Pencarian Pengendara Terdekat pada Aplikasi UBER dengan Pendekatan Branch and Bound

Alfonsus Raditya Arsadjaja / 13514088
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
alfonsusradityaarsadjaja@students.itb.ac.id

***Abstrak*—Makalah ini akan membahas mengenai aplikasi Uber, yaitu sebuah transportasi umum yang dipesan secara online. Aplikasi ini mem-*booming* pada akhir-akhir ini dengan mengalahkan beberapa transportasi umum lainnya, seperti taksi biasa. Makalah ini akan membahas mengenai algoritma yang digunakan dalam Uber dalam menentukan pengendara yang akan dipilih saat seseorang memesan sebuah "taksi" online tersebut.**

Keywords—A*, Branch-and-Bound, Closest Pair, Radius, Uber

I. PENDAHULUAN

Pada sekarang ini, jumlah kendaraan pribadi semakin bertambah, terutama di kota-kota besar di dunia. Di Indonesia saja, Jakarta, Bogor, dan Bandung sudah terkena imbasnya. Tentu saja, kemacetan sudah tidak terhindarkan lagi, apalagi saat liburan. Kemacetan yang luar biasa ini membuat banyak orang menjadi malas untuk berpergian, terutama untuk orang yang tidak mempunyai kendaraan pribadi. Angkutan umum pun terkadang masih belum dapat dijamin keamanannya, dan harus berdesak-desakan saat menaikinya.

Gambar 1.1 Kemacetan di Kota Bandung
Sumber : http://img2.bisnis.com/, diakses 7 Mei 2016 pukul 19:46

Karena itu, diperlukan suatu transportasi umum yang mudah dipesan dan nyaman dinaiki, dan keamanannya juga terjamin. Beberapa waktu terakhir ini, sudah mulai terkenal beberapa transportasi umum tersebut. Di Indonesia saja ada beberapa, seperti *Go-Jek*, *Grab*, dan *Uber* yang menyediakan fasilitas tersebut. Mereka semakin diminati akhir-akhir ini, bahkan dengan cepat mendahului pesaing-pesaingnya, seperti taksi konvensional, bajaj, dan busway.

Pada makalah ini, kita akan membahas satu saja secara spesifik, yaitu transportasi umum *Uber*. Apa yang dapat membuat mereka cepat berkembang? Salah satunya adalah UI yang simpel dan pelayanan yang mudah dan cepat, tidak ribet dalam menggunakannya. Tentunya dalam mendapatkan pelayanan yang cepat, terdapat beberapa algoritma pencarian didalamnya.

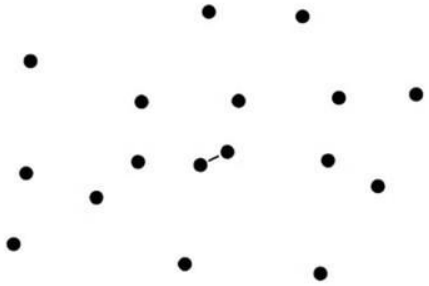
Gambar 1.1 Logo Uber
Sumber: http://www.sketchappsources.com/free-source/1805-new-uber-logo-vector-sketch-freebie-resource.html diakses 7 Mei 2016 pukul 22:23

Makalah ini akan membahas pencarian pengendara terdekat pada Uber, dengan menggunakan salah satu algoritma Branch and Bound, yaitu A-Star (A*) algorithm dan Closest Pair. Mari kita bahas satu-satu.

II. TEORI DASAR

A. *Closest Pair Problem*

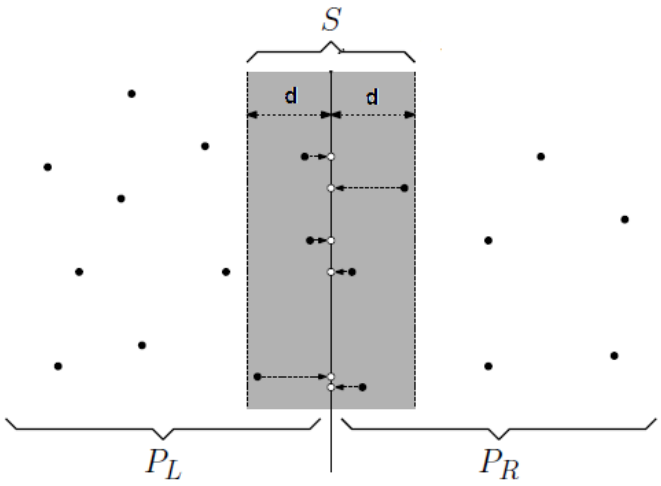
Permasalahan pasangan terdekat (*Closest Pair Problem*) adalah suatu permasalahan dimana terdapat banyak titik pada suatu bidang (bisa 1D, 2D, 3D, maupun banyak dimensi), dan kita harus dapat menemukan 2 titik yang memiliki jarak paling minimal. Jarak dalam hal ini adalah jarak vektor yang berupa garis lurus yang mengubungkan 2 titik tersebut.



