FIKI |Jurnal Teknologi Informasi dan Komunikasi |ISSN : 2087-2372
http://jurnal.unnur.ac.id/index.php/jurnalfiki

PENCARIAN RUTE TRAYEK ANGKOT TERPENDEK MENGGUNAKAN METODE ALGORITMA DIJKSTRA DAN HAVERSINE FORMULA

Tiofani Atsilahasna Labibah
Jurusan Teknik Komputer dan Informatika, Politeknik Negeri Bandung
Jl. Gegerkalong Hilir, Ciwaruga, Kec. Parongpong, Kab. Bandung Barat Jawa Barat, 40559
email: tiofanial@gmail.com

ABSTRAK

Sebuah aplikasi telah dialisis, dirancang, dan diimplementasi, aplikasi ini dirancang dengan menggunakan Algoritma Dijkstra dan Formula Haversine. Tujuan Aplikasi ini adalah pencarian rute terpendek trayek dilakukan dengan menggunakan algoritma dijkstra sebagai algoritma pencarian jarak dan formula haversine untuk mencari jarak antara dua titik dengan koordinat latitude dan longitude pada GoogleMaps. Hasil dari aplikasi ini adalah dapat digunakan untuk membantu orang-orang menggunakan transportasi secara efisien, dengan sampel angkot di Kota Bandung sebagai media penelitian. Untuk pengembagan mengintegrasikan 4 aplikasi: mobile-sensor, aplikasi-pengguna, manajemen-trayek dan server. Sistem dapat diimplementasi penuh ketika seluruh rute, verteks, dan mobile sensor telah tersedia

Kata Kunci: rute terpendek, trayek, angkot, algoritma dijkstra, formula haversine

1. PENDAHULUAN

Smart City merupakan penerapan *information and communication technology* (ICT) untuk menghubungkan (*connecting*), memonitor (*monitoring*), dan mengontrol (*controlling*) berbagai macam sumber daya yang ada di dalam kota secara efektif dan efisien. Angkot merupakan salah satu contoh tranportasi dengan penggunaan yang tepat dan pemanfaatan yang benar dapat mengurangi tingkat kemacetan. Tapi faktanya, angkot memiliki banyak permasalahan yang harus dipecahkan. Terutama, jika angkot akan dijadikan solusi transportasi dalam ranah *smart city*[1].

Permasalahan angkot dapat diamati dari sudut pandang sopir, pengelola angkot, maupun pengguna angkot. Permasalahan dari segi sopir dan pengelola angkot adalah kepemilikannya masih ditangan swasta, sehingga masalah pengelolaan angkot belum dapat terintegrasi dan terpusat di pemerintahan. Hal ini menjadi salah satu penyebab pemerintah sulit memberikan keputusan terkait pengembangan transportasi umum khususnya di Kota Bandung. Dari segi pengguna angkot adalah angkot memiliki aturan trayek sebagai lintasan yang dilalui dan angkot tidak memiliki jadwal operasional tetap. Dua hal tersebut mengakibatkan pengguna beralih menggunakan kendaraan pribadi karena pengguna dapat langsung ke tempat tujuan dan lebih on time, serta masih banyak lagi [2, 1, 3].

Demi ketercapaian moda transportasi yang mendukung smart city [2]. Peningkatan layanan, penyelesaian masalah-masalah angkot, perbaikan sistem dan infrastruktur menjadi hal yang penting untuk dilakukan. Salah satunya dengan melakukan pendekatan teknologi. Saat ini, sudah ada aplikasi terkait angkot dan smart city, seperti travel.kiri yang memberitahu pengguna cara-cara yang diperlukan untuk sampai ke tujuan. Angkot.tibandung.com yang dapat membantu memberikan rekomendasi-rekomendasi angkot yang perlu dinaiki. Tapi, kedua aplikasi tersebut masih dalam keadaan statis atau belum menggunakan data realtime keberadaan angkot di lapangan. Sehingga untuk mendapatkan informasi terkait status angkot, penggunaan, pengelolaan trayek angkot secara maksimal belum dapat terwujud.

