Penerapan Algoritma Branch and Bound untuk Pencarian Pengendara pada Aplikasi Go-Jek

Taufan Mahaputra / 13515028
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515028@std.stei.itb.ac.id

***Abstract*—Makalah ini akan membahas mengenai aplikasi transportasi online yang saat ini sedang ramai digunakan di Indonesia yaitu Go-jek. Dalam aplikasi ini terdapat beberapa fungsi yang mengharuskan mencari jarak terpedek misalnya untuk menentukan armada yang menjemput dan jalan yang harus dilalui untuk mencapai lokasi tujuan. Makalah ini akan membahas mengenai algoritma yang digunakan dalam Go-Jek dalam menentukan pengendara yang memiliki jarak terdekat dengan pemesean tersebut.**

Keywords—Go-Jek,A*,Branch-and-Bound,Closest Pair

I. PENDAHULUAN

Kebutuhan akan transportasi umum yang murah, nyaman, cepat, mudah, dana man merupakan masalah klasik yang sudah ada di Indonesia dari zaman-ke-zaman. Beberapa transportasi yang sudah ada saat ini misalnya transjakarta dirasa belum mampu untuk memenuhi kebutuhan tersebut, namun di era perkembangan teknologi sekarang yang sangat pesat ini. Berawal dari kebutuhan tersebut, suatu solusi bisa diciptakan. Dengan mengoptimalkan berbagai aspek seperti rute yang digunakan dan ketersediaan kendaraan yang memadai, membuat hal ini sangat cepat.

Gambar 1.1 Ojek *Go-Jek* sedang beroperasi

Sumber : http://www.kompasiana.com/

Beberapa waktu terakhir ini, sudah mulai terkenal beberapa transportasi umum yang dapat memenuhi kebutuhan tersebut. Di Indonesia ada beberapa, seperti *Go-Jek*, *Grab* dan *Uber*. Mereka semakin diminati akhir-akhir ini. *Goj-Jek* adalah salah satu perusahaan yang berusaha mewujudkan suatu konsep untuk memenuhi kebutuhan tersebut.

Dengan munculnya aplikasi seperti *Go-Jek* ini yang biasanya kita memesan ojek secara langsung/tatap muka ataupun menelpon jika kita menyimpan nomor ojek tersebut, sekarang hal tersebut sudah tidak perlu dilakukan lagi. Karena saat kita melakukan pemesanan secara otomatis akan dicarikan pengendara yang terdekat dengan kita melewati aplikasi tersebut.

Makalah ini akan membahas bagaimana aplikasi *Go-Jek* mencari pengendara terdekat, dengan menggunakan salah satu algoritma Branch and Bound yaitu A-Star(A*) algorithm dan Closest Pair.

II. GO-JEK

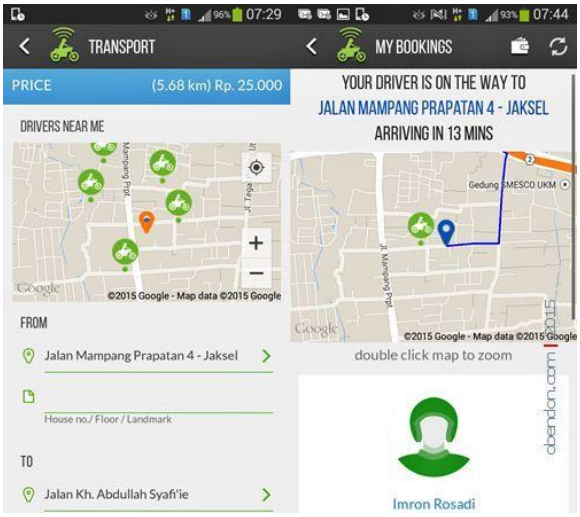
Gambar 2.1 Logo *Go-Jek*
Sumber : http://www.go-jek.com/

A. *Deskripsi*

Go-Jek adalah perusahaan yang memperbaharui transportasi umum terutama motor di Indonesia. *Go-Jek* bermitra dengan para pengendara sepeda motor diberbagai daerah meliputi area JABODETABEK, Bandung, Bali, & Surabaya. *Go-Jek* selain transportasi motor, sekarang *Go-Jek* sudah mengekspansi ke berbagai pasar sehingga muncul layanan-layanan yang lain seperti pengiriman barang, pesan antar makanan, berbelanja, bersih-bersih, ataupun pijat.

