (4,5)	$c_{14} + c_{45} – c_{15} = 302$
(5,1)	(5,2) – (2,1)	$c_{52} + c_{21} – c_{51} = 153$
(5,1)	(5,3) – (3,1)	$c_{53} + c_{31} – c_{51} = 462$
(5,1)	(5,4) – (4,1)	$c_{54} + c_{41} – c_{51} = 302$

Dari tabel 2 diberoleh tambahan jarak terkecil apabila:

Arc (1,5) diganti dengan *arc* (1,2) dan *arc* (2,5) atau *arc* (5,1) diganti dengan *arc* (5,2) dan *arc* (2,1) dari kemungkinan tersebut, bisa dipilih salah satu. Misal dipilih kemungkinan pertama maka *subtour* yang baru menjadi: → (1,2) → (2,5) → (5,1)

4. Selanjutnya dibuat tabel yang menyimpan kota yang bisa disisipkan dalam *subtour* beserta tambahan jaraknya, seperti ditampilkan dalam tabel 3.

Tabel 3. *Arc* Penambah *Subtour* ke 2

Arc yang akan diganti	*Arc* yang ditambahkan ke *subtour*	Tambahan Jarak
(1,2)	(1,3) – (3,2)	$c_{13} + c_{32} – c_{12} = 375$
(1,2)	(1,4) – (4,2)	$c_{14} + c_{42} – c_{12} = 233$
(2,5)	(2,3) – (3,5)	$C_{23} + c_{35} – c_{25} = 514$
(2,5)	(2,4) – (4,5)	$c_{24} + c_{45} – c_{25} = 318$
(5,1)	(5,3) – (3,1)	$c_{53} + c_{31} – c_{51} = 462$
(5,1)	(5,4) – (4,1)	$c_{54} + c_{41} – c_{51} = 302$

Dari tabel 3 diberoleh tambahan jarak terkecil adalah 233 dengan menggantikan *arc*(1,2) dengan *arc*(1,4) dan *arc*(4,2), sehingga *subtour* baru yang dihasilkan adalah: → (1,4) → (4,2) → (2,5) → (5,1)

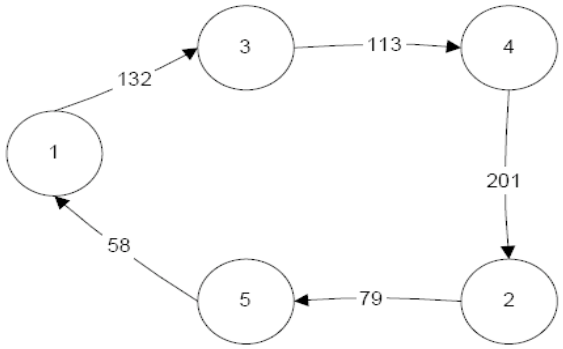
5. Karena masih ada kota yang belum masuk, perlu dibuat tabel yang menyimpan kota yang bisa disisipkan dalam *subtour* beserta tambahan jaraknya, seperti ditampilkan dalam Tabel 4.

Tabel 4. *Arc* Penambah *Subtour* ke 3

Arc yang akan diganti	*Arc* yang ditambahkan ke *subtour*	Tambahan Jarak
(1,4)	(1,3) – (3,4)	$c_{13} + c_{34} – c_{14} = 166$
(4,2)	(4,3) – (3,2)	$c_{43} + c_{32} – c_{42} = 202$
(2,5)	(2,3) – (3,5)	$C_{23} + c_{35} – c_{25} = 514$
(5,1)	(5,3) – (3,1)	$c_{53} + c_{31} – c_{51} = 462$

Dari tabel 4 diberoleh tambahan jarak terkecil adalah 166 dengan menggantikan *arc* (1,4) dengan *arc* (1,3) dan *arc* (3,4), sehingga *subtour* baru yang dihasilkan adalah: → (1,3) → (3,4) → (4,2) → (2,5) → (5,1)

Dari langkah-langkah tersebut diatas dapat diperoleh lintasan terpendek untuk mengunjungi 5 kota adalah → (1,3) → (3,4) → (4,2) → (2,5) → (5,1) seperti terlihat pada Gambar 2.