2. TINJAUAN PUSTAKA

Saat ini banyak *shortest path* yang dapat digunakan dan mendukung teknologi

geographic information systems (GIS) atau computer system yang berhubungan dengan posisi-posisi di permukaan bumi. Namun, ada satu kunci masalah dalam shortest path. Jika digunakan dalam sebuah network dan secara real-time, maka dapat menyebabkan kompleksitas komputasi perhitungan shortest path yang tinggi diantara lokasi yang berbeda. Belum lagi konflik yang dapat ditimbulkan dari data yang didapat secara bersamaan. Serta jumlah nodes yang banyak *bermunculan* juga berdampak pada banyak memory yang dibutuhkan dalam sebuah *computer*[9].

2.1 *Multi Processing*

Sistem Angkot *Tracer* (khususnya *server*) dapat mengalami *load traffic* dan *concurrent* user *access* yang cukup tinggi.

Concurrent user access adalah istilah yang menggambarkan suatu layanan yang diakses secara bersamaan oleh lebih dari satu *user*.

Salah satu efek yang ditimbulkan adalah *race condition*.

Race condition merupakan kondisi suatu data pada memory ketika sedang dijalankan (*read* and *write*) oleh dua atau lebih proses. Hasil dari proses tersebut bergantung pada urutan eksekusi pada data tersebut[4].

Sebagai contoh[5], terdapat dua proses menambahkan nilai pada suatu variabel dan menyimpannya pada variabel yang sama. Kemudian variabel itu diakses oleh kedua proses tersebut secara bersamaan.

Tabel 2.1 Bentuk Algoritma dalam Proses

Proses A	Proses B
hasil = *hasil* + 500	*hasil* = *hasil* + 200

Tabel 2.1 menggambarkan proses A dan B mengakses variabel hasil yang sama. Varibel hasil digunakan untuk menyimpan nilai yang dihitung oleh masing-masing proses. Proses yang terjadi pada A dan B adalah sebagai berikut:

1. Proses A:
   A1. Load hasil dari memory ke register 1.
   A2. Tambahkan hasil dengan 500
   A3. Simpan kembali nilai yang ada pada register 1 ke lokasi memory dari hasil.
2. Proses B:
   B1. Load hasil dari memory ke register 1.
   B2. Tambahkan hasil dengan 200.
   B3. Simpan kembali nilai yang ada pada register 1 ke lokasi memory dari hasil.

Terdapat 2 skenario yang dapat terjadi ketika kedua proses sedang berjalan, yaitu:

1. Skenario 1:
   A1. Load hasil dari memory ke register 1.
   A2. Tambahkan hasil dengan 500
   A3. Simpan kembali nilai yang ada pada register 1 ke lokasi memory dari hasil.
   B1. Load hasil dari memory ke register 1.
   B2. Tambahkan hasil dengan 200.
   B3. Simpan kembali nilai yang ada pada register 1 ke lokasi memory dari hasil.
   Pada skenario ini, variabel hasil bertambah dengan nilai akhir 700 (500+200).

2. Skenario 2:
   A1. Load hasil dari memory ke register 1.
   A2. Tambahkan hasil dengan 500
   B1. Load hasil dari memory ke register 1.
   B2. Tambahkan hasil dengan 200.
   B3. Simpan kembali nilai yang ada pada register 1 ke lokasi memory dari hasil.
   A3. Simpan kembali nilai yang ada pada register 1 ke lokasi memory dari hasil.
   Pada skenario ini, variabel hasil hanya bertambah dengan nilai pada proses A, yakni 500. Karena hasil yang didapat pada proses B telah berganti menjadi hasil yang didapat pada proses A. Jika 2 hal tersebut terjadi tentu dapat membahayakan integrity atau kesesuaian dari keaslian suatu data [6].

2.2 *Djikstra Algorithm*

Algoritma untuk mencari jalur terpendek dari satu sumber pada graph yang seluruh bobot pada edge-nya bernilai positif. Algoritma ini menerapkan konsep greedy dalam melakukan pencarian rute terpendek[7].

```
for each vertex v in graph G
do begin
  d[s] = 0
  d[v] = ∞
end

{Initialize visited vertices S in
graph G}
S = null

{Initialize Queue Q as set of all
 nodes in graph G}
Q = all vertices V
while Q ≠ ∅
```

```
do begin
        u=mind (Graph G, distance d)
        visited vertices S = S+u

        (Relaxation)
        for each vertex v in neighbor[u]
        do begin
        if d[v] > d[u] + w(u,v)
           then d[v] = d[u] + w (u,v)
           end
        end
        return d
```