Dengan menggunakan aplikasi *Go-Jek* pengguna dapat memesan ojek untuk mengakses ke berbagai layanan yang tersedia. Salah satunya ada fitur untuk ‘Use my location’

sehingga didapat fitur ‘Near Me’ yang mana mengartika ada apa saja disekitar saya. Dan fitur ini digunakan juga untuk mencaari pengendara terdekat di ‘sekitar saya’.



Gambar 2.2 Fitur Drivers Near Me

Sumber : https://obendon.files.wordpress.com

Dengan adanya fitur tersebut, saat pemesanan maka pengendara yang akan menjemput akan sangat cepat Karena lokasinya yang tidak berjauhan dengan posisi kita sekarang.

B. Sejarah

Gojek berdiri pada tahun 2011 oleh seorang pemuda yang sangat kreatif. Pendiri gojek bernama Michaelanglo maron dan Nadiem makarin. Mereka mendirikan sebuah perusahaan yang diberi nama PT Go-jek Indonesia. Perusahaan ini bertujuan untuk menghubungkan ojek dengan penumpang ojek. Mereka melihat para ojek pangkalan hanya menghabiskan waktu seharian dan belum tentu mendapatkan pelanggan. Jadi mereka membuat perusahaan ini, untuk membantu para tukang ojek mendapatkan penumpangnya dengan lebih cepat dan efisien. Sampai sekarang tujuan mereka memang terbukti ampuh. Tukang ojek harus lebih produktif supaya bisa mendapat penghasilan yang lebih banyak.

Para tukang ojek pangkalan tersebut terkadang menunggu 8 sampai 10 jam, tetapi paling hanya mendapatkan 4 sampai 7 orang penumpang saja. Pendiri gojek berinisiatif membuat sesuatu yang berbeda. Gojek ini menggunakan sebuah system yang lebih tertata rapi. Awalnya gojek hanya melayani lewat call center saja, tetapi lambat laun gojek mulai berkembang dan membuat aplikasi gojek. Dengan aplikasi ini, anda bisa memesan secara online, membayar secara kredit dan mengetahui keberadaan driver yang akan menjemput anda. inilah salah satu kelebihan dari gojek dibandingkan dengan ojek pangkalan lainnya.

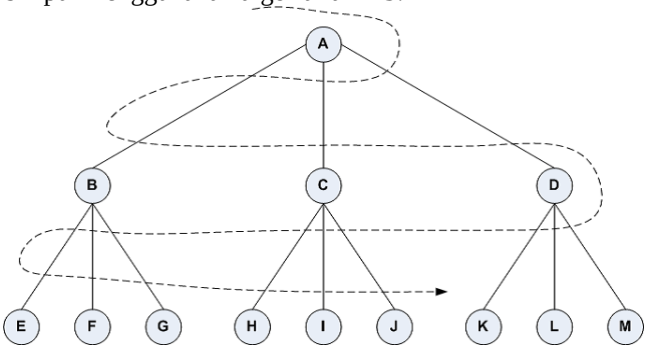
III. LANDASAN TEORI

A. 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: http://yoursknowladge.blogspot.co.id

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*)

B. Algoritma Depth First Search (Pencarian Mendalam)

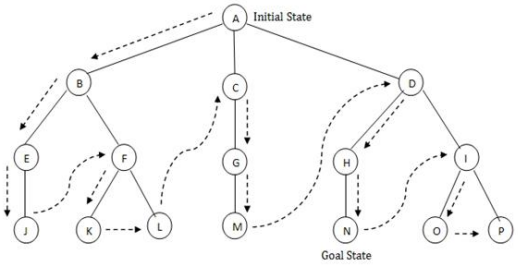
Algoritma ini adalah suatu algoritma untuk mencari jarak terdekat antara dua titik pada suatu graf dengan menggunakan pendekatan secara mendalam terlebih dahulu. Algoritmanya adalah sebagai berikut:

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://aa-miracle.blogspot.com

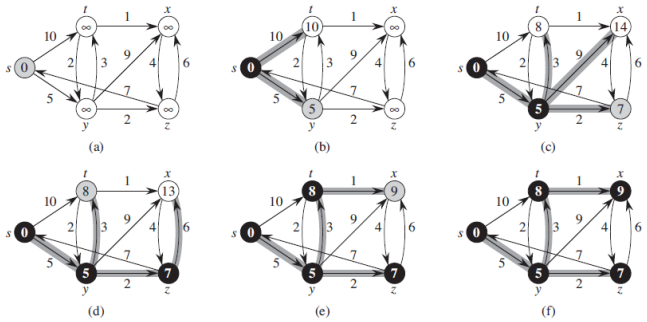
Keuntungan dari algoritma DFS ini adalah lebih hemat memori (untuk kebanyakan kasus) dibandingkan dengan BFS dan biasanya dipakai untuk pencarian dalam struktur data *tree*. BFS membutuhkan memori sebesar banyaknya *vertex* yang disimpan, sedangkan DFS hanya sebanyak yang "saat itu diekspan" sehingga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

C. 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:

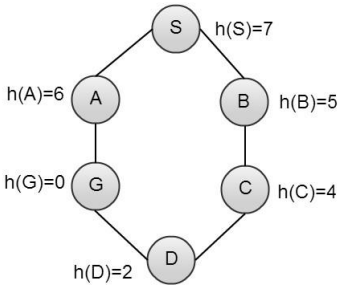


Gambar 2.5 Algoritma Dijkstra
Sumber: http://www.codebytes.in

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).

D. 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

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

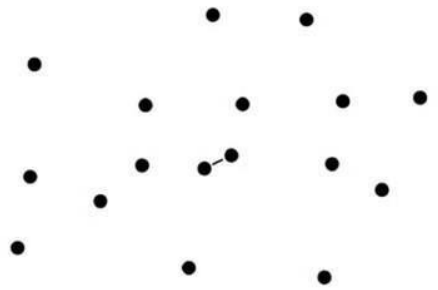
dan pendekatan heuristik digabung untuk menghasilkan algoritma yang mangkus dan optimal.

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. Closest Pair Problem

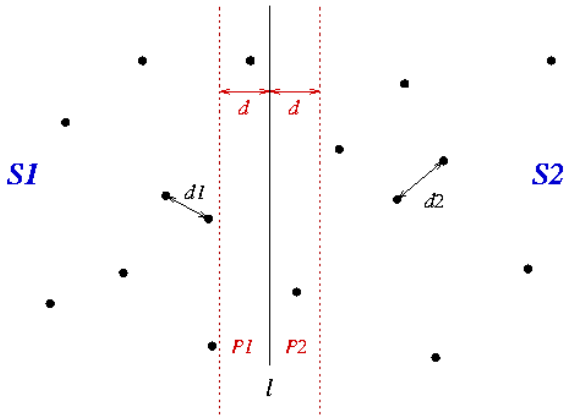
Permasalahan pasangan terdekat (*Closest Pair Problem*) adalah suatu permasalahan dimana terdapat banyak titik pada suatu bidang (bisa 1D, 2D, 3D, maupun banyak dimensi), dan kita harus dapat menemukan 2 titik yang memiliki jarak paling minimal. Jarak dalam hal ini adalah jarak vektor yang berupa garis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

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 : $O(N^2)$.

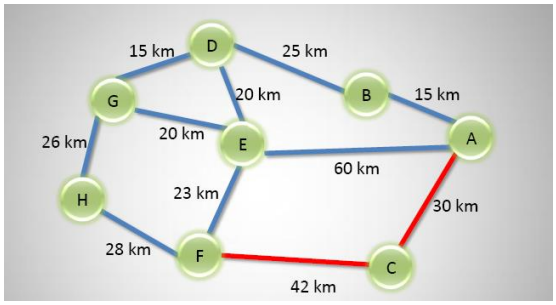
Solusi yang kedua adalah menggunakan pendekatan *divideand-conquer* dimana proses *divide*nya akan membagi titik-titik menjadi dua bagian yang sama besar secara rekursif, dan menggabungkannya kembali dengan cara : membandingkan closest pair bagian kiri, closest pair bagian kanan, dan closest pair antara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



Gambar 2.2 Closest Pair Prolem, algoritma Divide-and-Conquer
Sumber : www.cs.mcgill.ca

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.



