Optimisasi Rute Menuju Layanan Kesehatan dalam Upaya Mengurangi Risiko Kematian Pasien Gawat Darurat pada Pengembangan Kota Pintar

Addin Nabilal Huda - 13520045[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13520045@std.stei.itb.ac.id

***Abstrak*—Laju urbanisasi yang tinggi menimbulkan beberapa persoalan yang menuntut perkembangan kota pintar untuk meningkatkan kualitas hidup penduduk. Persoalan optimisasi rute untuk mengakses unit layanan kesehatan merupakan aspek penting pengembangan kota pintar dalam upaya mengurangi risiko kematian pasien gawat darurat. Persoalan ini dapat dimodelkan dengan graf berbobot berarah dengan komponen simpul graf berupa persimpangan jalan dan lokasi unit layanan kesehatan serta komponen sisi graf berupa jalan yang menghubungkan dua simpul. Optimisasi rute dilakukan dengan analisis waktu tercepat berdasarkan jarak dan kemacetan untuk menuju ke unit layanan kesehatan dengan mempertimbangkan ketersediaan unit gawat darurat. Sistem optimisasi menggunakan teknologi berupa *Global Positioning* System (GPS), *Floating Car* Data (FCD), dan *Geographic Information* System (GIS) serta memanfaatkan *map matching algorithm* dan algoritma Dijkstra yang dimodifikasi.**

***Kata Kunci*—Optimisasi Rute, Gawat Darurat, Kota Pintar, Graf Berbobot.**

I. PENDAHULUAN

Berdasarkan data dari United Nations, saat ini 55% populasi dunia tanggal di perkotaan. Angka ini diperkirakan akan terus meningkat dan mencapai angka 68% sebelum 2050. Besarnya populasi penduduk perkotaan memunculkan masalah di berbagai aspek kehidupan seperti mobiliasasi, energi, sanitasi, ekonomi, dan kesehatan. Fenomena ini mengarah pada mendesaknya kebutuhan akan inovasi berupa konsep kota yang dapat mengatasi berbagai persoalan sekaligus dapat meningkatkan kualitas hidup penduduk. Berangkat dari hal tersebut, belakangan berbagai pihak sedang berupaya mengembangkan konsep kota pintar. Tujuan utama pengembangan kota pintar adalah untuk mengoptimisasi fungsi kota, mendukung pembangunan ekonomi, dan meningkatkan kualitas hidup penduduk melalui pemanfaatan teknologi informasi.

Sebagai aspek fundamental kehidupan manusia, aksesibilitas layanan kesehatan tidak luput dari aspek yang diperhatikan dalam pengembangan kota pintar. Tantangan untuk meningkatkan aksesibilitas layanan kesehatan merupakan konsekuensi dari meningkatnya jumlah penduduk yang tidak selalu dibarengi dengan meningkatnya kemudahan akses layanan kesehatan. Salah satu masalah nyata adalah penanganan pasien gawat darurat yang kurang cepat tertangani. Menurut Kepala Pusat Krisis Kesehatan Kemenkes RI, 70% angka kematian di Indonesia terjadi sebelum sampai di rumah sakit. Sebuah studi menyatakan tingginya angka kematian pasien gawat darurat di antaranya disebabkan oleh lokasi rumah sakit yang sulit dicari serta lalu lintas yang sering macet dan sulit diprediksi. Selain itu, pada masa pandemi Covid-19, banyak kasus kematian pasien gawat darurat disebabkan oleh unit pelayanan kesehatan yang sudah penuh. Masalah-masalah tersebut melatarbelakangi pentingnya optimisasi akses menuju layanan kesehatan dari aspek rute perjalanan dan kondisi ketersediaan unit layanan kesehatan.

Persoalan ini dapat direpresentasikan dengan graf berbobot berarah dan dioptimisasi menggunakan algoritma Djikstra. Sejalan dengan konsep kota pintar, implementasi solusi yang dibahas dalam karya tulis ini memanfaatkan teknologi informasi seperti *Global Positioning System* (GPS), *Geographic Information System* (GIS), dan *Floating Car Data* (FCD). Optimisasi ini bertujuan untuk mengurangi risiko kematian pasien gawat darurat dengan mencari rute terbaik menuju unit layanan kesehatan ditinjau dari aspek jarak, kemacetan, dan ketersediaan unit gawat darurat.

II. TEORI DASAR

A. Graf

Graf merupakan salah satu bentuk struktur data yang merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Sebuah graf G dapat dinyatakan dengan pasangan himpunan

$$G=(V,E)$$

yang dalam hal ini

- V adalah himpunan tidak kosong dari simpul-simpul (*vertices)* yakni { $v_1, v_1, \ldots, v_n$ }
- E adalah himpunan sisi (*edges)* yang menghubungkan sepasang simpul { $e_1, e_1, \ldots, e_n$ }

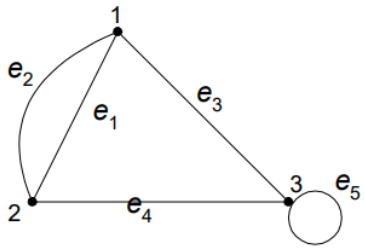
Terminologi yang sering digunakan dalam graf di antaranya:

1. Gelang *(loop)*

   Gelang merupakan terminologi yang digunakan untuk menggambarkan sisi yang berawal dan berakhir pada simpul yang sama. Gelang pada graf pada gambar ditunjukkan oleh sisi e_5.