2.3 *Haversine Formula*

Formula *haversine* digunakan untuk mencari jarak antara dua titik dengan koordinat *latitude* dan longitude. Berikut ini adalah formula haversine [10]:

$$d = 2r\ sin^{-1}\left(\sqrt{sin^2\left(\frac{\phi_2-\phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\psi_2-\psi_1}{2}\right)}\right)$$

Keterangan:

ψ_2, ψ_1	: Latitude
ϕ_2, ϕ_1	: Longitude
r	: Jari-jari bola
d	: Jarak antara dua titik

Jarak antara dua titik ini digunakan pada Sistem Angkot Tracer ketika pencarian rute angkot. Jarak tersebut muncul saat proses penginputan titik asal dan tujuan oleh pengguna angkot dilakukan.

3. HASIL DAN PEMMBAHASAN

Hasil akhir yang diakomodasi adalah untuk kepentingan manajemen, sopir angkot, dan pengguna angkot untuk pencarian rute. Rute yang ditampilkan adalah rute terpendek secara geografis. Sehingga waktu dan harga dari perjalanan dapat diminimalisir.

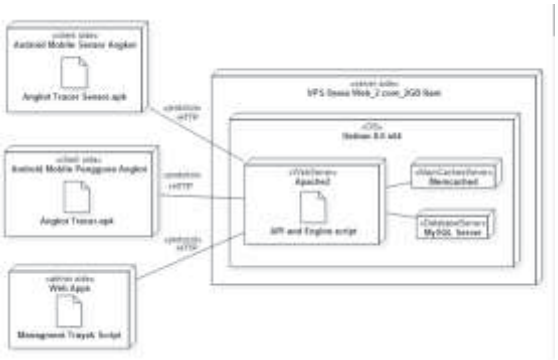
3.1 Infrastruktur dan Diagram Sistem Angkot *Traceri*

Gambar 3.1 Infrastruktur Sistem Angkot *Tracer*

Gambar 3.1 Infrastruktur Sistem Angkot Tracer Teknologi yang digunakan pada Sistem Angkot Tracer, yaitu:

1. *Apache* web *server* sebagai web *service* API Angkot *Tracer*
2. *Memcaced* sebagai aplikasi memcache *server*.
3. MySql sebagai aplikasi *database*.
4. Aplikasi manajemen trayek berbasis web PHP
5. Aplikasi mobile pengguna dan sensor angkot berbasis android.
6. GoogleMaps sebagai *engine* map.

Sedangkan untuk diagram Sistem Angkot *Tracer* dibentuk sebagai berikut :



Gambar 3.2 Diagram Sistem Angkot *Tracer*

Dari diagram di atas dapat terlihat infrastruktur fisik dan logik dari Sistem Angkot Tracer. Development software Sistem Angkot Tracer berada di 5 titik (tergambar dalam notasi artifact), yaitu:

1. Angkot *Tracer* Sensor.apk *Outcome* dari Aplikasi *Mobile* Sensor Angkot.
2. Angkot Tracer.apk *Outcome* dari Aplikasi Mobile Sensor Angkot.
3. Management Trayek *Script Outcome* dari Aplikasi Manajemen Trayek Angkot.
4. API and *Engine Script Outcome* dari API Angkot *Tracer*, termasuk diantaranaya *sciprt* dijkstra all *vertex* dan *script sync vertex from* db *to memory*.

3.2 Trayek Angkot

Angkot berada dalam suatu himpunan trayek. Trayek mendefinisikan struct atau tipe data komposit yang mengandung elemen *variable* berupa nama dari trayek dan jalur dari trayek yang digunakan sebagai titik awal dan akhir perjalanan.

