BAB II
KAJIAN LITERATUR

2.1 Tinjauan Pustaka

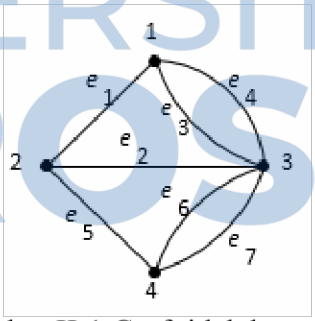
Pada subbab ini, akan dijelaskan tinjauan pustaka yang berkaitan dengan penelitan yang akan dilakukan.

2.1.1 Graph

Graph ialah pasangan himpunan (V,E) yang dituliskan menggunakan notasi G(V,E). Dimana V ialah himpunan yang tidak kosong (harus ada minimal 1 buah simpul) dari simpul – simpul *(node)* (Sam, 2016). Sedangkan E ialah himpunan sisi – sisi *(edges)* yang menghubungkan antar simpul dan boleh saja kosong (tidak ada) (Sam, 2016). Suatu graf dinyatakan kosong apabila E kosong sedangkan graf yang hanya memiliki 1 buah simpul dan tidak memiliki sisi dinamakan graf trivial (Marwan Sam, 2016). Berdasarkan jenisnya, graf terbagi menjadi 2 jenis, yaitu (Sam, 2016):

1. Graf tidak berarah *(undirected graph)*

   Sisi – sisi pada graf ini tidak memiliki orientasi arah, sehingga (vi, vj) = (vj, vi).

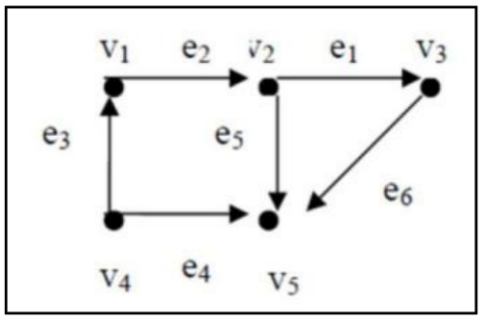


Gambar II-1 Graf tidak berarah

Pada gambar diatas, himpunan V = {1,2,3,4} adalah himpunan *node*. Sedangkan himpunan E = {e1,e2,e3,e4,e5,e6,e7} adalah himpunan *edges*.

2. Graf berarah *(directed graph)*

   Sisi – sisi pada graf ini memiliki orientasi arah, sehingga (u,v) ≠ (v,u).



Gambar II-2. Graf berarah

Ada beberapa istilah yang sering digunakan dalam graf, yaitu:

1. Bertetangga *(adjacent)*

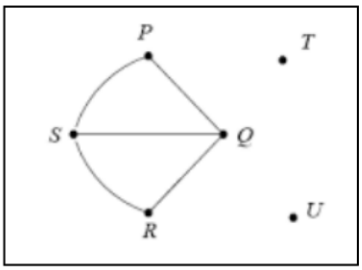
   Dua buah simpul dapat dikatakan bertetangga apabila kedua simpul tersebut saling terhubung di suatu sisinya.Pada Gambar II-1., *node* 1 dan *node* 2 dapat dikatakan bertetangga dikarenakan *node* 1 dan *node* 2 dihubungkan oleh *edges* e1, sedangkan *node* 1 dan *node* 4 tidak dapat dikatakan bertetangga dikarenakan tidak ada sisi dari kedua *node* tersebut yang berhubungan.

2. Bersisian *(incidency)*

   Suatu *edges* dapat dikatakan bersisian dengan *node* v1 dan v2 apabila e menghubungkan kedua *node* tersebut. Pada gambar II-1., e1 bersisian dengan *node* 1 dan 2 karena e1 menghubungkan kedua *node* tersebut namun tidak beririsan dengan *node* 4 dan 3.

3. *Node* terpencil *(isolated vertex)*

   Dikatakan *node* terpencil dikarenakan *node* tersebut tidak memiliki sisi yang beririsan dengannya.



Gambar II-3. *Node* terpencil

Pada gambar diatas, *node* T dan U merupakan *node* terpencil dikarenakan tidak ada *edges* yang beririsan.

4. Derajat *(degree)*

Jumlah sisi yang beririsan dengan *node* dinyatakan dalam derajat suatu *node*. Pada gambar II-1., derajat untuk tiap simpul yaitu: d(1) = 3, d(2) = 3, d(3) = 5, d(4) = 3.

5. Lintasan *(path)*

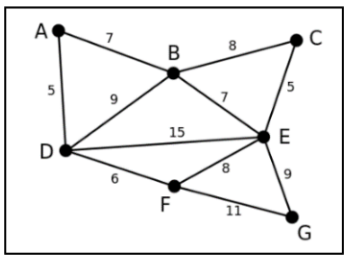
Lintasan yang panjangnya n dari *node* awal v0 ke *node* tujuan vn dalam sebuah *graph* ialah barisan berselang – seling *node* – *node* dan *edges* yang berbentuk v0,v1,v2,v3, …, vn. Contoh lintasan terdapat pada gambar II-1. dimana lintasannya adalah 1,2,3,1,3,4.

6. Siklus *(cycle)*

Pada graf ini, lintasan pada *node* awal akan berakhir pada *node* yang sama. Pada gambar II-1., siklus dapat terlihat pada: 2,,3,4,2.

7. Graf berbobot *(weighted graph)*

Pada graf ini, setiap sisi diberi nilai atau bobot.

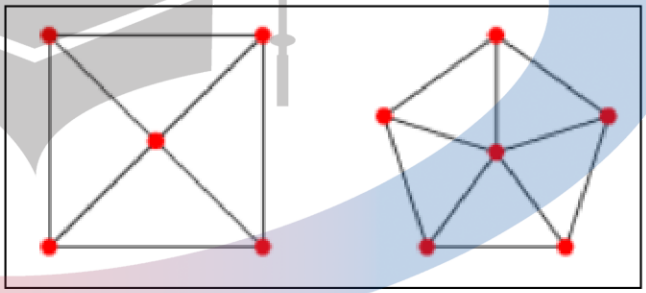


Gambar II-4. Graf berbobot

Bobot pada setiap sisi graf dapat berbeda – beda tergantung pada masalah yang dimodelkan. Bobot nilai pada graf dapat menyatakan jarak setiap *node*, biaya perjalanan waktu tempuh, ongkos produksi dan lain sebagainya.