Gambar 2.1 Closest Pair Problem
Sumber : http://cdni.wired.co.uk/455x303/s_v/Untitled-1_455x303_251.jpg, diakses 7 Mei 2016 pukul 19:37

Ada beberapa algoritma yang memiliki kompleksitas polynomial yang dapat menyelesaikan persoalan ini. Ada 2 alternatif solusi yang digunakan untuk menyelesaikan soal ini. Yang pertama adalah dengan menggunakan *brute force*, dimana setiap 2 pasang titik dicek jaraknya, dan diambil yang paling minimal. Kompleksitas : $O(N^2)$.

Solusi yang kedua adalah menggunakan pendekatan *divide-and-conquer* dimana proses *divide*nya akan membagi titik-titik menjadi dua bagian yang sama besar secara rekursif, dan menggabungkannya kembali dengan cara : membandingkan closest pair bagian kiri, closest pair bagian kanan, dan closest pair antara kiri dan kanan. Algoritma ini memiliki kompleksitas O(N log N).



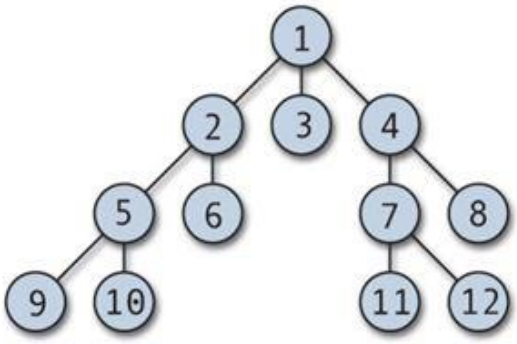
Gambar 2.2 Closest Pair Prolem, algoritma Divide-and-Conquer
Sumber : http://d2dskowxfbo68o.cloudfront.net/, diakses 8 Mei 2016 pukul 1:03

B. Algoritma Breadth First Search (Pencarian Melebar)

Algoritma ini adalah suatu algoritma untuk mencari jarak terdekat antara dua titik pada suatu graf dengan menggunakan pendekatan secara melebar terlebih dahulu. Algoritmanya adalah sebagai berikut:

1. Dimulai dari simpul v
2. Mengambil urutan terdepan pada "antrian"
3. Jika simpul tersebut merupakan solusi, maka stop. Apabila simpul bukan solusi, maka akan mengekspan simpul tersebut dengan simpul-simpul yang belum pernah diekspan, sambil memasukkan simpul-simpul hasil ekspannya kedalam antrian
4. Menghapus simpul tersebut dari antrian
5. Jika masih ada antrian, ulangi langkah kedua. Jika tidak ada, maka tidak ada solusi.
6. Selesai.

Berikut ini adalah contoh bagaimana urutan ekspan suatu simpul menggunakan algoritma BFS.



Gambar 2.3 Urutan pencarian menggunakan BFS
Sumber: https://onbuble.wordpress.com, diakses 8 Mei 2016 pukul 1:08

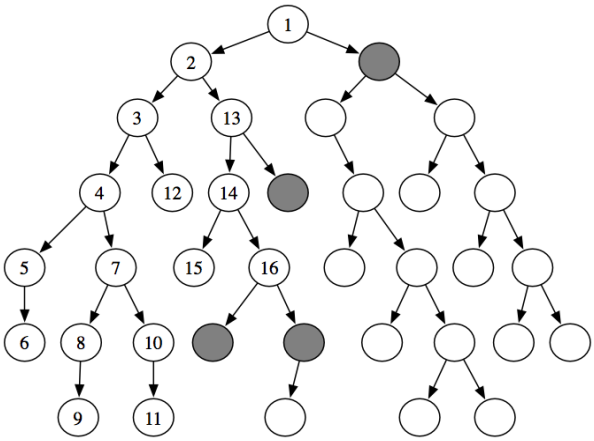
Algoritma ini biasanya digunakan untuk mencari jarak terdekat suatu simpul dengan simpul yang lain. Akan tetapi, ini hanya berhasil apabila semua busurnya tidak memiliki bobot (*unweighted graph*)