Gambar 2. Lintasan Terpendek antar 5 Kota

Dengan lintasan tersebut diperoleh jarak tempuhnya adalah:

c13 + c34 + c42 + c25 + c51 = 132 + 113 + 201 + 79 + 58 = 668

3. TEORI ANT COLONY SYSTEM (ACS)

Secara informal, ACS bekerja sebagai berikut [6]: pertama kali, sejumlah *m ants* ditempatkan pada sejumlah *n* kota berdasarkan beberapa aturan inisialisasi (misalnya, secara acak). Setiap semut membuat sebuah tur (yaitu, sebuah solusi TSP yang mungkin) dengan menerapkan sebuah aturan transisi status secara berulang kali. Selagi membangun turnya, seekor semut juga memodifikasi jumlah *pheromone* pada ruas-ruas yang dikunjunginya dengan menerapkan aturan pembaruan *pheromone* lokal yang telah disebutkan tadi. Setelah semua *ants* mengakhiri tur mereka, jumlah *pheromone* yang ada pada ruas-ruas dimodifikasi kembali (dengan menerapkan aturan pembaruan *pheromone* global).

3.1 Aturan Transisi Status pada ACS

Seekor semut yang ditempatkan pada kota *r* memilih untuk menuju ke kota *s* dengan menerapkan aturan yang ditunjukkan oleh persamaan (3)

$$S = \begin{cases} \arg\max_{u \in J_k(r)} \left\{ [\tau(r,u)] \cdot [\eta(r,u)]^{\beta} \right\} & \text{jika } q \le q_0 \\ & \text{(exploitation)} \\ S & \text{sebaliknya} \\ & \text{(biased exploration} \end{cases} \quad (3)$$

dimana *q* adalah sebuah bilangan pecahan acak antara 0 s.d. 1 [0 .. 1], *q*0 adalah sebuah parameter (0 ≤ *q*o), ≤ 1 dan *S* adalah sebuah variabel acak yang dipilih berdasarkan distribusi probabilitas yang telah diberikan pada persamaan (1) di atas.

Aturan transisi status yang dihasilkan dari persamaan (3) dan (1) dinamakan aturan *random-proportional* semu (*pseudorandom- proportional rule*). Aturan transisi status ini, sebagaimana dengan aturan *random-proportional* terdahulu, mengarahkan *ants* untuk bertransisi ke kota-kota yang dihubungkan dengan ruas-ruas yang pendek dan memiliki jumlah *pheromone* yang besar. Setiap kali seekor semut yang ada pada kota *r* harus memilih kota *s* sebagai tujuan berikutnya, ia membangkitkan sebuah bilangan acak antara 0 s.d. 1 (0 ≤ *q*o), ≤ 1). Jika *q*≤*q*o maka semut tersebut akan memanfaatkan pengetahuan yang ada mengenai masalah tersebut, yaitu pengetahuan heuristik tentang jarak antara kota tersebut dengan kota-kota lainnya dan juga pengetahuan yang telah didapat dan disimpan dalam bentuk *pheromone trail*. Hal ini mengakibatkan ruas terbaik (berdasarkan persamaan (3)) dipilih (*exploitation*). Jika sebaliknya maka sebuah ruas dipilih berdasarkan persamaan (1) (*biased exploration*).

3.2 Aturan Pembaruan *Pheromone* Global pada ACS

Pada sistem ini, pembaruan *pheromone* secara global hanya dilakukan oleh semut yang membuat tur terpendek sejak permulaan percobaan (ruas-ruas yang lain tidak diubah). Tingkat *pheromone* itu diperbarui dengan persamaan (4)

$$\tau(r,s) \leftarrow (1-\alpha) \cdot \tau(r,s) + \alpha \cdot \Delta\tau(r,s)$$

$$\text{dimana } \Delta\tau(r,s) = \begin{cases} \left(L_{gb}\right)^{-1} & \text{jika } (r,s) \in \text{global-best-tour} \\ 0 & \text{sebaliknya} \end{cases} \quad (4)$$

0<α<1 adalah parameter *pheromone decay*, dan *Lgb* adalah panjang dari tur terbaik secara global sejak permulaan percobaan. Seperti yang terjadi pada *ant system*, pembaruan *pheromone* global dimaksudkan untuk memberikan *pheromone* yang lebih banyak pada tur-tur yang lebih pendek. Persamaan (4) menjelaskan bahwa hanya ruas-ruas yang merupakan bagian dari tur terbaik secara global yang akan menerima penambahan *pheromone*.