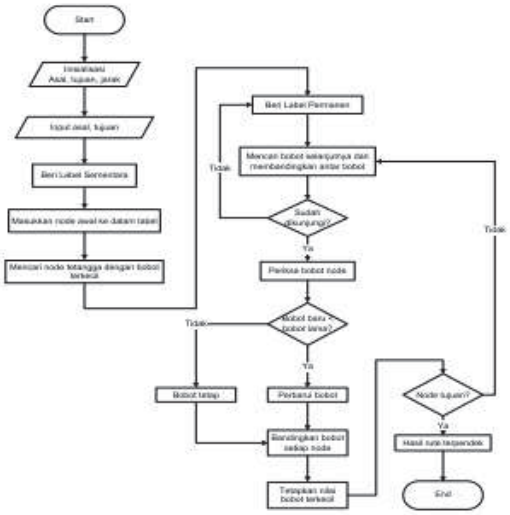
Gambar 3.3 Proses Pembuatan Trayek Angkot

Proses ini dilakukan untuk pembuatan sebuah trayek. Memerlukan data *nodes* (titik – titik) pada Maps yang kemudian digunakan untuk pembuatan jalur angkot menggunakan *polyline*. Untuk mengubah rute trayek, dilakukan dengan mengubah posisi atau menambahkan jalur untuk membentuk rute yang baru (sesuai kebijakan yang berlaku).

3.3 Rute Terpendek

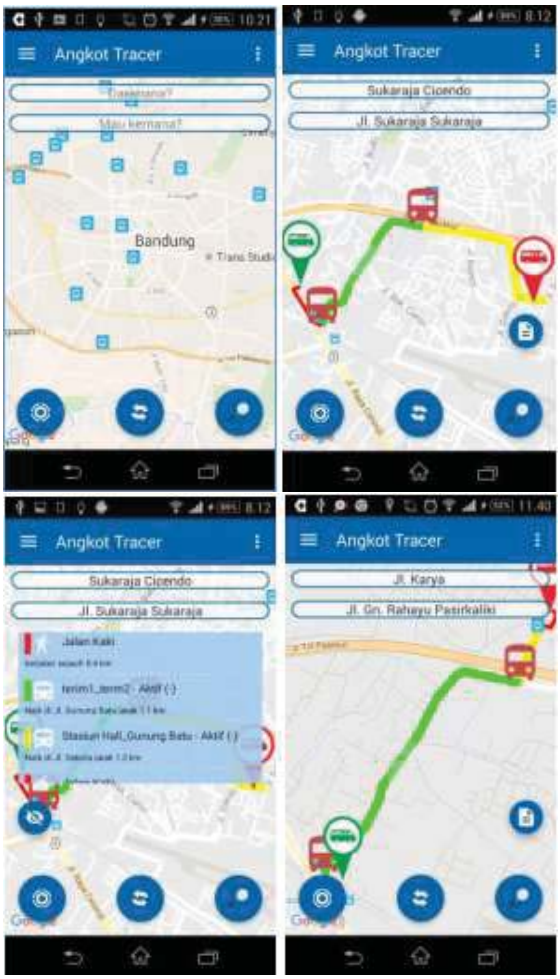
Rute terpendek adalah entitas yang merepresentasikan rute terpendek dari suatu *verteks* ke *verteks* yang lain. Oleh karena itu, entitas ini memiliki atribut *verteksSource* dan *verteksDestination*. Rute terpendek ini secara eksplisit berasosiasi dengan entitas trayek dan edge. Asosiasi tersebut mendefinisikan bahwa rute terpendek tersusun dari nol atau banyak trayek dan *edge*.

Algoritma Dijkstra digunakan terkait optimasi pencarian lintasan terpendek untuk sebuah *graf* berarah dengan bobot-bobot sisi yang bernilai tak-negatif. Dijkstra dapat digunakan untuk menemukan jarak terpendek antara dua *vertex*.



Gambar 3.5 Flowchart Djikstra

Dengan melakukan seleksi seluruh *verteks* yang ada. Kemudian, jarak terpendek dari *verteks* sumber ke *neighbor* verteks tersebut terus di-*update* (relaxation algorithm). Untuk mendapat jarak terpendek ke semua titik pada *graph* [8].



Gambar 3.6 Hasil Implementasi Rute Terpendek

Pengguna dapat mencari rute dengan memasukkan titik asal dan titik tujuan. Hasil dari proses ini adalah informasi berupa rincian urutan angkot trayek yang dapat digunakan untuk mencapai titik tujuan dari titik asal. Selain itu terdapat informasi mengenai banyaknya trayek angkot yang terlibat, serta status angkot pada trayek yang dipilih. Merepresentasikan jarak atau jalur terpendek yang dapat dilalui oleh pengguna angkot dengan pergantian angkot yang paling sedikit.