C. Algoritma Depth First Search (Pencarian Mendalam)

Algoritma ini adalah suatu algoritma untuk mencari jarak terdekat antara dua titik pada suatu graf dengan menggunakan pendekatan secara mendalam terlebih dahulu. Algoritmanya adalah sebagai berikut:

1. Dimulai dari simpul akar, anggap saja simpul v
2. Kunjungi simpul yang bersisian dengan v, anggap saja w
3. Jika w sudah pernah dikunjungi, maka stop dan melakukan *backtrack* (dilakukan secara rekursif). Jika tidak, ulangi langkah 2 untuk melakukan DFS untuk simpul w.
4. Pencarian selesai apabila semua simpul sudah terkunjungi.

Kira-kira urutannya adalah sebagai berikut:



Gambar 2.4 Urutan Pencarian menggunakan DFS
Sumber: http://artint.info/html/ArtInt_53.html, diakses 8 Mei 2016 pukul 2:18

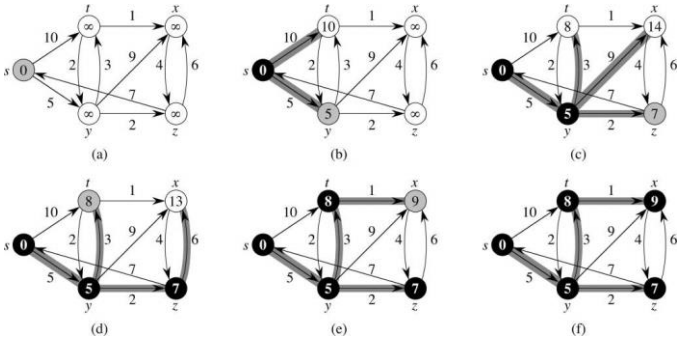
Keuntungan dari algoritma DFS ini adalah lebih hemat memori (untuk kebanyakan kasus) dibandingkan dengan BFS dan biasanya dipakai untuk pencarian dalam struktur data *tree*. BFS membutuhkan memori sebesar banyaknya *vertex* yang disimpan, sedangkan DFS hanya sebanyak yang "saat itu diekspan" sehingga pada kasus rata-rata hanya membutuhkan memori sebesar log(n) dimana n adalah banyaknya node yang diekspan.

D. Algoritma Dijkstra

Algoritma Dijkstra adalah suatu algoritma yang dikembangkan oleh Edsger Wybe Dijkstra pada tahun 1959 untuk mencari jarak terdekat dari suatu titik ke titik-titik lainnya, pada suatu graf. Perbedaan antara algoritma ini dengan algoritma BFS adalah algoritma ini dapat digunakan pada graf berbobot, yang tidak dapat dicakup oleh BFS (BFS hanya dapat mencari "banyak busur" terdekat pada suatu graf, alias graf tidak berbobot. Formulanya : f(n) = g(n) dimana g(n) adalah jarak terdekat dari sumber sampai titik tersebut. Algoritmanya adalah sebagai berikut:

1. Dimulai dari simpul awal (source)
2. Mencari dari simpul-simpul yang akan diekspan, manakah yang belum pernah diekspan dan memiliki jarak sementara paling minimum. Jika sudah tidak ada yang bisa diekspan, maka stop dan mengeluarkan pesan bahwa tidak ada jalan dari simpul awal ke simpul tujuan. Selain itu, lanjut ke langkah 3.
3. Anggap saja simpul yang paling minimum ini adalah simpul v.
4. Jika simpul v merupakan tujuan, stop. Selain itu, lanjut ke langkah 5.
5. Ekspan simpul v dan meminimumkan jarak antara simpul yang diekspan, dengan jarak antara v dengan simpul tersebut ditambah dengan jarak *source* ke v.
6. Ulangi langkah 2.