8. Keterhubungan

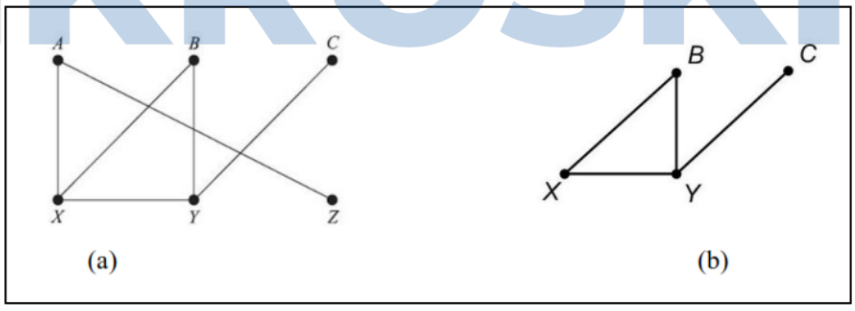
Setiap *node* akan dinyatakan terhubung apabila terdapat lintasan pada *node* tersebut. Jika setiap pasang *node* yang terdapat lintasan, maka graf tersebut dinamakan *connected graph*. Sebaliknya, jika tidak terdapat lintasan pada setiap pasang *node* tersebut, maka graf tersebut dinamakan *disconnected graph*.



Gambar II-5. *Connected Graph*

9. Subgraf

Misalkan G = (V,E) adalah sebuah graf. G1 = (V1,E1) dinamakan subgraf dari G jika V1 ≤ V dan E1 ≥ E.



Gambar II-6. Graf (b) merupakan subgraf dari graf (a)

2.1.2. Metode Pencarian

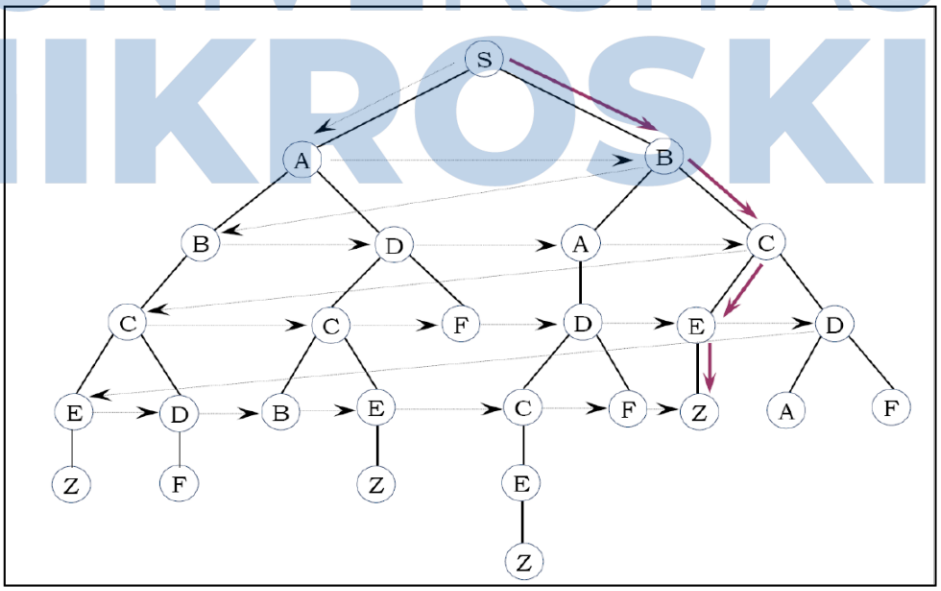
Ada banyak metode yang dapat digunakan untuk pencarian jalur terdekat pada suatu graf. Metode pencarian tersebut dapat dikelompokkan ke dalam dua jenis, yaitu pencarian buta/tanpa informasi (*blind* atau *un-informed search*) dan pencarian heuristik/dengan informasi (*heuristic* atau *informed search*).

2.1.2.1. Pencarian Buta (*Blind Search/Un-informed Search*)

Dikatakan pencarian buta, karena pada pencarian ini tidak ada informasi awal. Disini hanya akan dibahas dua metode pencarian, yaitu *Breadth First Search* dan *Depth First Search*.

2.1.2.2. *Breadth First Search* (*BFS*)

Pencarian dilakukan pada semua verteks pada level n secara berurutan dari kiri ke kanan. Jika pada satu level belum ditemukan solusi, maka pencarian dilanjutkan pada level berikutnya *(n+1)*. Demikian seterusnya sampai ditemukan solusi. Dengan strategi ini, maka dapat dijamin bahwa solusi yang ditemukan adalah yang paling baik (*Optimal*). Tetapi BFS harus menyimpan semua node yang pernah dibangkitkan,hal ini harus dilakukan untuk penelusuran balik jika solusi sudah ditemukan, sehingga membutuhkan memori yang cukup banyak.

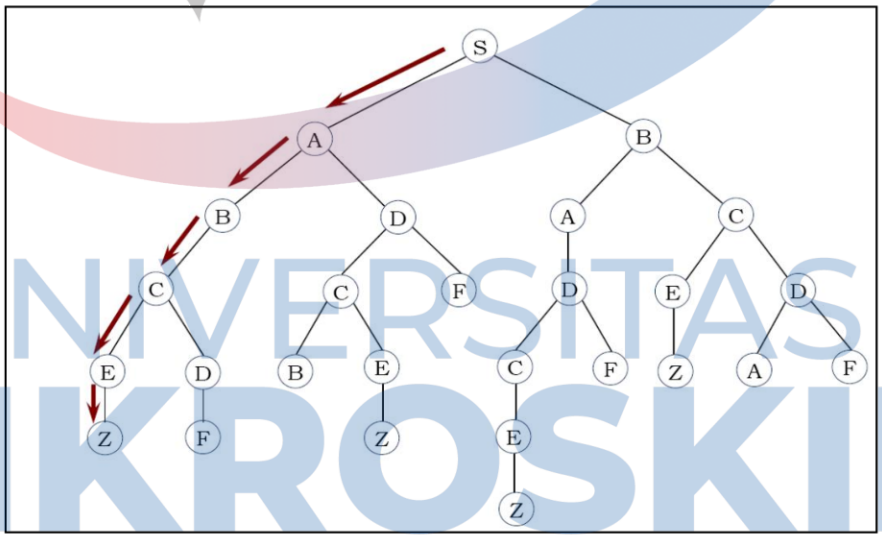


Gambar II-7. Tree untuk *Breadth First Search*

2.1.2.3. *Depth First Search*(DFS)

Pencarian dilakukan pada satu verteks dalam setiap level dari yang paling kiri. Jika pada level yang paling dalam, solusi belum ditemukan, maka pencarian dilanjutkan pada verteks sebelah kanan. Verteks yang kiri dapat dihapus dari memori. Jika pada level yang paling dalam tidak ditemukan solusi, maka pencarian dilanjutkan pada level sebelumnya. Demikian seterusnya sampai ditemukan solusi. Jika solusi ditemukan maka tidak diperlukan proses *backtracking* (penelusuran balik untuk mendapatkan jalur yang dinginkan).

Kelebihan dari algoritma ini adalah pemakaian memori yang lebih sedikit, sedangkan kelemahannya adalah jika pohon yang dibangkitkan memiliki level yang sangat dalam (tak terhingga), maka tidak ada jaminan menemukan solusi. Artinya, DFS tidak *complete* (tidak ada jaminan penemuan solusi).