3.4 Komponen Pendukung

1. Koordinat
   Terdiri atas informasi terkait latitude dan longitude yang digunakan angkot untuk memberikan informasi lokasi angkot tersebut.
2. GoogleMaps
   Merepresentasikan visual peta. Entitas ini secara langsung berasosiasi dengan verteks dan rute terpendek, untuk keperluan menampilkan data secara visual dalam peta. GoogleMaps menampilkan nol atau banyak vertex dan rute terpendek.
3. API Angkot *Tracer*
   Memiliki atribut token berupa kode otentikasi. API Angkot *Tracer* berasosiasi dengan trayek, rute terpendek, dan server. Untuk merepresentasikan API Angkot *Tracer* mengakses hanya ke satu server saja. API Angkot *Tracer* memberikan nol atau banyak trayek dan rute terpendek.
4. Server
   Server berasosiasi satu ke satu dengan entitas *Memcached* (menyimpan data di tepat satu *memcached*).
5. *Memcached*
   Fitur yang bertugas menyimpan dan mengakses data di *memory*.
6. *Update* koordinat angkot
   Fitur dimana ketika mesin mobil aktif, setiap 15 detik 1 kali secara berkala mengirimkan koordinat *Geolocation* dari *mobile* sensor ke *server* angkot *Tracer*.
7. *Update* status angkot
   Fitur ini terdapat pada internal Sistem Angkot *Tracer*. *Scheduler* adalah aktor khusus yang dibuat untuk melakukan aktifitas *monitoring* dalam sistem. *Scheduler* berjalan selama 5 menit, melakukan 1 kali untuk melakukan eksekusi fitur ini. Fitur ini digunakan untuk perubahan status dari angkot-angkot yang ada pada sistem. Fitur ini berdasarkan *geolocation* yang dimiliki oleh masing-masing *mobile* sensor. Aturan yang dimiliki *scheduler* untuk masing-masing angkot, yaitu: Sistem membaca *current latitude* dan *longitude* angkot, *latitude before* dan *longitude before*, *last update* dari data untuk menentukan status angkot. Dari scheduler tersebut didapat hasil, seperti berikut:
   a. Status angkot *off* (tidak beroperasi).
      Jika *delta current time* dan *last update* serta latitude dan longitude angkot diluar radius terminal sejauh 1 Km selama lebih dari 30 menit.
   b. Status *stand by* (sedang di terminal).
      - Jika *delta current time* dan *last update* serta *latitude* dan *longitude* angkot (minimal 1) berada dalam radius terminal (sejauh 1 Km) selama lebih dari 30 menit.
      - Jika *delta current time* dan *last update* serta *latitude* dan *longitude* angkot (minimal 1) berada dalam radius terminal (sejauh 1 Km) selama lebih dari 10 menit dan kurang dari 30 menit.

   c. Status *idle* (ngetem/menunggu).
      - Jika *delta current time* dan *last update* dan *latitude* dan *longitude* angkot diluar radius terminal sejauh 1 Km selama lebih dari 10 menit dan kurang dari 30 menit
      - Jika *latitude* dan *longitude* angkot sama dengan latitude *before* dan *longitude before* angkot.

   d. Status *running* (aktif beroperasi).
      Jika tidak termasuk ke dalam 4 kategori di atas.
8. *Update* jumlah angkot aktif
   Scheduler disini digunakan untuk melakukan pengecekan status terhadap seluruh angkot per trayeknya. Kemudian pengecekan tersebut dihitung dan disimpan di *memory*. Hasil dari proses ini adalah mendapatkan jumlah angkot aktif per trayeknya yang disimpan di *memory*.
9. *Sync* rute terpendek ke *memory*
   Scheduler disini dibuat untuk melakukan aktifitas *monitoring* dalam sistem. *Scheduler* mengeksekusi fitur *sync* rute terpendek ke *memory* pada setiap *server*. Sehingga, seluruh data rute terpendek

yang berada di *database* diaktifkan di *memory* untuk keperluan running aplikasi

3.5 Pemrograman

Bagian ini menjelaskan mengenai cuplikan program dari implementasi Sistem Angkot *Tracer*.