Kira-kira simulasinya adalah sebagai berikut:



Gambar 2.5 Algoritma Dijkstra
Sumber:
https://www.cs.indiana.edu/~achauhan/Teaching/B403/LectureNotes/images/10-dijkstra.jpg, diakses 8 Mei 2016 pukul 3:14

Algoritma ini cukup efektif untuk menentukan jarak terpendek dari suatu simpul ke simpul-simpul lainnya. Algoritma termangkus yang bisa didapatkan untuk persoalan ini adalah O(|E| + V log V).

E. Pendekatan Heuristik (Heuristic Approach)

Pendekatan Heuristik merupakan suatu strategi/pendekatan terhadap sebuah problem yang memakai suatu metode yang belum dijamin akan paling optimal, akan tetapi setidaknya mendekati optimal. Teknik ini dipakai banyak untuk memprediksi apakah kita menuju suatu solusi yang lebih optimal / tidak.

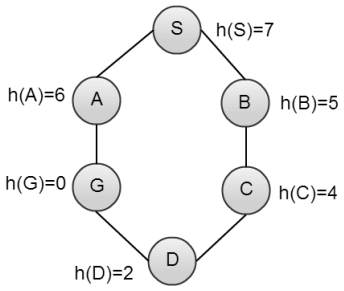
Tidak ada langkah pasti yang dibutuhkan dalam menentukan heuristik. Tetapi suatu hal yang pasti adalah, suatu heuristik harus *admissible* dan tidak boleh merubah solusi yang optimal menjadi tidak optimal.

Salah satu contoh heuristik yang dipakai dalam suatu graf adalah jarak (vektor) antara 2 titik sebagai jarak minimum antara 2 titik tersebut (jarak antara 2 titik tidak mungkin lebih kecil dari garis lurus yang menghubungkan 2 titik tersebut).

F. Algoritma Greedy Best First Search

Algoritma Greedy Best First Search merupakan algoritma yang memakai 100% pendekatan heuristic untuk memecahkan suatu masalah. Dalam teori graf, algoritma ini digunakan juga untuk mencari shortest path seperti algoritma *Dijkstra*, akan tetapi hanya memakai info heuristiknya saja. Formulanya : f(n) = h(n) dimana h(n) merupakan jarak estimasi dari suatu titik ke titik tujuan.

Langkah-langkah dalam menentukan jalan yang harus ditempuh hampir sama seperti Dijkstra, hanya parameternya diubah dari jarak antara titik awal dengan titik tersebut, menjadi jarak estimasi antara titik tersebut dengan titik tujuan. (g(n) → h(n)).



Gambar 2.6 Algoritma Greedy Best First Search, dengan simpul awal adalah S dan simpul akhir adalah G.
Sumber : http://i.stack.imgur.com/hVDkh.png, diakses 8 Mei 2016 pukul 3:52

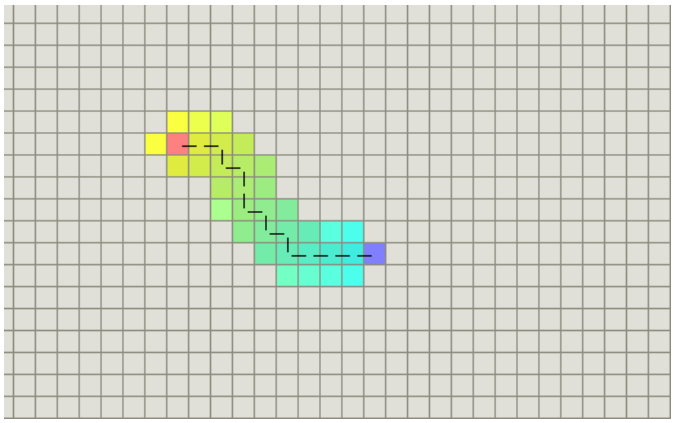
Pada gambar 2.6 diatas, h(n) menunjukkan jarak estimasi antara titik n dengan titik tujuan. Path yang diambil pada gambar diatas adalah S-B-C-D-G.

Dari gambar diatas dapat dilihat bahwa lintasan yang diambil belum tentu optimal karena hanya menggunakan estimasi ke titik tujuan. Setelah ini kita akan melihat bagaimana Dijkstra dan pendekatan heuristik digabung untuk menghasilkan algoritma yang mangkus dan optimal.