3.3 Aturan Pembaruan *Pheromone* Lokal pada ACS

Selagi melakukan tur untuk mencari solusi dari TSP, *ants* mengunjungi ruas-ruas dan mengubah tingkat *pheromone* pada ruas-ruas tersebut dengan menerapkan aturan pembaruan *pheromone* lokal yang ditunjukkan oleh persamaan (5)

$$\tau(r,s) \leftarrow (1-\rho) \cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s) \quad (5)$$

dimana 0<ρ<1 adalah sebuah parameter. Peranan dari aturan pembaruan *pheromone* lokal ini adalah untuk mengacak arah tur-tur yang sedang dibangun, sehingga kota-kota yang telah dilewati sebelumnya oleh tur seekor semut mungkin akan dilewati kemudian oleh tur *ants* yang lain. Dengan kata lain, pengaruh dari pembaruan lokal ini adalah untuk membuat tingkat ketertarikan ruas-ruas yang ada berubah secara dinamis: setiap kali seekor semut menggunakan sebuah ruas maka ruas ini dengan segera akan berkurang tingkat ketertarikannya (karena ruas tersebut kehilangan sejumlah *pheromone*-nya), secara tidak langsung *ants* yang lain akan memilih ruas-ruas lain yang belum dikunjungi. Konsekuensinya, *ants* tidak akan memiliki kecenderungan untuk berkumpul pada jalur yang sama.

4. PEMBAHASAN

Berikut ini adalah tabel hasil percobaan masalah TSP yang telah diselesaikan dengan algoritma Cheapest Insertion Heuristic [3] dan algoritma Ant Colony System [4].

ISSN: `1907-5022

Tabel 5. Hasil Pengujian dengan CIH

Jumlah Kota	*Waktu proses (detik)*
5	1
10	3
15	7
20	17
25	33
30	56
35	87

Tabel 6. Hasil Pengujian dengan ACS

Banyak Kota	*Lama Proses (M sec)*
10	10
	10
	10
	10
	10
20	10
	10
	20
	10
	10
30	20
	31
	20
	20
	10
40	40
	41
	20
	40
	50
50	40
	91
	60
	40
	70

Dari hasil percobaan di atas, percobaan dengan algoritma Cheapest Insertion Heuristic (CIH) menggunakan jumlah kota dengan kelipatan 5, sedangkan percobaan Ant Colony System (ACS) dengan jumlah kota kelipatan 10. Kemudian dapat dilihat juga bahwa penyelesaian algoritma CIH membutuhkan waktu proses yang lebih singkat daripada menggunakan algoritma ACS, meskipun hasil yang didapatkan belum tentu optimal. Namun, hal ini hanya berlaku untuk jumlah kota kurang dari 20. Untuk jumlah kota yang lebih banyak dari 20, algoritma CIH mengalami peningkatan waktu proses yang cukup besar, sedangkan peningkatan waktu proses algoritma ACS relatif lebih stabil meskipun diuji dengan jumlah kota kelipatan 10.

Berdasarkan teori, tahapan algoritma CIH mengharuskan perhitungan tambahan jarak pada setiap iterasi subtour, yaitu menghitung semua kemungkinan tambahan jarak bila disisipkan sebuah vertex baru. Dengan kata lain, perhitungan dilakukan untuk mencari selisih jarak terkecil dari semua kemungkinan pergantian salah satu edge dalam subtour dengan dua buah edge pengganti (akibat dari penyisipan sebuah vertex). Misalnya, untuk tour T dan sebuah vertex k $\notin$ T dapat definisikan *insert*(T,k) untuk menjadi sirkuit dengan penghapusan sebuah edge (i,j) $\in$ T dan menggantinya dengan menyisipkan dua edge (i,k) dan (k,j).