Haversine Formula

Untuk Mencari titik pada peta yang terdekat dengan titik pada polyline dengan *index vertexIndex*

```
circle ← new google.maps.Circle(
(center:{lat:Number(vertex.lat()),
lng:Number(vertex.lng())},
  radius:100,
  map:map,
  visible:false))
  isContainMarker←false
  markersContainedIndex ← new Array()
  while(not isContainMarker)
  do
     for (i = 0 to allVertices.length-1, i++)
     do
     latLng ← new google.maps.LatLng
     (Number(allVertices[i].VERTEX_LAT),
     Number(allVertices[i].VERTEX_LONG))
     if(circle.getBounds().contains(latLng))
     then
         markersContainedIndex.push(i)
             isContainMarker←true
     end if
     end for
     if(not isContainMarker)
     then

    circle.setRadius(circle.getRadius()+100)
    end while
    minDistance←Number.MAX_VALUE
    minIndex=-1
    for(i=0 to markersContainedIndex.length-1,
    i=i+1)
    do
     center={lat:Number(vertex.lat()),
     lng:Number(vertex.lng())}
     v ← {lat:Number(allVertices
    [markersContainedIndex[i]].VERTEX_LAT),
    lng:Number(allVertices
    [markersContainedIndex[i]].VERTEX_LONG)}
     distance ← haversineDistance(center,v)
     if(distance<minDistance)
     then
        minDistance ← distance
        minIndex←markersContainedIndex[i]
     end if
    end for
    nearestVertex ← new google.maps.LatLng
    (Number(allVertices[minIndex].VERTEX_LAT),
    Number(allVertices[minIndex].VERTEX_LONG))
return nearestVertex
```

Menggunakan formula *haversine* untuk menghitung jarak antara *verteks1* dengan *verteks2*.

```
rEarth ← 6371000
lat1 ← radians(v1.lat)
lat2 ← radians(v2.lat)
lng1 ← radians(v1.lng)
lng2 ← radians(v2.lng)

operand1 ← Math.pow(Math.sin((lng2-lng1)/2),2)
operand2 ← Math.cos(lng1)*Math.cos(lng2)*
Math.pow(Math.sin((lat2-lat1)/2),2)

d ←
2*rEarth*Math.asin(Math.sqrt(operand1+operand2))
return d
```

Dijkstra

Dijkstra digunakan pada proses mendapatkan rute terpendek untuk disimpan dan diproses pada tahapan selanjutnya.

```
Procedure Dijkstra (input m:matriks, s:simpul
awal)
   {
   Mencari lintasan terpendek dari simpul awal
s ke semua simpul lainnya
   Masukan (input) : matriks ketetanggaan (m)
dari graf  berbobot G dan simpul awal s
   Keluaran (output) : lintasan terpendek dari
s ke semua simpul lainnya
   }

   Deklarasi
   s1, s2, ....., sn  : integer
   d1, d2, ....., dn : integer
   i, j, k : integer
   Algoritma
(langkah 0 (inisialisasi:))
      for i ← 1 to n do
          si ← 0
          di ← msi
      endfor
(langkah 1:)
   sn ← 1 (karena simpul s adalah simpul asal
lintasan terpendek, jadi simpul s sudah pasti
terpilih dalam lintasan terpendek)
   dc ← ∞ (tidak ada lintasan terpendek dari
simpul s ke s)
   (langkah 2, 3, ..., n-1:)
   for k ← 2 to n-1 do
         j ← simpul dengan  sj = 0 dan  dj
minimal
       sj ← 1 (simpul j sudah terpilih ke
dalam lintasan terpendek)
       (perbarui tabel d)
       for semua simpul i dengan si = 0 do
            if  dj + mji < di then
                di ← dj + mji
            endif
       endfor
   endfor
}
```

4. PENUTUP

Sistem Angkot *Tracer* saat ini baru terimplementasi sebanyak 3 trayek, 478 *verteks* dan 624 *edge* , dan rute terpendek yang kurang lebih mencakup 125.493 perjalanan. Sistem dapat diimplementasi penuh ketika seluruh rute, *verteks*, dan *mobile* sensor telah tersedia.