G. Algoritma A-Star (A)*

Algoritma A-Star merupakan algoritma yang menggabungkan Dijkstra dan heuristik agar didapatkan algoritma yang optimal. Pada setiap pencarian, formula yang dipakai adalah formula f(n) = g(n) + h(n) dimana g(n) adalah jarak dari titik asal ke titik sekarang, dan h(n) merupakan perkiraan / estimasi jarak ke titik hasil.

Langkah-langkah algoritmanya hampir sama dengan Dijkstra dan Greedy Best First Search, perbedaannya adalah fungsi pembandingnya adalah g(n) dan h(n) sekaligus, menggabungkan jarak antara titik awal dengan titik n, ditambah dengan jarak estimasi antara titik n dengan titik tujuan.



Gambar 2.7 Pencarian menggunakan A-Star
Sumber : http://theory.stanford.edu/~amitp/game-programming/a-star/a-star.png, diakses 7 Mei 2016 pukul 22:30

Untuk menghasilkan hasil yang optimal, h(n) harus bagus, dengan kata lain heuristic yang dipakai bersifat admissible / dapat diterima. Heuristik yang dapat diterima adalah ketika jarak estimasi lebih kecil atau sama dengan jarak aslinya yang akan ditempuh, sehingga tidak menyebabkan perhitungan jalur yang salah / tidak optimal.

Jika h(n) bersifat *admissible* dan memiliki nilai yang bagus, tentunya akan lebih mangkus dari Dijkstra dan tetap menghasilkan algoritma yang optimal. *Branch and Bound* disini terdapat pada bound untuk h(n). Bound disini adalah lower bound, dimana estimasi jarak total yang ditempuh tidak mungkin lebih kecil dari boundnya, sehingga kita tetap akan mendapatkan hasil yang optimal dengan ekspektasi ruang pencarian (simpul yang diekspan) yang lebih kecil.

Algoritma ini memiliki kompleksitas sebesar O(E) untuk kasus rata-rata, lebih cepat dibandingkan Dijkstra biasa, yang memiliki kompleksitas O(|E| + V log V) dimana E adalah jumlah busur dan V adalah jumlah titik / simpul pada graf tersebut.

H. Uber

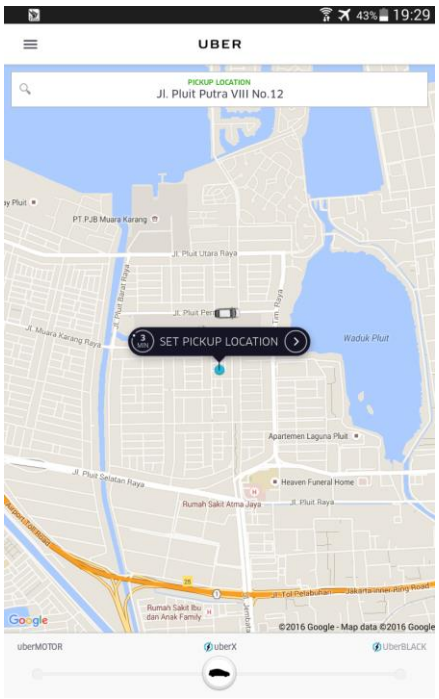
Uber adalah sebuah aplikasi yang dikembangkan oleh Travis Kalanick pada bulan Maret 2009 dalam tujuan untuk membuat sebuah transportasi umum seperti taksi yang mudah, nyaman, dan cepat. Uber menyediakan layanannya dengan cara menyediakan aplikasi dan dalam aplikasi tersebut, pengguna hanya perlu memilih tempat asal dan tempat tujuan, dan secara otomatis pengendara yang paling dekat dengan tempat asal pengguna akan dipilih dan menjadi pengendara untuk sang pengguna.

Ada juga aplikasi lain dengan jenis yang sama (transportasi umum daring), beberapa diantaranya adalah *Go-jek*, *Go-Car*, *Grab-Bike*, *Grab-Car* yang cukup terkenal di kalangan orang-orang di Indonesia.

III. PENERAPAN ALGORITMA

Setelah mengetahui apa saja bidang yang terkait dengan makalah ini, kita akan membahas mengenai bagaimana pengaplikasiannya pada perangkat lunak tersebut. Awalnya, saat pengguna akan membuka aplikasi untuk memesan, maka akan muncul layar untuk memasukkan tempat awal dan tempat tujuan. Tempat awal dan tempat tujuan dari pengguna dapat dimasukkan secara manual (input alamatnya), maupun mengambil location berdasarkan *pin* (sudah disediakan oleh aplikasinya) yang bertujuan untuk mengambil lokasi secara lebih akurat.