2. Sisi ganda (*multiple edges/paralel edges)*

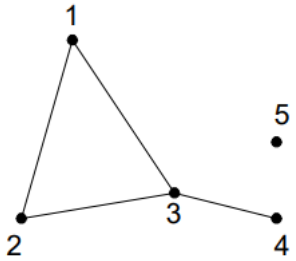
Sisi ganda merupakan pasangan sisi yang mengubungkan dua buah simpul yang sama. .Sisi ganda pada graf pada gambar ditunjukkan oleh pasangan sisi e_1 dan e_2.



Gambar 1. Graf yang memiliki sisi gelang dan sisi ganda
Sumber: [9]

3. Bertentangga (*adjacent)*

Simpul v_1 dan v_2 pada graf G disebut bertentangga jika v_1 terhubung langsung dengan v_2 melalui sisi e=(v_1, v_2). Dalam graf pada gambar, simpul 1 bertetangga dengan simpul 2 dan 3, tetapi tidak bertetangga dengan simpul 5 dan 4.



Gambar 2. Graf untuk menunjukkan hubungan ketetanggaan
Sumber: [9]

4. Bersisian (*incidency)*

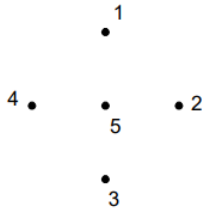
Untuk sembarang sisi e=(u,v), sisi e dikatakan bersisian dengan simpul u dan simpul v. Dalam graf pada gambar, sisi e_2 bersisian dengan simpul 1 dan 2, tetapi tidak bersisian dengan simpul 3.

5. Simpul terpencil

Simpul terpencil merupakan simpul yang tidak memiliki sisi yang bersisian dengan simpul tersebut. Pada gambar, simpul 5 merupakan simpul terpencil karena tidak ada sisi yang beririsan dengan simpul tersebut.

6. Graf kosong

Graf kosong merupakan graf yang memiliki himpunan sisi berupa himpunan kosong. Graf kosong dinotasikan dengan N_n dengan n adalah jumlah simpul.



Gambar 3. Graf kosong
Sumber: [9]

7. Derajat (*degree)*

Derajat suatu simpul merupakan jumlah sisi yang bersisian dengan simpul tersebut. Derajat simpul dinyatakan dengan notasi d(v). Berdasarkan lemma jabat tangan, jumlah derajat semua simpul pada suatu graf adalah genap, yakni dua kali jumlah sisi graf tersebut.

8. Lintasan (*path)*

Suatu lintasan dengan panjang n pada suatu graf G dari simpul awal v_0 menuju simpul tujuan v_n adalah barisan berselang-seling antara simpul-simpul dan sisi-sisi berbentuk v_0, e_1, v_1, e_2, v_2,…, v_{n-1}, e_n, v_n sedemikian sehingga $e_1=(v_0, v_1)$, $e_2=(v_1, v_2)$,…, $e_n=(v_{n-1}, v_n)$ adalah sisi-sisi graf G.

9. Siklus (*cycle)* atau sirkuit (*circuit)*

Siklus atau sirkuit adalah lintasan yang berawal dan berakhir pada simpul yang sama. Suatu sirkuit dikatakan sirkuit sederhana bila melalui sisi yang berbeda, yakni ketika sebuah sisi tidak boleh dilalui lebih dari sekali.

10. Keterhubungan (*connected)*

Simpul v_1 dan simpul v_2 dikatakan terhubung bila terdapat lintasan dari v_1 ke v_2. Graf tak berarah g disebut graf terhubung bila untuk setiap pasang v_i dan simpul v_j dalam himpunan simpul V terdapat lintasan dari v_i ke v_j. Graf berarah G disebut terhubung bila graf tak berarahnya terhubung. Definisi keterhubungan pada graf berarah dibagi menjadi graf terhubung kuat, yaitu graf berarah yang untuk sembarang simpul di graf tersebut terdapat lintasan dan graf terhubung lemah, yaitu graf berarah yang tidak semua pasangan simpul mempunyai lintasan.

(a) (b)

Gambar 4. (a) Graf terhubung (b) graf tidak terhubung
Sumber: [9]

11. Upagraf (*subgraph)*

Graf G1=(V_1, E_1) disebut upagraf dari graf G=(V,E) jika $V_1 \subseteq V$ dan $E_1 \subseteq E$.

12. Upagraf merentang (*spanning subgraph)*

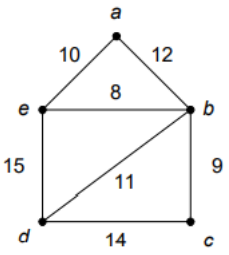
Upagraf G1=(V_1, E_1) disebut upagraf merentang dari graf G=(V,E) bila G1 mengandung seluruh simpul G, yaitu ketika V_1=V.