Sistem Angkot Tracer dibuat untuk menangani kepentingan terkait angkot (di Kota Bandung) secara umum dapat:

1. Mengakuisisi data koordinat angkot untuk kepentingan kontrol dan *monitoring*.
2. Sistem Angkot *Tracer* dapat mendefinisikan status dari masing-masing angkot terdaftar dalam kasus ini status angkot aktif, tidak aktif serta sedang *idle*/nge-tem.
3. Menghasilkan informasi rute perjalanan terpendek dengan menggunakan algoritma dijkstra yang kemudian direpresentasikan menjadi trayek-trayek yang perlu dinaiki

dalam kasus ini melalui aplikasi *mobile* pengguna angkot.
4. Melakukan pengelolaan trayek terkait menejerial trayek angkot, dalam kasus ini melalui aplikasi manajemen trayek angkot.
5. Memberikan informasi jumlah angkot aktif secara *realtime* beserta koordinatnya untuk kepentingan terkait data mining serta berbagai hal terkait big data selanjutnya,

Berdasarkan hasil dari penelitian ini, adapun saran yang diberikan untuk pengembangan selanjutnya yaitu:
1. Metode dalam proses pembuatan rute terpendek dapat dikembangkan atau dimodifikasi untuk menangani kompleksitas komputasi mengubah ratusan data *edge* menjadi rute terpendek dengan waktu yang lebih efektif dan relative singkat (*generate* seluruh rute terpendek 500-1000 *edge* dengan waktu max 1 – 2 hari).
2. Studi lebih lanjut mengenai penggunaan algoritma A* pada Sistem Angkot *Tracer*.
3. Pada algoritma pencarian rute terpendek perlu memerhatikan jalur yang dicari adalah bagian dari jalur yang telah terdaftar, sehingga dapat mengurasi *resource storage*.
4. Perlu adanya penanganan overlap jalur angkot yang melewati rute dan arah yang sama dalam satu waktu.
5. Adanya perluasan wilayah, tidak hanya menangani di Kota Bandung saja.
6. Perbaikan posisi *vertex* untuk menghasilkan tampilan yang sesuai.
7. Memberikan rekomendasi titik yang bertetanggaan secara *graf* untuk meminimalisasi kesalahan memasukkan jalur angkot ketika tambah trayek.
8. Menampilkan *edge* yang tidak ada di database saat melakukan tambah dan edit trayek

DAFTAR PUSTAKA

[1] Kamil R., "Smart City Bandung", https://sustainabledevelopment.un.org/content/documents/12659kamil.pdf. Diakses, 25 Maret 2016
[2] Aminah S., "Transportasi Publik dan Aksesibilitas Masyarakat Perkotaan" http://www.journal.unair.ac.id/filerPDF/Transportasi%20Publik%20dan%20Aksesibilitas.pdf. Diakses, 25 Maret 2016
[3] Rini, Interviewee (2015) : Smart Transportation Bandung. IDMF-ITB, Desember 2.
[4] S. Carr, J. Mayo dan C.-K. Shene, (2001). "Race Conditions: A Case Study" : The Journal of Computing in Small Colleges, vol. 17, pp. 88-102
[5] A. Stavrou, "CS 571 Operating Systems-Process Synchronization" https://cs.gmu.edu/~astavrou/courses/CS_571_F09/CS571_Lecture3_synchronization.pdf. Diakses, 8 Januari 2016
[6] Arius Dony (AMIKOM), (2009). "Aspek-Aspek Keamanan Komputer" http://amikom.ac.id/research/index.php/karyailmiahdosen/article/view/1305. Diakses, 24 Maret 2016
[7] N. Choubey, (2013). "Survey on Certain Algorithms Computing Best Possible Routes for Transportation Enquiry Services" : International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), vol. 2, no. 1, pp. 238-242, Januari
[8] V. Patel dan C. Baggar, (2014). "A Survey Paper of Bellman-Ford Algorithm and Dijkstra Algorithm for Finding Shortest Path in GIS Application" : International Journal of P2P Network Trends and Technology (IJPTT), vol. 5, no. 2, pp. 1-4, Februari
[9] D. P. V. Ingole dan M. M. K. Nichat, (2013). "Landmark based shortest path detection by using Dijkestra Algorithm and Haversine Formula" : International Journal of Engineering Research and Applications (IJERA), vol. 3, no. 3, pp. 162-165
[10] M. Yan, "DIJKSTRA'S ALGORITHM", http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf. Diakses, 20
[11] Maret 2016 Putra, Pahlevi Ridwan, dkk. 2016. "Pencarian Rute Trayek Angkot Terpendek di Kota Bandung pada Sistem Angkot Tracer".Teknik Komputer dan Informatika. Politeknik Negeri Bandung. Bandung.