Gambar II-8 Tree untuk *Depth First Search*

2.1.2.4. Pendekatan Heuristik dan Metaheuristik

Heuristik

Pada metode pencarian buta, tidak dimiliki pengetahuan khusus tentang permasalah yang dihadapi sehingga metode tersebut tidak efisien untuk banyak kasus karena bisa saja metode tersebut tidak *complete* dan atau tidak optimal

dalam mendapatkan solusi, optimal disini adalah tidak menjamin menemukan solusi yang terbaik jika terdapat beberapa solusi yang berbeda. Menggunakan informasi khusus yang spesifik untuk suatu masalah tertentu akan sangat memperbaiki kecepatan pencarian solusi, karena teknik ini membantu memutuskan kemungkinan solusi mana yang pertama kali perlu di evaluasi. Pencarian heuristik digunakan untuk mengeliminasi beberapa kemungkinan solusi, tanpa harus mengeksplorasinya secara penuh.

Berikut akan dijelaskan beberapa algoritma pencarian dengan informasi*(informed search algorithm*) yang menggunakan fungsi heuristik dalam mencari solusi, yaitu *Generate and test*, *hill climbing,* dan *Best First Search (greedy best first search dan A*).*

Metaheuristik

ACO adalah sebuah metode metaheuristik yang terinspirasi dari kecerdasan semut dalam pencarian jalur terpendek menuju sumber makanan. Semua algoritma metaheuristik memiliki dua komponen inti: intensifikasi (pencarian lokal) dan diversifikasi (pencarian global). Intensifikasi mengeksplorasi daerah tetangga yang menjanjikan dengan harapan menemukan solusi yang lebih baik. Di sisi lain, diversifikasi memastikan bahwa semua wilayah ruang pencarian telah dikunjungi, yang memungkinkan algoritma untuk melompat keluar dari optimum lokal. Menyeimbangkan interaksi antara dua komponen dapat secara signifikan mempengaruhi efisiensi algoritma metaheuristik kinerja algoritma metaheuristik sangat tergantung pada keseimbangan yang baik antara kedua komponen ini.

2.1.2.5. *Generate and Test* (bangkitkan dan Uji)

Metode *Generate*-and-*Test* adalah metode yang paling sederhana dalam pencarian *heuristic*. Jika pembangkitan *possible solution* dikerjakan secara sistematis, maka algoritma ini akan mencari solusinya, jika ada. Tetapi jika ruang masalahnya sangat luas, mungkin memerlukan waktu yang sangat lama. Algoritma *Generate*-and-*Test* menggunakan prosedur DFS karena solusi harus dibangkitkan secara lengkap sebelum dilakukan *test*. Algoritma ini berbentuk

sistematis, pencarian sederhana yang mendalam dari ruang permasalahan. *Generate &test* juga dapat dilakukan dengan pembangkitan solusi secara acak, tetapi tidak ada jaminan solusinya akan ditemukan.

2.1.2.6. *Hill Climbing* (Pendakian Bukit)

Hill Climbing berbeda *Generate*-and-*Test*, yaitu pada *feedback* dari prosedur *test* untuk membantu pembangkit menentukan yang langsung dipindahkan dalam ruang pencarian. Dalam prosedur *Generate &test* , respon fungsi pengujian hanya yaatau tidak. Tapi jika pengujian ditambahkan dengan atauran fungsi-fungsi yang menyediakan estimasi dari bagaimana mendekati *state* yang diberikan ke *state* tujuan, prosedur pembangkit dapat mengeksplorasi ini sebagaimana ditunjukkan di bawah. *Hill Climbing* sering digunakan jika terdapat fungsi heuristik yang baik untuk mengevaluasi *state*. Sebagai contoh, anda berada di sebuah kota yang tidak dikenal, tanpa peta dan anda ingin menuju ke pusat kota. Cara sederhana adalah gedung yang tinggi. Fungsi heuristik-nya adalah jarak antara lokasi sekarang dengan gedung yang tinggi dan *state* yang diperlukan adalah jarak yang terdekat.

2.1.2.7. *Best First Search*(BFS)

Best first search merupakan kombinasi dari beberapa kelebihan *Depth first search* dan *breadth first search*. Pada pencarian dengan *hill climbing* tidak diperbolehkan untuk kembali ke verteks pada level yang lebih rendah meskipun verteks pada level yang lebih rendah tersebut memiliki nilai heuristik yang lebih baik, sedangkan pada *best first search*, pencarian diperbolehkan untuk mengunjungi verteks yang berada pada level yang lebih rendah.

Best First Search membangkitkan verteks berikutnya dari sebuah verteks (yang sejauh ini terbaik diantara semua *leafnodes* yang pernah dibangkitkan. Untuk menentuan verteks terbaik dapat dilakukan dengan menggunakan informasi berupa biaya perkiraan dari suatu verteks menuju ke *goal* atau gabungan antara

biaya sebenarnya dan biaya perkiraan tersebut. Biaya perkiraan tersebut dapat diperoleh dengan menggunakan suatu fungsi yang disebut fungsi heuristik. Terdapat dua jenis algoritma *best first search*, yaitu: 1) algoritma *greedy best first search*,yang hanya memperhitungkan biaya perkiraan saja; dan 2) algoritma A*, yang menghitung gabungan biaya antara biaya sebenarnya (*actual cost*) dan biaya perkiraan.

2.1.2.8. *Greedy Best First Seach*

*Greedy Best First Seach*merupakan pencarian *best first search* dengan hanya mempertimbangkan harga perkiraan (*estimated cost*) saja, yaitu f(n) = h(n). Sedangkan harga sesungguhnya tidak digunakan. Sehingga solusi yang dihasilkan tidak optimal, karena hanya memperhitungkan biaya perkiraan yang belum tentu kebenarannya.

2.1.3. Lintasan Terpendek (*Shortest Path*)

Lintasan terpendek (*Shortest Path*) merupakan lintasan minimum yang diperlukan untuk mencapai suatu titik dari titik tertentu. Dalam pencarian lintasan terpendek, masalah yang dihadapi adalah mancari lintasan mana yang akan dilalui, sehingga didapat lintasan yang paling pendek dari satu verteks ke verteks yang lain.

Ada beberapa macam persoalan lintasan terpendek, antara lain:

1. Lintasan terpendek antara dua buah verteks.
2. Lintasan terpendek antara semua pasangan verteks.
3. Lintasan terpendek dari verteks tertentu ke semua verteks yang lain
4. Lintasan terpendek antara dua buah verteks yang melalui beberapa verteks tertentu.

Pada persoalan lintasan terpendek, yang menjadi masalah adalah lintasan terpendek antara dua buah verteks, dimana bobot pada setiap *edge graph* digunakan untuk menyatakan jarak antar kota dalam satuan kilometer (km).