13. *Cut-set*

Cut-set dari graf terhubung G adalah himpunan sisi yang bila dihilangkan akan menyebabkan G menjadi tidak terhubung. Suatu *cut-set* tidak boleh mengandung himpunan bagian *cut-set* lainnya.

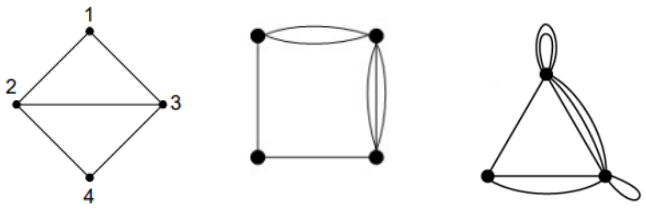
14. Graf berbobot (*weighted graph)*

    Graf berbobot merupakan graf dengan sisi yang memiliki "bobot" atau "harga". Bobot sisi tersebut dapat merepresentasikan jarak, waktu, atau apapun yang dapat menggambarkan keterhubungan antara pasangan simpul yang bertetangga, misalnya jarak atau biaya perjalanan.



Gambar 5. Graf berbobot
Sumber: [9]

Graf dapat digolongkan menjadi beberapa jenis bergantung pada karakter pengelompokan. Berdasarkan keberadaan sisi ganda atau gelang, graf digolongkan menjadi graf sederhana dan graf tak-sederhana. Graf sederhana didefinisikan sebagai graf yang tidak mengandung gelang atau sisi ganda. Sedangkan, graf tak-sederhana merupakan graf yang memiliki gelang atau sisi ganda. Graf tak-sederhana dapat dikelompokkan menjadi graf ganda, yaitu graf yang mengandung sisi ganda dan graf semu, yaitu graf yang mengandung sisi gelang dan mungkin memiliki sisi ganda.



(a) (b) (c)

Gambar 6. (a) Graf sederhana (b) Graf tak-sederhana ganda (c) Graf tak-sederhana semu
Sumber: [9]

Berdasarkan orientasi arah pada sisi, graf dapat dibedakan menjadi graf berarah, yaitu graf yang tidak mempunyai orientasi arah dan graf berarah, yaitu graf yang sisi-sisinya memiliki orientasi arah.

(a) (b)

Gambar 7. (a) Graf tak-berarah (b) Graf berarah
Sumber: [9]

B. Algoritma Dijkstra

Algoritma Djikstra merupakan algoritma yang dapat digunakan untuk memecahkan persoalan pencarian jarak terendah antarsimpul pada suatu graf berbobot (*shortest path problem)*. Algoritma ini pertama kali diusulkan oleh seorang *computer scientist* berkebangsaan Belanda bernama Dr. Edsger W. Dijkstra pada tahun 1959. Algoritma Dijkstra termasuk ke dalam bentuk *greedy algorithm*

Dengan algoritma Dijkstra, dapat ditentukan jarak terendah antara simpul asal menuju seluruh simpul lainnya dalam graf. Algoritma Dijkstra menganalisis jarak antarsimpul dengan terus mencatat jarak terpendek terbaru dari simpul asal ke setiap simpul dan memperbaharui nilai jarak tersebut setiap kali menemukan jarak yang lebih pendek. Dalam prosesnya, ketika algoritma sudah menemukan jarak terpendek antara simpul asal ke simpul lainnya, simpul tersebut akan ditandai dengan *visited* dan ditambahkan ke suatu data. Proses ini terus berlanjut sampai seluruh simpul telah dikunjungi, yakni hingga sudah ditemukan rute yang menghubungkan simpul asal ke seluruh simpul lainnya serta rute terpendek antara simpul asal ke seluruh simpul lainnya

Secara umum, pseudocode algoritma Dijkstra pada graf berbobot yang direpresentasikan oleh *adjacency matrix* adalah sebagai berikut

```
void Dijkstra(int g[V][V], int src)
  dist[V]: arr V of integer
  visited[V]: arr V of boolean
  i,j: integer
  for i ← 0 to V
    dist[i] ← INT_MAX
    visited[i] ← false
  endfor
  dist[src] ← 0

  for count ← 0 to V
    u ← minDistance(dist, visited)
    visited[u] ← true
    for v ← 0 to V
      if (not(visited[v]) and g[u][v] and
          (dist[u]!=INFINITY) and
          (dist[u]+g[u][v]<dist[v])) then
             dist[v] ← dist[u] +g[u][v]
    enfor
  endfor
```

Prosedur `Dijkstra` pada pseudocode tersebut menerima input berupa *adjacency matrix* g dengan jumlah simpul sebanyak V dan simpul `src` sebagai simpul asal. Pertama didefinisikan *array* bernama `dist` dengan elemen berupa integer sebanyak V yang digunakan untuk menyimpan nilai jarak terpendek antara simpul asal `src` menuju seluruh simpul pada graf. `dist[i]` menyimpan nilai jarak terpendek antara `src` dan simpul i. *Array* `visited` dengan elemen berupa boolean sejumlah V elemen didefinisikan sebagai tempat menyimpan keterangan apakah jara terpendek dari `src` sudah terfinalisasi. Selanjutnya, seluruh elemen pada array `dist` diinisialisasi dengan infinity dan seluruh elemen pada array `visited` diinisialisasi dengan nilai false.