Terkadang alamat pada peta tidak begitu akurat sehingga mayoritas orang memilih untuk menginput langsung berdasarkan pin (untuk titik awal). Sedangkan untuk tujuan lebih banyak yang memilih menuliskan alamatnya, karena untuk lokasi tujuan dapat diatur oleh percakapan antara pengguna dan pengendara.



Gambar 3.1 Layar meminta pengguna untuk menginput lokasi awal
Sumber : dokumen pribadi

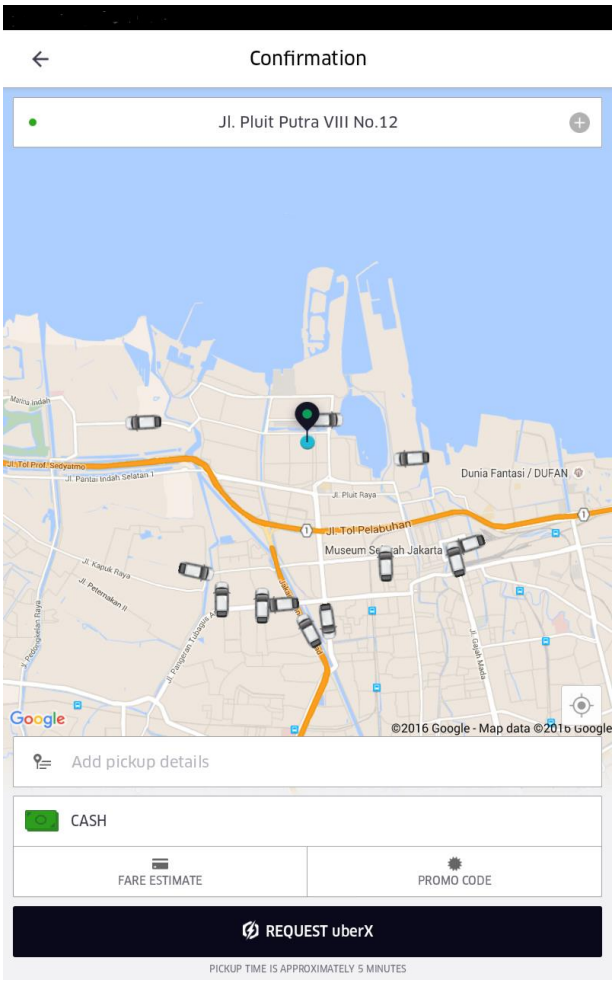
Saat pengguna akan memasukkannya ke layar, dapat dilihat ada beberapa mobil yang berada di sekitar pengguna tersebut, yang menandakan mobil-mobil terdekatnya. Cara untuk mengetahui mobil mana saja yang berada di sekitar pengguna tersebut dapat diimplementasikan dengan menggunakan gabungan *clustering* (tidak akan dibahas pada makalah ini), closest pair, dan branch and bound (dalam hal ini A-Star Algorithm).

A. Mencari Pengendara dalam Lingkup yang Sama

Proses ini diawali dengan mencari pengendara (*pengendara*) yang ada di sekitar pelanggan dengan menggunakan clustering (disimpan pada database, tidak dibahas pada makalah ini) untuk mempersempit pencarian pengendara. Pengendara yang dipilih untuk diseleksi selanjutnya adalah yang memiliki "daerah" yang sama dengan pelanggan. Hal ini sangat mempercepat proses pencarian, karena tidak harus mencari keseluruhan data pengendara (yang banyak diluar daerah pengguna yang bersifat *useless*.

Lalu, dengan menggunakan algoritma *brute force*, mencari pengendara yang berada pada suatu radius tertentu dari sang pelanggan. Dari pengamatan penulis, radius berada sekitar 0-3 km dari pengguna, jarak yang cukup optimal untuk mendapatkan mobil-mobil yang jumlahnya cukup, tetapi tidak terlalu jauh dari pengguna.

Penyelesaian permasalahan ini sebenarnya mirip seperti closest pair problem, hanya saja salah satu titik sudah diketahui sehingga problem direduksi menjadi jarak terdekat ke suatu titik (anggap titik pengguna merupakan titik origin sehingga penyelesaian menjadi lebih mudah).