2.1.4. Algoritma Pencarian Rute

Algoritma adalah kumpulan instruksi/perintah yang dibuat secara jelas dan sistematis berdasarkan urutan yang logis (logika) untuk penyelesaian suatu masalah. Namun Algoritma pencarian rute tujuannya adalah algoritma yang menentukan bagaimana memilih rute optimal antara awal dan tujuan dengan memperhitungkan waktu kalkulasi terpendek. Untuk itu, beberapa algoritma yang sebelumnya sudah dikembangkan, antara lain Algoritma Dijkstra, Moore, Ant Calony, Tabu Search, A* dan lain sebagainya.

2.1.5. *Travelling Salesman Problem (TSP)*

Traveling Salesman Problem(TSP) merupakan sebuah permasalahan optimasi yang dapat diterapkan pada berbagai kegiatan seperti *routing*. Masalah optimasi *TSP* terkenal dan telah menjadi standar untuk mencoba algoritma yang komputational. Pokok permasalahan dari *TSP* adalah seorang salesman harus mengunjungi sejumlah kota yang diketahui jaraknya satu dengan yang lainnya dan kembali ke kota asal (Sitanggang.2015).

Traveling salesman problem dapat didefenisikan sebagai berikut : ada satu set kota {C1, C2, C3,...Cn} dan d {Ci, Cj} adalah jarak antar kota ke-i dan ke-j. Tujuannya adalah menemukan urutan π dari rumus berikut untuk mendapatkan nilai yang paling minimal (Delima Sitanggang,2015).

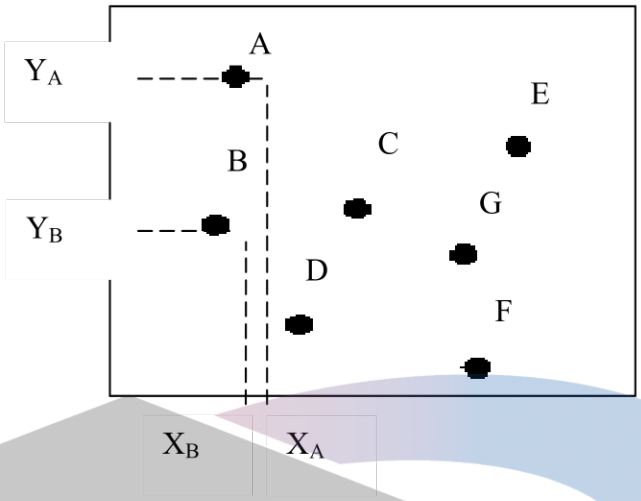
$$\sum d \left(C_{\pi(i)}, C_{\pi(i)}\right) + d \left(C_{\pi(N)}, C_{\pi(1)}\right) \quad (2.1)$$

Hasil dari rumus tersebut disebut sebagai panjang dari perjalanan seorang *salesman* mengunjungi kota-kota sesuai urutan π dimana setelah mengujungi semua kota, *salesman* tersebut kembali lagi ke kota asalnya.

Permasalahan *Traveling Salesman Problem* ada dua macam, yaitu *Symetric Salesman Problem* yang biasa disebut dengan *Traveling Salesman Problem* dan *Asymetric Salesman Problem*. Pada *Symetric Salesman Problem*, d {Ci, Cj}=d {Cj, Ci} sedangkan jika *Asymetric Salesman Problem* maka d {Ci, Cj}≠d {Cj, Ci} untuk 1≤i dan j≥n.

Traveling Salesman Problem adalah salah satu masalah optimasi yang merupakan *NP-complete problem* (*non deterministic polynomial complete problem*), yaitu suatu permasalahan dimana dengan bertambahnya variabel secara linier maka waktu yang diperlukan untuk memecahkan masalah itu bertambah secara eksponensial. Hal ini sering juga disebut dengan *combinatorial exponential*.

TSP secara sederhana dapat didefinisikan sebagai proses pencarian lintasan terefisien dan terpendek dari beberapa kota yang dipresentasikan, melewati setiap kota tersebut dan kembali ke kota awal. Setiap kota hanya dapat dikunjungi sebayak satu kali saja. Persoalan yang dihadapi *TSP* ialah bagaimana merencanakan total jarak yang minimum. Untuk menyelesaikan persoalan tersebut, tidak mudah dilakukan karena terdapat ruang pencarian dari sekumpulan permutasi sejumlah kota, maka *TSP* kemudian dikenal dengan persoalan *Non Polynomial*. Gambaran sederhana dari pengertian *TSP* dapat diperlihatkan pada Gambar II-7.



Gambar II-9. Posisi kota-kota yang akan dilewati

Kota-kota pada Gambar II-7 masing-masing mempunyai koordinat (x,y), sehingga jarak antar kedua kota dapat dihitung dengan rumus (2.2).

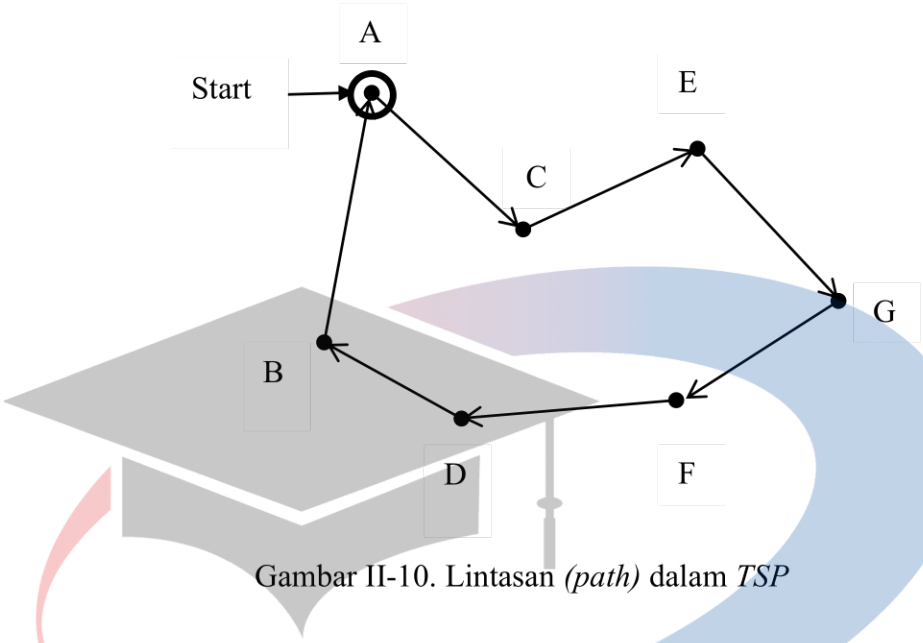
$$d_{A,B} = \sqrt{\left(x_A - x_B\right)^2 + \left(y_A - y_B\right)^2} \quad (2.2)$$