Misal sebelumnya sudah didefinisikan fungsi `minDistance` yang menerima parameter berupa array `dist` dan `visited` dan mengembalikan simpul demgan jarak terpendek yang sebelumnya belum dimasukkan ke rute

terpendek/elemen pada `visited` bernilai false. Pada proses penentuan rute terpendek ke semua simpul, akan dilakukan iterasi sebanyak V kali untuk memroses hasil *return* fungsi `minDistance` dan menyimpannya ke dalam variabel u. Simpul u ditandai sudah diproses dengan menginisialisasi `visited[u]` dengan nilai true. Selanjutnya, dilakukan iterasi dengan menganalisis seluruh simpul v untuk memperbaharui nilai jarak terpendek pada `dist[v]` bila simpul v belum pernah diproses sebelumnya, terdapat hubungan antara simpul u dan v, dan total jarak dari `src` ke v melalui u lebih kecil dari nilai yang terdapat saat ini di `dist[v]`.

Pada akhir proses, elemen-elemen pada dist sudah berisi jarak terpendek antara `src` dengan setiap simpul dan nilai elemen pada `visited` seluruhnya bernilai true yang berarti setiap simpul sudah selesai diproses. Kompleksitas waktu pada algoritma Dijkstra yang menggunakan representasi *adjacency matrix* adalah O(V^2) dan dapat dioptimisasi menggunakan representasi lainnya, misal dengan representasi *adjacency list* dengan bantuan struktur data *binary heap.*

Algoritma Dijkstra dapat diimplementasikan untuk berbagai *path problem* seperti pada persoalan jaringan telepon dan rute transportasi umum. Dalam impelemtasinya, komponen-persoalan yang diselesaikan. Simpul, sisi, bobot sisi, dan jarak pada graf dapat memiliki merepresentasikan makna berbeda. Selain itu, prioritas analisis juga dapat diubah menjadi terurut membesar atau mengecil. Kode juga dapat dimodifikasi untuk hanya menganalisis jarak antara suatu simpul ke simpul tertentu (tidak seluruh simpul).

C. Floating Car Data dan *Map Matching Algorithm*

Sistem FCD mengumplkan data berupa posisi aktual, arah, kecepatan, dan informasi lainnya dari sebuah kendaraan yang dilengkapi GPS(Global Positioning System). Sistem yang dilengkapi dengan FCD dapat menghasilkan informasi dinamis mengenai waktu tempuh dan informasi lainnya yang dapat diperoleh dari waktu tempuh dinamis. Dalam membangun sebuah sistem FCD, diperlukan *map matching algorithm* untuk mengakses kondisi lalu lintas terkini. Model algoritma tersebut termasuk faktor arah, jarak, dan aksesibilitas lalu lintas. Berikut salah satu formula dasar yang digunakan pada model *map matching algorithm:*

$$f\,(MM) = g\,(Dis) + h\,(Dir) + l(Acc) \ldots (1)$$

$$g\,(Dis) = W_{dis} \times \frac{Maxdis - Dis}{Maxdis} \ldots (2)$$

$$h\,(Dir) = W_{dir} \times \frac{Maxdir - Dir}{Maxdir} \ldots (3)$$

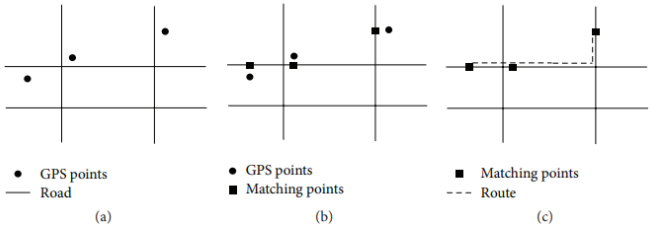
$$\mathrm{l(Acc)} = W_{acc} \times \frac{\mathrm{Maxacc} - \mathrm{Dispp}}{Maxacc} \mathrm{Maxacc} \ldots (4)$$

dengan f(MM) merupakan hasil komprehelnsif dari segi jarak, arah, dan aksesibilitas lalu lintas. Parameter g(Dis), h(Dir), dan l(Acc) berturut-turut merupakan faktor jarak, arah, dan aksesibilitas. Dis, Dir, dan Acc merupakan masukan jarak, arah, dan aksesibilitas, sementara $W_{dis}, W_{dir}\, W_{acc}$ berturut-turut merupakan bobot pengali untuk ketiga faktor tersebut dengan jumlah antara $W_{dis}, W_{dir}\, W_{acc}$ sama dengan satu. Dis dan Maxdis menggambarkan jarak/panjang jalanan dan jarak maksimum jalanan. Maxdir merupakan sudut maksimal antara arah kemudi dan azimut jalan. Sementara, Dispp dan Maxacc adalah jarak dari titik-titik yang berhubungan dan jarak maksimum kendaraan dalam waktu teretentu.