Gambar 3.2 Pengendara yang berada pada lingkup yang sama dengan pengguna
Sumber : dokumen pribadi

Kira-kira algoritma untuk menentukan pengendara yang akan muncul di layar adalah sebagai berikut:

```
function get_set_pengendara() → set

set_pengendara = {}
foreach(pengendara di lingkup tersebut) do
  if(jarak(pengendara) <= radius) then
    set_pengendara ← set_pengendara ∪ pengendara
  endif
endfor

→ set_pengendara
```

B. Mencari Pengendara yang akan dipilih

Setelah mendapatkan beberapa pengendara tersebut, maka akan dicari pengendara yang paling dekat dengan pengguna. Karena pelanggan maupun pengendara dapat berubah tempat dalam hitungan detik saja, maka dibutuhkan algoritma yang mangkus untuk mendapatkan hasilnya secepat mungkin. Hal ini tentunya tidak dapat diselesaikan apabila algoitmanya tidak mangkus (dapat menyebabkan titik tempat pengguna akan dijemput bergeser dan menyebabkan salah tempat). Hal ini dapat diselesaikan dengan menggunakan salah satu algoritma Branch and Bound, yaitu A*.

Pseudocode untuk menentukan jarak setiap pengendara dengan pelanggan adalah sebagai berikut.

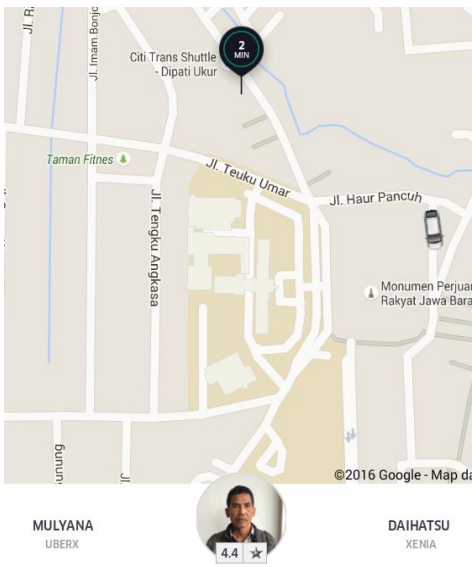
```
function A_star → integer
precompute heuristik untuk setiap n{h(n)}
masukkan titik awal pelanggan ke queue

while(queue belum kosong atau belum ketemu titik tujuan)
do
  v ← elemen pada queue dengan f(v) memiliki nilai
terkecil
  queue = queue - v
  if(v belum pernah dikunjungi) then
    foreach(setiap node yang terhubung dengan v) do
      if(g[node] < g[v] + jarak(v,node)) then
        g[node] ← g[v] + jarak(v,node)
        queue = queue ∪ node
      endif
    endfor
  endif
endwhile

→ g[pengendara]
```

Algoritma A* diatas, seperti sudah dibahas pada Bab II, diperlukan suatu heuristik yang bagus untuk mendapatkan hasil secepat mungkin. Akan tetapi, heuristik pada pencarian di Uber ini tidak harus selalu optimal, asal mendekati optimal. Hal ini disebabkan ada pengendara yang jaraknya juga tidak jauh dari pengendara yang memiliki jarak tempuh paling kecil, sehingga walaupun jaraknya tidak optimal, perbedaan waktu yang dibutuhkan untuk mencapai pengguna tidak terlalu signifikan (tidak sampai beberapa menit).

Setelah mendapatkan jarak tiap pengendara dengan pengguna, maka menggunakan algoritma pencarian minimum, bisa didapatkan pengendara yang mempunyai jarak minimum dengan pengguna. Algoritma ini cukup mangkus, yaitu memiliki kompleksitas O(N).



Gambar 3.3 Pengendara terdekat telah ditemukan
Sumber : dokumen pribadi

IV. PERKEMBANGAN LEBIH LANJUT

Hal diatas dapat diaplikasikan dengan kompleksitas dan waktu yang cukup baik, dan cukup baik dalam pencarian pengendara. Akan tetapi, tentunya tidak cukup sampai disitu, Algoritma yang dipakai oleh *Uber* sekarang ini dalam menentukan semua itu sudah jauh lebih *advanced* dan lebih mangkus dibandingkan dengan algoritma A* dan closest pair yang dipakai diatas.