Proses perhitungan dan perbandingan tambahan jarak pada CIH ini tentu saja dapat menghabiskan waktu proses yang singkat jika jumlah kota yang dihitung relatif sedikit, namun waktu proses akan semakin meningkat jika jumlah kota yang dihitung semakin banyak. Untuk 5 kota saja membutuhkan 16 perhitungan dengan 3 iterasi, 6 kota akan membutuhkan 30 perhitungan dengan 4 iterasi, 7 kota membutuhkan 50 perhitungan 5 iterasi, dan demikian seterusnya. Meskipun pemilihan subtour awal dibuat berbeda (dalam kasus TSP yang sama), maka akan dihasilkan rute perjalanan yang tetap sama. Oleh sebab itu, penyelesaian kasus TSP dengan menggunakan algoritma CIH menghabiskan waktu proses yang relatif lebih lama karena memungkinkan terjadinya perulangan perhitungan tambahan jarak (probabilitas edge yang akan digantikan) pada iterasi yang berbeda.

Sedangkan algoritma ACS akan membutuhkan waktu proses yang relatif cepat jika digunakan untuk jumlah kota yang banyak. Hal ini disebabkan karena salah satu langkah pada algoritma ACS mengkondisikan pemilihan titik selanjutnya berdasarkan aturan pembaruan *pheromone* global yang hanya dilakukan pada ruas-ruas yang merupakan bagian dari tur terbaik, sehingga dapat mengurangi jumlah kemungkinan yang berulang-ulang. Semut-semut akan 'dipandu' oleh informasi heuristik dari *pheromone*: Sebuah ruas dengan jumlah *pheromone* yang tinggi merupakan pilihan yang sangat diinginkan. Oleh sebab itu, algoritma ACS membutuhkan waktu yang lebih singkat untuk jumlah kota yang banyak karena tidak terjadi perulangan perhitungan probabilitas yang terus-menerus seperti halnya algoritma CIH.

Namun, untuk membuktikan perbandingan kehandalan kedua algoritma ini, maka dibutuhkan suatu implemetasi lebih lanjut agar dapat dianalisis perbedaannya dari segi waktu proses maupun hasil optimum dari sejumlah kota berbeda. Oleh sebab itu, sudi ini akan dijadikan sebagai uji literatur dari tema penelitian yang akan penulis angkat. Laporan hasil implementasi akan kami laporkan pada paper review selanjutnya.

ISSN: `1907-5022

5. KESIMPULAN

Berdasarkan analisa teori di atas, maka dapat diambil kesimpulan sebagai berikut:

- Algoritma Cheapest Insertion Heuristic dan Ant Colony System merupakan algoritma yang dapat digunakan untuk penyelesaian masalah TSP.
- Banyaknya jumlah kota menjadi salah satu faktor kecepatan waktu proses untuk kedua algoritma.
- Untuk jumlah titik kota yang sedikit (kurang dari 20 titik), algoritma CIH memiliki waktu proses yang lebih sedikit dibandingkan algoritma ACS, meskipun hasil yang didapatkan belum tentu optimal.
- Algoritma ACS lebih bagus diterapkan pada kasus TSP dengan jumlah kota yang banyak, karena waktu proses ACS relatif lebih cepat dibandingkan dengan algoritma CIH.

PUSTAKA

[1] Munir, Rinaldi. (2006). Diktat Kuliah IF2153 Matematika Diskrit Edisi Keempat. Departemen Teknik Informatika, Institut Teknologi Bandung .

[2] Lutfi, Ahmad. (2008). *Penyelesaian Traveling Salesman Problem Dengan Menggunakan Metode Cheapest Insertion Heuristic*. Diakses pada 21 Maret 2010 dari karya-ilmiah.um.ac.id/index.php/matematika/article/view/3642/1651.

[3] Kusrini, (2007). *Penyelesaian Travelling Salesman Problem Dengan Algoritma Cheapest Insertion Heuristics Dan Basis Data* Diakses pada 21 Maret 2010 dari dosen.amikom.ac.id/.../artikel/Kusrini-CIH-Gematika_Feb%2007_.pdf.

[4] Mario, T. *Implementasi Perbandingan Algoritma Ant Colony System Dengan Algoritma Subset Dynamic Programming Pada Kasus Travelling Salesman Problem* . Seminar Nasional Aplikasi Teknologi Informasi 2006 (SNATI 2006).

[5] Vitra, I. *Perbandingan metode-metode dalam algoritma genetika untuk Travelling Salesman Problem.* Proceedings Seminar Nasional Aplikasi Teknologi Informasi 2004.

ISBN: 979-756-061-6