Berikut merupakan contoh FCD:

Timestamp	CarID	X	Y	*Speed*	*Angle*
12244501	13512	114.311214	30.610150	16.512	124.39
12244501	18759	114.305105	30.613587	10.15	125
12244601	14702	115.29425	30.619715	10.654	298.99

Tabel 1. Contoh data FCD



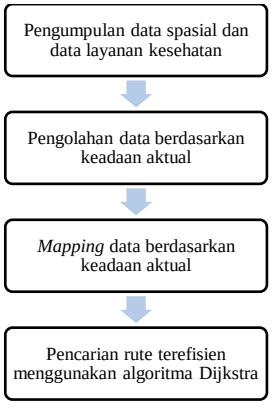
Gambar 8. Contoh data FCD dan hasil *map matching algorithm*
Sumber: [9]

III. PEMBAHASAN

A. Gambaran Umum Sistem

Secara umum, proses yang terjadi dalam sistem yang akan dibuat dimulai dengan pengumpulan data spasial seperti lokasi unit layanan kesehatan, jalanan, dan persimpangan jalan serta data umum terkait unit layanan kesehatan. Data spasial diperoleh dari proses digitalisasi peta area yang sekaligus dapat menyediakan koordinat posisi unit layanan kesehatan. Data spasial ini terintegrasi dengan *Geographic Information System (GIS).* Data umum terkait unit layanan kesehatan seperti jumlah unit gawat darurat diperoleh dari survei langsung ke rumah sakit terkait atau dari data pemerintah. Data-data tersebut merupakan data yang dinamis, artinya dapat berubah-ubah sehingga perlu adanya sistem informasi yang terintegrasi antara unit layanan kesehatan dan sistem optimasi rute.

Data yang diperoleh kemudian akan diolah dan dilakukan pemetaan lokasi-lokasi terkait dalam bentuk graf berbobot menggunakan *map matching algorithm* untuk dapat diproses dalam algoritma Djikstra. Algoritma Dijkstra pada sistem akan menghasilkan rute terefisien, yaitu rute tercepat yang ditinjau dari segi jarak dan kondisi lalu lintas menuju ke layanan kesehatan dengan unit gawat darurat yang masih tersedia.



Gambar 9. Rancangan Sistem
Sumber: Dokumentasi pribadi

B. Implementasi Sistem

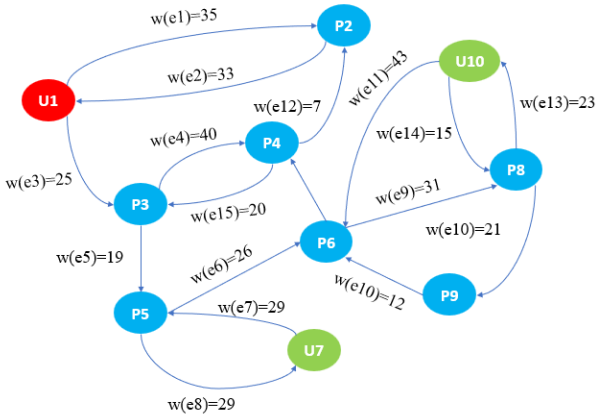
Untuk dapat diproses dalam algoritma Dijkstra, data yang diperoleh perlu diolah menjadi struktur data graf menggunakan GPS dan *map matching algorithm*. Setelah itu, dilakukan kalkulasi untuk menentukan bobot sisi graf dengan analisis rata-rata waktu kendaraan yang melewati sisi tersebut berdasarkan jarak dan kemacetan. Sistem ini menggunakan algoritma Dijkstra yang dimodifikasi dengan komponen simpul yang menggambarkan persimpangan jalan dan lokasi unit layanan kesehatan. Lokasi-lokasi tersebut ditandai sehingga dapat dibedakan antara simpul yang menunjukkan persimpangan jalan dan simpul yang menunjukkan lokasi layanan kesehatan. Sedangkan, sisi graf menggambarkan kondisi lalu lintas jalan yaitu jarak antarpersimpangan jalan dan kemacetan di jalur tersebut. Pembobotan pada komponen graf dilakukan dengan dasar sebagai berikut:

- Bobot sisi merupakan hasil analisis waktu yang dibutuhkan untuk melalui jalanan dari satu persimpangan jalan ke persimpangan jalan lainnya. Analisis waktu diperoleh berdasarkan jarak dan *rate* yang menggambarkan kemacetan di lokasi tersebut. Bobot sisi semakin kecil berarti waktu semakin cepat (lebih baik dari segi jarak dan kondisi lalu lintas)
- Simpul lokasi layanan kesehatan menyimpan data aktual kondisi layanan kesehatan tersebut yaitu ketersediaan unit gawat darurat.