Banyak hal yang menunjang agar aplikasi / perangkat lunak tersebut dapat berjalan dengan jauh lebih optimal, yang tidak dibahas di mata kuliah Strategi Algoritma, seperti basisdata pengguna dan pengendara, *Google Maps API*, User Interface dan pengalokasian memori yang sangat rumit. Cara menghitung jarak terdekat antar pengguna dan pengendara adalah dengan menggunakan fitur yang telah disediakan oleh *Google Maps API*, dimana algoritmanya sudah disediakan oleh mereka. Pencarian jarak terdekat dengan *google maps* sendiri sebenarnya sudah bukan hanya memakai A* lagi, tetapi juga sudah ada data-data yang disimpan (memakai precompute) apabila sudah pernah ada orang yang meng-query jalan-jalan yang termasuk (sehingga sudah ada list pathnya, tidak perlu menghitung ulang, dan hasilnya bisa jauh lebih cepat ditemukan).

V. KESIMPULAN

Algoritma pencarian pengendara dengan menggunakan algoritma *closest pair* dan *Branch and Bound* (A*) diatas dapat diaplikasikan pada aplikasi Uber, karena algoritmanya yang cukup mangkus. *Closest pair* mangkus dalam hal mencari pengendara dalam radius tertentu, sedangkan *Branch and* Bound mangkus dalam hal mencari jarak terdekat antara suatu pengguna dan pelanggan. Akan tetapi, algoritma ini hanya bisa dijalankan apabila masih dalam skala kecil. Jika sudah dalam skala besar, kecepatannya menjadi lambat, sehingga diperlukan algoritma lain selain pencarian pengendaranya yang lebih mangkus lagi, untuk menjamin kenyamanan pengguna dan tepat sasaran.

Salah satu alternatif yang digunakan untuk mendapatkan *query* yang lebih cepat lagi adalah menggunakan *Google Maps API* yang sudah menggunakan precomputation untuk mendapatkan *shortest pathnya* sehingga *query* dapat diselesaikan dengan lebih cepat lagi. Selain itu ada juga beberapa algoritma yang membuat transfer data menjadi jauh lebih cepat dan mangkus, dibahas di bidang basis data (tidak dibahas di mata kuliah strategi algoritma), dan masih banyak lagi yang tidak dibahas pada makalah ini.

VI. UCAPAN TERIMA KASIH

Penulis menyampaikan syukur kepada Tuhan yang Maha Kuasa, karena makalah ini dapat tersusun dengan baik. Penulis berterima kasih kepada Dr. Ir. Rinaldi Munir, MT. dan Dr. Nur ulva Maulidevi, ST., MT., yang telah membimbing penulis dalam penulisan makalah ini melalui perkuliahan Strategi Algoritma. Penulis juga mengucapkan terima kasih kepada seluruh pihak yang berperan dalam penyusunan makalah ini sehingga penulis dapat menyelesaikan makalah ini tepat waktu.

REFERENSI

[1] Munir, Rinaldi. 2006. *Strategi Algoritma*. Bandung : Penerbit Informatika.

[2] https://www.quora.com/What-is-the-algorithm-used-by-Uber-Ola-Cabs-to-show-nearby-cabs-on-their-application-How-is-the-time-calculated, diakses 7 Mei 2016 pukul 1:58

[3] https://onbuble.wordpress.com, diakses 8 Mei 2016 pukul 1:08

[4] http://eprints.dinus.ac.id/14269/1/slide_13b.pdf, diakses 8 Mei 2016 pukul 1:11

[5] https://eng.uber.com/schemaless-part-one/, diakses 8 Mei 2016 pukul 1:17

[6] http://examples.yourdictionary.com/examples-of-heuristics.html, diakses 8 Mei 2016 pukul 3:47

[7] http://theory.stanford.edu/, diakses 7 Mei 2016 pukul 22:30

[8] https://www.cs.indiana.edu/, diakses 8 Mei 2016 pukul 3:14

[9] https://onbuble.wordpress.com, diakses 8 Mei 2016 pukul 1:08

[10] http://cdni.wired.co.uk, diakses 7 Mei 2016 pukul 19:31

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.

Bandung, 8 Mei 2016

Alfonsus Raditya Arsadjaja
13514088