Keterangan :

$d_{A,B}$ = Jarak kota A dan kota B

x,y = Koordinat titik (Kota)

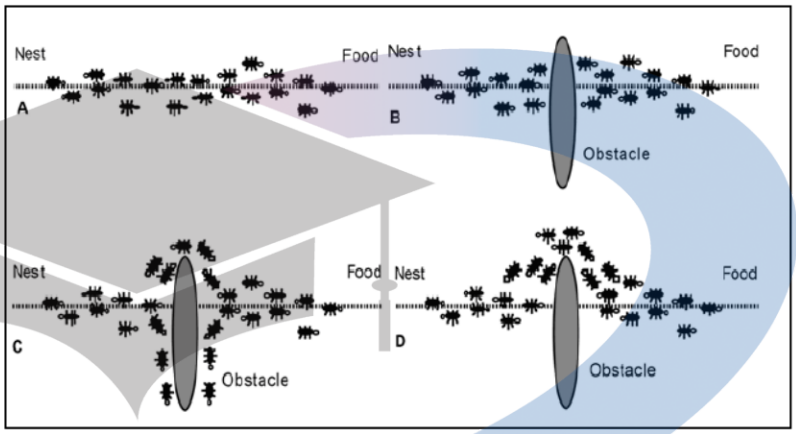
Setelah jarak-jarak yang menghubungkan tiap kota diketahui, maka dicari rute terpendek dari jalur yang akan dilewati untuk kembali ke kota awal diperlihatkan pada Gambar II-10.



Gambar II-10. Lintasan *(path)* dalam *TSP*

2.1.6. *Ant Colony Optimization (ACO)*

Algoritma *ACO* diperkenalkan oleh Marco Dorigo (Ping Duan, 2016). *ACO* menerapkan metode penyelesaian masalah optimasi berdasarkan prinsip komunikasi antar koloni semut. Berdasarkan hasil akhirnya ACO dapat dikatagorikan kedalam optimasi *local optimum* karena nilai optimal yang dapat dicapai oleh sebuah berada dalam rentang nilai tertentu yang telah dibatasi sedangkan *global optimum* mencari nilai optimal dari keseluruhan input. Pada dasarnya, semua semut akan meninggalkan jejak berupa zat khusus yang dikenal dengan istilah feromon. Feromon inilah yang kemudian akan menjadi pedoman bagi semut – semut yang lain dalam melakukan pencarian. Jika semakin pendek jalur yang dikunjungi, maka semakin sedikit pula penguapan yang terjadi dan semakin tinggi pula feromon yang ada pada jalur atau lintasan tersebut. Semut – semut akan cenderung bergerak mengikuti lintasan yang memiliki jejak feromon yang tinggi (Sitanggang, 2018), Sehingga ACO merupakan salah satu algoritma yang dapat digunakan dalam menyelesaikan permasalahan TSP.



Gambar II-11. Perjalanan semut dalam menemukan sumber makanan

Langkah – langkah algoritma *ACO* dalam menentukan jalur terpendek, adalah: (Zarman, 2016)

1. Inisialisasi parameter yang digunakan dalam algoritma *ACO*:
    1. Intensitas jejak semut antar kota dan perubahannya (τ_{ij})
    2. Banyak kota (n) termasuk x dan y (koordinat) atau dij (jarak antar kota)
    3. Penentuan kota berangkat dan kota tujuan
    4. Tetapan siklus semut (Q)
    5. Tetapan pengendali intensitas jejak semut (α)
    6. Tetapan pengendali visibilitas (β)
    7. Visibilitas antar kota =1/dij (η_{ij})
    8. Jumlah semut (m)
    9. Tetapan penguapan jejak semut (ρ)
    10. Jumlah siklus maksimum (NCmax) bersifat tetap selama algoritma dijalankan, sedangkan τij akan selalu diperbaharui harganya pada setiap

siklus algoritma mulai dari siklus pertama (NC=1) sampai tercapai jumlah siklus maksimum (NC=NCmax) atau sampai terjadi konvergensi.

2. Inisialisasi kota pertama pada setiap semut. Setelah inisialisasi τij dilakukan, kemudian m semut ditempatkan pada kota pertama yang telah ditentukan.
3. Isi kota pertama ke dalam *tabu list*. Hasil inisialisasi kota pertama semut pada langkah diatas, harus diisikan sebagai elemen pertama pada *tabu list*. Hasil dari langkah ini ialah terisinya elemen pertama *tabu list* setiap semut dengan indeks kota pertama.
4. Penyusunan jalur kunjungan setiap semut ke setiap kota. Koloni semut yang sudah terdistribusi ke kota awal akan mulai melakukan perjalanan dari kota pertama sebagai kota awal dan salah satu kota – kota lainnya sebagai kota tujuan. Kemudian dari kota kedua, masing – masing koloni semut akan melanjutkan perjalanan dengan memilih salah satu dari kota – kota yang tidak terdapat pada $tabu_k$ sebagai kota tujuan selanjutnya. Perjalanan koloni semut akan berlangsung secara terus – menerus hingga mencapai kota tujuan. Jika s menyatakan indeks urutan kunjungan, maka kota asal dinyatakan sebagai $tabu_k(s)$ dan kota - kota yang lain akan dinyatakan dalam {N-$tabu_k$}, maka untuk menentukan kota tujuan, digunakan persamaan probabilitas kota untuk dikunjungi sebagai berikut:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{k' \in \{N - tabu_k\}} [\tau_{ik'}]^{\alpha} \cdot [\eta_{ik'}]^{\beta}} \text{ untuk } j \in \{N\text{-}tabu_k\}$$

$$p_{ij}^{k} = 0, \text{ untuk j lainnya} \quad (2.3)$$

Dengan i sebagai indeks kota asal dan j sebagai indeks kota tujuan.

5. Menghitung panjang jalur setiap semut. Menghitung panjang jalur tertutup *(length closed tour)* atau L_k setiap semut dilakukan setelah satu siklus diselesaikan oleh semua semut. Perhitungan dilakukan berdasarkan $tabu_k$ masing – masing dengan persamaan berikut:

$$L_k = d_{tabu_k(n),tabu_k(1)} + \sum_{s=1}^{n-1} d_{tabu_k(s),tabu_k(s+1)} \quad (2.4)$$

Dengan d_{ij} adalah jarak antar kota i ke kota j yang dihitung berdasarkan persamaan:

$$d_{i,j} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \quad (2.5)$$

6. Pencarian jalur terpendek. Setelah L_k setiap semut dihitung, akan diperoleh nilai minimal panjang jalur tertutup setiap siklus atau L_{minNC} dan nilai minimal panjang jalur tertutup secara keseluruhan adalah L_{min}.
7. Perhitungan perubahan harga intensitas jejak kaki semut antar kota. Koloni semut akan meninggalkan jejak – jejak kaki pada lintasan antar kota yang dilaluinya. Adanya penguapan dan perbedaan jumlah semut yang lewat, menyebabkan kemungkinan terjadinya perubahan harga intensitas jejak kaki semut antar kota. Persamaan perubahannya adalah:

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \quad (2.6)$$

Dengan $\Delta\tau_{ij}^{k}$ adalah perubahan harga intensitas jejak kaki semut antar kota setiap semut yang dihitung berdasarkan persamaan

$$\Delta\tau_{ij}^{k} = \frac{Q}{L_k} \quad (2.7)$$

untuk (i,j) ∈ kota asal dan kota tujuan dalamtabu$_k$.

$$\Delta\tau_{ij}^{k} = 0 \text{ , untuk (i,j) lainnya} \quad (2.8)$$

8. Perhitungan harga intensitas jejak kaki semutantar kota untuk siklus selanjutnya.Harga intensitas jejak kaki semut antar kotapada semua lintasan antar kota adakemungkinan berubah karena adanyapenguapan dan perbedaan jumlah semut yangmelewati. Untuk siklus selanjutnya, semut yangakan melewati lintasan tersebut hargaintensitasnya telah berubah. Harga intensitasjejak kaki semut antar kota untuk siklusselanjutnya dihitung dengan persamaan:

$$\tau_{i\ j} = \rho\ .\tau_{i\ j} + \Delta\tau_{i\ j} \quad (2.9)$$

9. Atur ulang harga perubahan intensitas jejak kakisemut antar kota.Untuk siklus selanjutnya perubahan hargaintensitas jejak semut antar kota perlu diatur kembali agar memiliki nilai sama dengan nol.
10. Pengosongan tabu list, dan ulangi langkahdua jika diperlukan. Tabu list perlu dikosongkanuntuk diisi lagi dengan urutan kota yang baru padasiklus selanjutnya, jika jumlah siklus maksimumbelum tercapai atau belum terjadi konvergensi.Algoritma diulang lagi dari langkah dua denganharga parameter intensitas jejak kaki semut antarkota yang sudah diperbaharui.

2.1.7. Algoritma *Tabu Search*

Algoritma *Tabu Search* merupakan salah satu algoritma yang berada dalam ruang lingkup metode heuristik. Algoritma ini menggunakan *short-term memory* untuk menjaga agar proses pencarian tidak terjebak pada nilai optimum lokal. Algoritma ini menggunakan *tabu list* untuk menyimpan sekumpulan solusi yang baru saja dievaluasi. Selama proses optimasi pada setiap iterasi, solusi yang akan dievaluasi akan dicocokkan terlebih dahulu dengan isi *tabu list* untuk melihat apakah solusi tersebut sudah ada pada *tabu list*. Apabila solusi tersebut sudah ada pada *tabu list*,maka solusi tersebut tidak akan dievaluasi lagi padaiterasi

berikutnya. Dan jika sudah tidak ada lagi solusi yang tidak akan menjadi anggota *tabu list*, maka nilai terbaik yang baru saja diperoleh merupakan solusi yang sebenarnya. Dua komponen yang sangat penting dalamalgoritma *Tabu Search* adalah strategi intesifikasi dan strategi diversifikasi (Hasugian, 2017).

1. Strategi Intensifikasi

   Strategi ini berdasarkan pada modifikasi aturan pemilihan untuk mendorong atau menguatkan kombinasi pergerakan dan solusi yang mempunyai histori yang baik. Strategi ini juga dapat memberikan suatu nilai kepada daerah potensial untuk diproses lebih mendalam. Strategi ini mencari *"neighbors"*dengan menggabungkan semua komponen dari solusi yang baik atau dengan menggunakan evaluasi dari strategi yang telah dimodifikasi menjadi sebuah solusi yang dapat berkembang.

2. Strategi Diversifikasi

   Strategi ini didasarkan atas suatu proses pencarian yang digunakan untuk menguji daerah yang tidak pernah dikunjungi atau dibahas sebelumnya untuk menghasilkan suatu solusi yang berbeda dari alternatif-alternatif solusi yang pernah ada atau dapatjuga dikatakan bahwa pengukuran diversifikasi berhubungan dengan banyaknya perpindahan yang dibutuhkan untuk memindahkan satu solusi ke solusi yang lain. Strategi ini mendorong proses untuk pencarian untuk mencoba daerah yang belum pernah dikunjungi dan untuk menghasilkan solusi yang berbeda dalam banyak hal dengan solusi yang pernah diketahui sebelumnya.

Tabu Search dapat diaplikasikan langsung ke *statement* verbal maupun simbolik dari berbagai macam masalah pengambilan keputusan tanpa perlu untuk mengubahnya menjadi bentuk rumus matematisnya. Meskipun begitu, ada gunanya juga untuk menggunakan notasi matematika untuk menggambarkan lingkup yang lebih besar dari masalah sebagai dasar untuk menjelaskan beberapa hal dalam *Tabu Search*. *Tabu Search* mengkarateristikkan bagian dari masalah dengantujuan megoptimalkan dari fungsi f(x) dengan x € X,dimana f(x) dapat berupa linear maupun non-linear dan X ringkasan yang mengandung nilai

keputusan x. *Tabu Search* mulai dengan cara yang sama seperti *ordinary local* atau *neighborhood search*, memulai proses iterasi dari satu titik (solusi) ke solusi lain sampai kriteria atau syarat berhenti tercapai. Setiap x € X berasosiasi dengan *neighborhood* N(x) € X dan setiap solusi x € N(x) diperoleh dari x dari operasi yang dinamakan *move*. Beberapa elemen utama pada *Tabu Search* adalah (Hasugian, 2017):

1. Representasi solusi: setiap solusi yang mungkin pada suatu permasalahan optimasi harus direpresentasikan.
2. Fungsi *cost*: setiap fungsi *cost* akan memetakan setiap solusi yang mungkin ke nilai *cost*-nya.
3. *Neighborhood* (tetangga): suatu fungsi yang memetakan setiap solusi yang mungkin ke solusi – solusi lainnya.
4. *Tabu List* (memori jangka pendek): yaitu memori untuk menyimpan jumlah solusi yang terbatas yang memungkinkan terjadinya perulangan.
5. *Aspiration criteria*: yaitu elemen untuk mencegah proses pencarian mengalami stagnasi (terhambat) karena adanya proses pengujian yang disertai *tabu move*.
6. *Long Term Memory* (memori jangka panjang): yaitu elemen untuk menyimpan atribut solusi yang akan digunakan dalam *intensification* (untuk memperioritaskan pada atribut dari satu set solusi) dan *diversification* (untuk memperkecil atribut solusi ketika dipilih untuk memperluas pencarian solusi).