Struktur data graf berbobot yang digunakan memiliki komponen sebagai berikut:

- Simpul menyimpan informasi PointID untuk menandakan ID persimpangan jalan atau unit layanan kesehatan, dan informasi lokasi berupa koordinat garis lintang dan bujur
- Sisi menyimpan informasi EdgeID, dua simpul yang bersisian dengannya yaitu StartID dan EndID, dan bobot sisi

Misalkan pada kota A akan dibuat sistem optimasi rute menuju layanan kesehatan. Setelah dilakukan pengumpulan data spasial menggunakan GPS dan data rumah sakit, data tersebut diolah menggunakan *map matching algorithm* hingga menghasilkan struktur data berupa graf berbobot yang siap diproses untuk mencari rute perjalanan terefisien untuk mengakses unit layanan kesehatan. Data kota A tersebut dapat dimodelkan dengan graf berbobot sebagai berikut:



Gambar 10. Pemodelan rute lalu lintas kota A
Sumber: Dokumentasi pribadi

Informasi sisi graf berbobot tersebut dimuat pada tabel berikut:

EdgeID	StartID	EndID	Weight
e1	1	2	35
e2	2	1	33
e3	1	3	25
e4	3	4	40
e5	3	5	19
e6	5	6	26
e7	7	5	29
e8	5	7	29
e9	6	8	31
e10	8	9	21
e11	10	6	43
e12	4	2	7
e13	8	10	23
e14	10	8	15
e15	4	3	20

Tabel 2. Data komponen sisi graf berbobot pada pemdolen rute lalu lintas kota A.

Sementara, simpul-simpul yang menandai persimpangan jalan dan unit layanan kesehatan dimuat dalam tabel berikut dengan PointID yang diawali dengan huruf U menandai lokasi unit layanan kesehatan dan PointID yang diawali huruf P menandai persimpangan jalan.

PointID	Lon (koordinat bujur)	Lan (koordinat lintang)
U1	114.5586	122.9945
P2	114.5621	123.1121
P3	114.3399	122.9959
P4	114.3346	122.9961
P5	114.3289	122.9961
P6	114.3401	123.0010
U7	114.3612	122.9994
P8	114.3410	123.0097
P9	114.2989	123.0012
U10	113.3479	123.0089

Tabel 3. Data komponen simpul graf berbobot pada pemdolen rute lalu lintas kota A.

Untuk setiap unit layanan kesehatan, terdapat data mengenai informasi nama unit layanan kesehatan dan ketersediaan unit

gawat darurat yang dimuat dalam tabel berikut.

PointID	Nama Unit Layanan Kesehatan	Ketersediaan UGD
U1	Puskesmas A	0
U7	Rumah Sakit B	3
U10	Rumah Sakit C	5

Tabel 4. Data unit layanan kesehatan pada kota A.

Berikut merupakan pseudocode algoritma Dijkstra menggunakan representasi *adjajency list* yang sudah dimodifikasi untuk menyelesaikan persoalan ini:

```
function Dijkstra(Graph, src)
  dist[src] <- 0 //inisialisasi
  create node set Q
  for each node v in Graph do
    if v != src then
      dist[v] <- INFINITY
      prev[v] <- UNDEFINED //predesesor v
    Q.add_with_priority(v,dist[v])
  endfor
  while Q is not empy do
    u <- Q.extract_min() //hilangkan min
    for each neighbor v still in Q do
       alt <- dist[u] + length(u,v)
       if alt<dist[v] then
         dist[v] <- alt
         prev[v] <- u
         Q.derease_priority(v,alt)
     endfor
  endwhile
  return dist[], prev[]
```

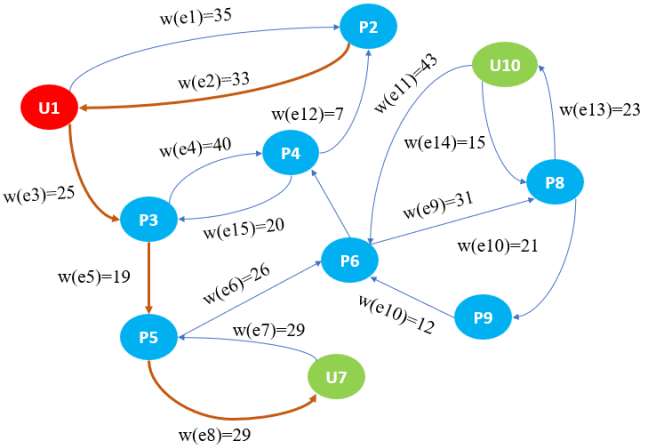
Hasil pencarian rute terefisien menggunakan algoritma tersebut adalah sebagai berikut:

```
Path (1 —> 2): Minimum cost = 35, Route = [1, 2]
Path (1 —> 3): Minimum cost = 25, Route = [1, 3]
Path (1 —> 4): Minimum cost = 65, Route = [1, 3, 4]
Path (1 —> 5): Minimum cost = 44, Route = [1, 3, 5]
Path (1 —> 6): Minimum cost = 70, Route = [1, 3, 5, 6]
Path (1 —> 7): Minimum cost = 73, Route = [1, 3, 5, 7]
Path (1 —> 8): Minimum cost = 101, Route = [1, 3, 5, 6, 8]
Path (1 —> 9): Minimum cost = 122, Route = [1, 3, 5, 6, 8, 9]
Path (1 —> 10): Minimum cost = 124, Route = [1, 3, 5, 6, 8,
10]
Path (2 —> 1): Minimum cost = 33, Route = [2, 1]
Path (2 —> 3): Minimum cost = 58, Route = [2, 1, 3]
Path (2 —> 4): Minimum cost = 98, Route = [2, 1, 3, 4]
Path (2 —> 5): Minimum cost = 77, Route = [2, 1, 3, 5]
Path (2 —> 6): Minimum cost = 103, Route = [2, 1, 3, 5, 6]
Path (2 —> 7): Minimum cost = 106, Route = [2, 1, 3, 5, 7]
Path (2 —> 8): Minimum cost = 134, Route = [2, 1, 3, 5, 6, 8]
Path (2 —> 9): Minimum cost = 155, Route = [2, 1, 3, 5, 6, 8,
9]
Path (2 —> 10): Minimum cost = 157, Route = [2, 1, 3, 5, 6, 8,
10]
Path (3 —> 1): Minimum cost = 80, Route = [3, 4, 2, 1]
Path (3 —> 2): Minimum cost = 47, Route = [3, 4, 2]
Path (3 —> 4): Minimum cost = 40, Route = [3, 4]
Path (3 —> 5): Minimum cost = 19, Route = [3, 5]
Path (3 —> 6): Minimum cost = 45, Route = [3, 5, 6]
Path (3 —> 7): Minimum cost = 48, Route = [3, 5, 7]
Path (3 —> 8): Minimum cost = 76, Route = [3, 5, 6, 8]
Path (3 —> 9): Minimum cost = 97, Route = [3, 5, 6, 8, 9]
Path (3 —> 10): Minimum cost = 99, Route = [3, 5, 6, 8, 10]
Path (4 —> 1): Minimum cost = 40, Route = [4, 2, 1]
Path (4 —> 2): Minimum cost = 7, Route = [4, 2]
Path (4 —> 3): Minimum cost = 20, Route = [4, 3]
Path (4 —> 5): Minimum cost = 39, Route = [4, 3, 5]
Path (4 —> 6): Minimum cost = 65, Route = [4, 3, 5, 6]
Path (4 —> 7): Minimum cost = 68, Route = [4, 3, 5, 7]
Path (4 —> 8): Minimum cost = 96, Route = [4, 3, 5, 6, 8]
Path (4 —> 9): Minimum cost = 117, Route = [4, 3, 5, 6, 8, 9]
Path (4 —> 10): Minimum cost = 119, Route = [4, 3, 5, 6, 8,
10]
Path (5 —> 6): Minimum cost = 26, Route = [5, 6]
Path (5 —> 7): Minimum cost = 29, Route = [5, 7]
Path (5 —> 8): Minimum cost = 57, Route = [5, 6, 8]
Path (5 —> 9): Minimum cost = 78, Route = [5, 6, 8, 9]
Path (5 —> 10): Minimum cost = 80, Route = [5, 6, 8, 10]
Path (6 —> 8): Minimum cost = 31, Route = [6, 8]
Path (6 —> 9): Minimum cost = 52, Route = [6, 8, 9]
Path (6 —> 10): Minimum cost = 54, Route = [6, 8, 10]
Path (7 —> 5): Minimum cost = 29, Route = [7, 5]
Path (7 —> 6): Minimum cost = 55, Route = [7, 5, 6]
Path (7 —> 8): Minimum cost = 86, Route = [7, 5, 6, 8]
Path (7 —> 9): Minimum cost = 107, Route = [7, 5, 6, 8, 9]
Path (7 —> 10): Minimum cost = 109, Route = [7, 5, 6, 8, 10]
Path (8 —> 6): Minimum cost = 66, Route = [8, 10, 6]
Path (8 —> 9): Minimum cost = 21, Route = [8, 9]
Path (8 —> 10): Minimum cost = 23, Route = [8, 10]
Path (10 —> 6): Minimum cost = 43, Route = [10, 6]
Path (10 —> 8): Minimum cost = 15, Route = [10, 8]
Path (10 —> 9): Minimum cost = 36, Route = [10, 8, 9]
```

Bila ingin dilakukan pencarian rute terefisien untuk menuju ke unit layanan kesehatan dari pengguna yang berada di titik P2, maka dari rute-rute berikut akan dipilih rute dengan cost minimal menuju ke unit layanan kesehatan dengan unit gawat darurat yang masih tersedia. Pilihan rute yang ada adalah sebagai berikut:

- Path (2 —> 1): Minimum cost = 33, Route = [2, 1]
- Path (2 —> 7): Minimum cost = 106, Route = [2, 1, 3, 5, 7]
- Path (2 —> 10): Minimum cost = 157, Route = [2, 1, 3, 5, 6, 8, 10]

Dari ketiga rute tersebut, yang terefisien dari segi waktu adalah rute menuju U1. Namun, setelah dilakukan pengecekan pada *database*, ternyata tidak terdapat unit gawat darurat yang tersedia pada unit layanan kesehatan U1. Maka, akan dicek ke alternatif selanjutnya yaitu rute menuju U7. Karena rumah sakit B yang ditandai dengan simpul U7 masih memilihi unit gawat darurat yang kosong, maka rute menuju U7 melewati simpul-simpul P2,P1,P3,P5, dan P7 dipilih.