Secara umum *Tabu Search* memiliki algoritma, yaitu (Hasugian, 2017):

1. Pilih solusi i yang mungkin dalam S. Set i^* = I dan k = 0.
2. Tetapkan k = k+1 dan hasilkan himpunan bagian V^* dari solusi dalam himpunan Solusi N(i,k).
3. Pilih solusi terbaik j dalam himpunan V^*. Tetapkan i=j.
4. Jika $f(i) \leq f(i^*)$ makan tetapkan $i^* = i$.
5. Jika kondisi berhenti terpenuhi maka pencarian berhenti. Jika tidak, Lakukan kembali langkah 2.

Algoritma dasar pada *Tabu Search* dapat dijelaskan sebagai berikut (Paska Marto Hasugian, 2017):

1. Pilih solusi awal i dalam himpunan S. Tetapkan $i^* = i$ dan $k = 0$ dimana i^* adalah solusi terbaik dan k adalah banyaknya perulangan yang terjadi saat dilakukannya pencarian solusi terbaik i^*.
2. Tetapkan $k = k + 1$ dan hasilkan himpunan bagian V^* dari solusi dalam solusi himpunan N(i,k) sehingga *tabu conditions* tidak memenuhi dan *aspirationsconditions* terpenuhi.
3. Pilih solusi terbaik jdalam himpunan V^*. Tetapkan i = j.
4. Jika $f(i) \leq f(i^*)$ maka tetapkan $i^* = i$.
5. *Updatetabu* dan *aspirationsconditions*.
6. Jika kondisi berhenti (*stopping conditions)* terpenuhi, maka pencarian berhenti. Jika tidak, lakukan langkah 2.

Kondisi berhenti (*stopping conditions*) akan terpenuhi jika(Paska Marto Hasugian, 2017):

1. N(i,k+1) = Ø atau jika tidak ada solusi yang mungkin disekitar solusi i.
2. Nilai k lebih besar dari batas maksimum perulangan yang diijinkan.
3. Banyaknya perulangan yang terjadi dari mulai perbaikan solusi i adalah besar dari jumlah yang ditetapkan.

Dimana:

i*= Solusi terbaik dari solusi yang ditemukan

k= Perulangan

j= Solusi yang ditemukan untuk perulanganberikutnya

S= Himpunan solusi yang mungkin

V*= Himpunan bagian dari N(i,k).

N(i,k)= Himpunan solusi yang mungkin untuk semuaperulangan

2.1.8 Algoritma SMARTER

Metode SMARTER (*Simple Multi-Attribute Rating Technique Exploiting Ranks*) Merupakan metode pengambilan keputusan multi kriteria yang diusulkan oleh Edwards dan Baron pada tahun 1994. Teknik pengambilan keputusan multi

kriteria ini didasarkan pada teori bahwa setiap alternatif terdiri dari sejumlah kriteria yang memiliki nilai-nilai dan setiap kriteria memiliki bobot yang menggambarkan seberapa penting ia dibandingkan dengan kriteria lain. Pembobotan pada metode SMARTER menggunakan range antara 0 sampai 1, sehingga mempermudah perhitungan dan perbandingan nilai pada masing-masing alternatif (Haryanti, D., Nasution, H. & Sukamto, A. S. 2016).

Pada metode SMARTER, bobot dihitung dengan menggunakan rumus pembobotan Rank-Order Centroid (ROC) (Roberts, R. and Goodwin, P. 2002), (Baker, D., Bridges, D., Hunter, R., Johnson, G., Krupa, J., Murphy, J. and Sorenson, K. 2002), (Jayanath Ananda and Gamini Herath 2009). ROC ini didasarkan pada tingkat kepentingan atau prioritas dari kriteria. Pembobotan ROC didapat dengan prosedur matematika sederhana dari prioritas. Ide dasarnya dapat diilustrasikan dengan 2 atribut, A dan B. Jika A ranking pertama, maka bobotnya harus berada diantara 0,5 dan 1 sehingga titik tengah interval 0,75 diambil sebagai bobot perkiraan, yang merupakan dasar dari sebuah prinsip komitmen minimum. Seperti bobot B akan menjadi 0,25 (merupakan titik tengah antara 0 dan 0,5) Prosedur ini dapat dirumuskan sebagai berikut (jika ada K kriteria) :

(2.10)

Secara umum, jika K adalah jumlah kriteria, maka bobot dari kriteria ke K adalah :

$$Wk = \frac{1}{K}\sum_{i=k}^{K}\frac{1}{i} \qquad (2.11)$$

Keterangan:

W = Nilai pembobotan kriteria,

K = Jumlah kriteria

i = Nilai alternatif

Selanjutnya adalah perhitungan nilai Utility rumus yang digunakan adalah:

$$v(x) = \sum_{i=1}^{n} wi\ vi\ (x) \quad (2.12)$$

Keterangan:

Wi = Bobot yang mempengaruhi dari dimensi ke i terhadap nilai keseluruhan evaluasi.

Vi = Objek evaluasi pada dimensi ke i

n = Jumlah dimensi nilai yang berbeda.

Pada penghitungan nilai utility, nilai dihasilkan dari penjumlahan nilai tiap-tiap nasabah lalu dikalikan nilai dari pembobotan subkriteria, lalu hasilnya dijumlahkan. Untuk selanjutnya perhitungan nilai akhir menggunakan rumus:

$$n_1 = \sum_{j=1}^{k} nw_j\ u_{ij} \quad (2.13)$$

Keterangan:

Wj = Bobot dari kriteria ke 1

Uij = Nilai Utility kriteria ke –j untuk keluarga ke-i

ni = Nilai Akhir

Dimana nilai utility dikalikan dengan nilai bobot kriteria. Hasil akhir ini yang akan menentukan pilihan alternative yang akan dipilih. Pada tahap pembobotan, bobot ROC untuk kriteria umum adalah 0.75. Sementara itu untuk kriteria khusus adalah 0.25. Pada penilaian data nasabah, kriteria dipecah menjadi beberapa bagian sub kriteria dan sub subkriteria (Haryanti, D., Nasution, H. & Sukamto, A. S. 2016).

2.1.9. Penelitian Terkait

Beberapa penelitian yang berhubungan algoritma *Ant Colony Optimization* dan *Tabu Search* dapat dilihat seperti pada Tabel II-1.

Tabel II-1 Penelitian Terkait

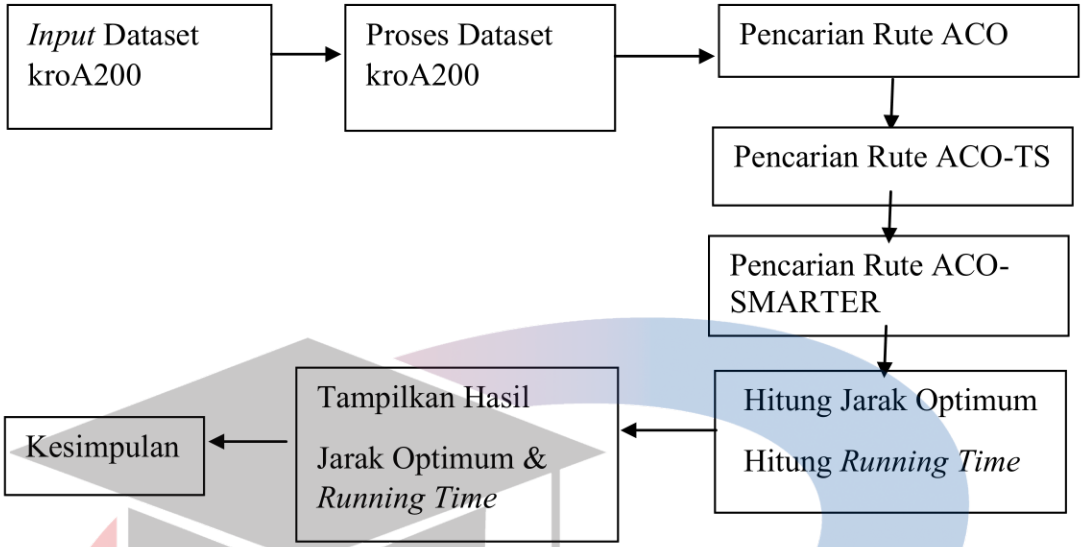
Tahun	Peneliti/Tahun	Metode Yang Digunakan	Keterangan (Hasil)
2016	Jufri, A., Sunaryo & Santoso, P. B., Modifikasi *ACO* untuk Penentuan Rute Terpendek ke Kabupaten/Kota di Jawa (Studi kasus *TSP* kota-kota di Jawa)/2016	Modifikasi *ACO*	Modifikasi Algoritma *ACO* dengan memberikan nilai secara otomatis terhadap parameter jumlah semut sebesar 35% dari ukuran masalah, terbukti mampu meningkatkan kecepatan pencarian rute terpendek sebesar 3 kali dari kecepatan algoritma *ACO* yang asli. Sedangkan dari sisi pemakaian memori, modifikasi *ACO* terbukti lebih efisien dengan rata-rata pemakaian memori 93% dari Algoritma *ACO* yang asli.
2015	Swiatnicki, Z, *Application of Ant Colony Optimization Algorithms for Transportation Problems Using the Example of the Travelling Salesman Problem*	*Ant Colony Optimization Algorithms (ACO)*	Hasil eksperimen menunjukkan bahwa ada perubahan heuristik menyebabkan perubahan efisiensi algoritma dan kualitas solusi yang diidentifikasi. Heuristik Ull ternyata universal. Algoritma menggunakan heuristik tersebut telah

			memberikan hasil yang sangat baik terlepas dari ukuran masalah (organisasi kota). Algoritma yang menggunakan kombinasi dari dua heuristik pembaruan feromon global - *BSF* (rute terbaik-sejauh ini) dan *IB* (rute terbaik-iterasi) menunjukkan efektivitas tinggi. Saat mengoperasikan tidak ada metode pembaruan feromon yang memberikan hasil yang memuaskan.
2016	Khairunnisa, M. P., Pramono, B. & Saputra, R. A. Implementasi Algoritma *Tabu Search* Pada Aplikasi Penjadwalan Mata Pelajaran (Studi Kasus: SMA Negeri 4 Kendari)	*Tabu Search*	Sistem informasi penjadwalan berbasis Algoritma *Tabu Search* dapat mengoptimalkan proses pembelajaran dengan menekan adanya bentrok jadwal, seperti adanya guru yang mengajar di kelas yang berbeda, tetapi di waktu yang sama.
2020	Ginting, Subhan Hafiz Nanda Peningkatan Kinerja	*ACO-SMARTER*	Hasil percobaan diperoleh dari Kriteria perbandingan

	Algoritma Ant Colony Optimization (ACO) dengan Menggunakan Simple Multi-Attribute Rating Technique Exploiting Ranks (Smarter		dalam penelitian ini adalah jarak optimal (terbaik), optimum rata-rata jarak (AVG) dan waktu proses (berjalan waktu). Tiga kriteria tersebut, ACO-SMARTER algoritma lebih unggul kecuali untuk AVG Secara keseluruhan algoritma ACO-SMARTER lebih baik sebesar 26,17% dibandingkan dengan algoritma ACO di penelitian Junjie, P. & Dingwei, W. (2016)

2.2 Kerangka Konsep Pemecahan Masalah

Kerangka konsep pemecahan masalah menggambarkan bagaimana masalah penelitian dapat diselesaikan melalui solusi-solusi yang diusulkan serta dari solusi tersebut diharapkan memiliki dampak yang dapat menyelesaikan permasalahan penelitian. Berikut adalah gambaran kerangka konsep pemecahan masalah dari penelitian yang akan dilakukan, dapat dilihat pada Gambar II-10. berikut:



Gambar II-12. Kerangka Konsep Pemecahan Masalah

Penelitian yang dilakukan sebelumnya oleh (Ginting, 2020) Hasil percobaan diperoleh dari kriteria perbandingan dalam penelitian ini adalah jarak optimal (terbaik), optimum rata-rata jarak (AVG) dan waktu proses (berjalan waktu). Tiga kriteria tersebut, ACO-SMARTER algoritma lebih unggul kecuali untuk AVG Secara keseluruhan algoritma ACO-SMARTER lebih baik sebesar 26,17% dibandingkan dengan algoritma ACO. Solusi untuk mendapatkan *running time* yang lebih singkat serta jarak terbaik, perlu dilakukan perbandingan untuk menentukan kualitas dari suatu algoritma, semakin singkat *running time*, maka akan semakin baik, namun tetap memperhatikan kualitas jalur terpendek yang dihasilkan dari algoritma tersebut. Hal pertama yang akan dilakukan yaitu , input data berupa dataset berupa kordinat kota-kota yang diambil dari dataset Hasler Whithney TSPLIB95. Selanjutnya data tersebut diolah dengan menggunakan algoritma ACO, ACO-TS serta ACO-SMARTER. Hasil proses adalah jarak optimum dan *running time* masing-masing algoritma. Setelah itu diambil kesimpulan besaran nilai *running time* untuk kedua algoritma.

2.3 Hipotesis

Berdasarkan penelitian yang telah diajukan pada bagian sebelumnya. Maka dapat diambil suatu hipotesis atas permasalahan tersebut. Hipotesis tersebut adalah sebagai berikut:

1. Algoritma kombinasi ACO-TS diharapkan mampu mendapatkan *running time* yang lebih cepat serta jarak terpendek yang lebih baik dibandingkan dengan algoritma ACO-SMARTER.
2. Dengan mengkombinasikan algoritma ACO dengan *Tabu Search* akan menjadi solusi untuk mendapatkan algoritma kombinasi yang memiliki *running time* yang lebih singkat serta jarak yang lebih pendek dibandingkan dengan algoritma ACO-SMARTER.