Gambar 11. Rute terefisien menuju layanan kesehatan dari P2 ke U7
Sumber: Dokumentasi pribadi

D. Potensi Penerapan dan Pengembangan Sistem

Sistem ini memiliki potensi untuk dapat diterapkan pada pengembangan kota pintar dan digunakan menggunakan aplikasi pada gawai. Bila terjadi keadaan gawat darurat, pengguna dapat langsung mengakses aplikasari dan aplikasi akan otomatis memroses data menggunakan GPS dari gawai pengguna. Aplikasi akan memberikan informasi berupa rute perjalanan yang terefisien menuju unit layanan kesehatan.

Ke depannya, sistem seperti ini tidak hanya dapat dimanfaatkan pada persoalanan gawat darurat, tetapi dapat diterapkan pada persoalan rute evakuasi bencana, jalur air, dan persoalan lainnya yang merupakan aspek penting pengembangan kota pintar.

IV. SIMPULAN

Persoalan optimisasi rute untuk mengakses unit layanan kesehatan pada pengembangan kota pintar dapat dimodelkan dengan graf berbobot yang termasuk ke dalam lingkup teori matematika disktrit. Dengan menggunakan teknologi berupa GPS, FCD, dan GIS serta memanfaatkan *map matching algorithm* dan algoritma Dijkstra yang dimodifikasi, sistem optimisasi dapat memberikan rekomendasi rute terefisien dari segi waktu berdasarkan jarak dan kemacetan untuk dapat mengakses unit layanan kesehatan yang menyediakan unit gawat darurat.

Sistem optimasi ini dapat dikembangkan dengan menambahkan fitur-fitur untuk semakin memudahkan pengguna serta dapat diterapkan untuk dapat diakses melalui gawai penduduk sehingga tujuan awal yaitu mengurangi risiko kematian pasien gawat darurat dapat tercapai dengan lebih optimal.

V. UCAPAN TERIMA KASIH

Puji dan syukur penulis ucapkan kepada Allah SWT. atas segala berkah dan rahmat-Nya yang tidak pernah berhenti mengiringi hidup penulis hingga penulis dapat menyelesaikan pembuatan makalah ini dengan baik. Ucapan terima kasih penulis sampaikan kepada orang tua dan keluarga yang telah mendukung penulis dari segi moral maupun material.

Penulis mengucapkan terima kasih sebesar-besarnya kepada seluruh pihak yang telah menyediakan sarana dan prasarana serta memberikan dukungan dalam penyelesaian makalah ini. Terima kasih kepada Bapak Rinaldi Munir selaku dosen pengampu mata kuliah IF2120 Matematika Diskrit kelas K01 dan kepada seluruh tim pengajar mata kuliah IF2120 yang telah memberikan bantuan dan ilmunya kepada penulis dan teman-teman penulis sekalian hingga kami dapat menyelesaikan pembuatan masalah dengan baik dan tepat waktu.

Terima kasih kepada rekan-rekan penulis, terutama segenap mahasiswa program studi Teknik Informastika Institut Teknologi Bandung yang telah memberikan motivasi, dukungan, dan saran kepada penulis selama pengerjaan makalah.

REFERENSI

[1] Limanatara, R., Herjunianto, & A. Roosalina. (2015). Factors Affecting High Mortality at Hospital's Emergency Room. *Jurnal Kedokteran Brawijaya*, vol. 28, 2015, pp. 1–5. Diakses pada 11 Desember 2021 dari https://jkb.ub.ac.id/index.php/jkb/article/viewFile/968/461

[2] Bambani Amri, Asnil. (2021, June 26). *Penuh, pasien meninggal sebelum mendapatkan layanan RS dan ICU*. Diakses pada 11 Desember 2021 dari Kontan: https://nasional.kontan.co.id/news/penuh-pasien-meninggal-sebelum-mendapatkan-layanan-rs-dan-icu

[3] *Dijkstra's Shortest Path Algorithm | Greedy Algo-7*. (2021, September 10). Diakses pada 11 Desember 2021 dari Geeks for Geeks: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

[4] E.C. Navone. (2020, September 28). *Dijkstra's Shortest Path Algorithm - A Detailed and Visual Introduction*. Diakses pada 11 Desember 2021 dari freeCodeCamp: https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/

[5] *Dijkstra's Application.* (n.d.). Diakses pada 11 Desember 2021 dari CSL Michigan Technological University Edu: http://www.csl.mtu.edu/cs2321/www/newLectures/30_More_Dijkstra.htm

[6] Shen, J., & Ban, Y. (2016.). Route Choice of the Shortest Time Based on Floating Car Data. *Journal of Sensors*, vol. 2016, pp 1–12. Diakses pada 14 Desember 2021 dari https://www.hindawi.com/journals/js/2016/7041653/

[7] *Single-Source Shortest Paths – Djikstra's Algorithm.* (n.d.). Diakses pada 14 Desember 2021 dari Techie Delight: https://www.techiedelight.com/single-source-shortest-paths-dijkstras-algorithm/

[8] Charite, Daniel, (n.d.). *Smart Cities on the rise*. Diakses pada 14 Desember 2021 dari https://www2.deloitte.com/nl/nl/pages/public-sector/articles/smart-cities-on-the-rise.html

[9] Munir, Rinaldi. (n.d.). *Graf (Bag. 1)* . Diakses pada 14 Desember 2021 dari *https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf*

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Tangerang Selatan, 14 Desember 2020

Addin Nabilal Huda
13520045