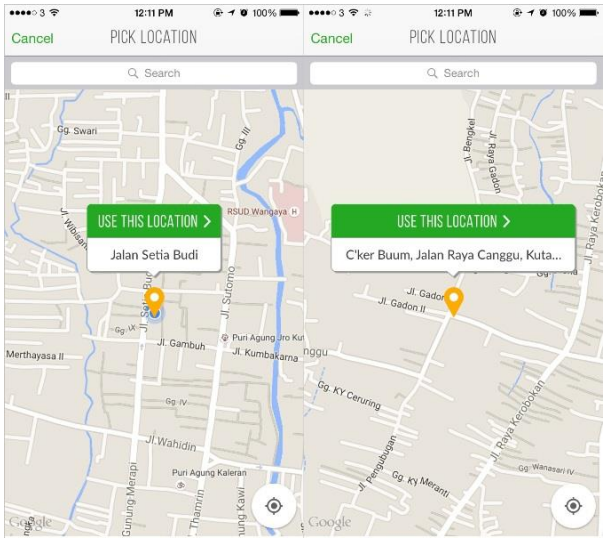
Gambar 2.7 Pencarian menggunakan A-Star
Sumber : http://zafnatpaneyah.blogspot.co.id

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.

IV. PENERAPAN ALGORTIMA

Guna mencari pengemudi terdekat untuk menjemput pengguna pada aplikasi *Go-Jek* dengan cara yang lebh optimal dapat diterapkan algoritma *Closest Pair* dan algortima *Branch and Bound(A*)*. Pertama-tama akan dibagi menjadi 2 tahap, yaitu tahap mencari pengendara dalam lingkup/radius yang sama dan tahap kedua yaitu mencari pengendara yang terdekat untuk dipilih. Sebelum menuju tahap pertama, terlebih dahulu pengguna meng-*set* lokasi untuk penjemputan. Lokasi awal inilah yang dijadikan acuan sebagai titik awal. Yang mana kemudian akan muncul driver-driver yang terdapat disekitar lokasi penjemputan tersebut.



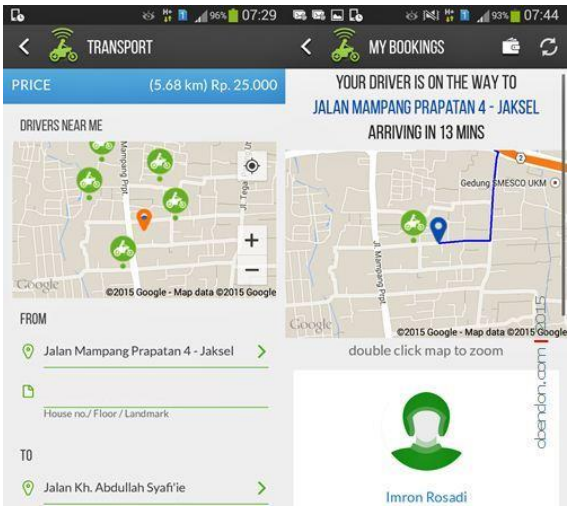
Gambar 4.1 Layar meminat pengguna untuk menginput lokasi awal

Sumber: www.jessicaliani.com

A. *Mencari Pengendara dalam Radius yang Sama*

Pada proses ini diawali dengan mencari pengendara dalam radius yang sama, menurut pengamatan Alfonsus, Teknik Informatika 2014 mengatakan radius berada sekitar 0-3km dari pengguna. Yang mana jarak tersebut terbilang cukup, tidak terlalu jauh dan tidak terlalu dekat untuk mendapatkan motor/mobil yang tersedia.

Penyelesaian permasalahan ini sebenarnya mirip seperti *closest pair problem*, tetapi kasus ini menggambarkan hanya salah satu titik yang sudah diketahui, titik-titik lain merupakan posisi *random* dari motor/mobil pada radius yang sudah ditetapkan.



Gambar 4.2 Pengendara yang berada pada radius yang sama dengan pengguna
Sumber : www.obendon.com

Jadi misal pengguna meng-*set* lokasi di suatu titik, maka titik tersebut akan di ekspansi sebesar radius, kemudian dalam aplikasi *Go-Jek* tiap pengendara terintegrasi dengan GPS yang dapat dilacak lokasinya secara *real-time*. Oleh Karena itu, setelah mendapatkan lingkup yang dituju, algortiima ini akan mencari 'titik-titik' yang merupakan pengendara-pengendara yang berada dalam radius tersebut.

Dengan melakukan iterasi pada perhitungan jarak disetiap pengendara terhadap radius yang ditetapkan, dimasukkan kedalam sebuah list jika jarak pengendara kurang dari sama dengan radius. Sehingga setelah selesai iterasi akan terdapat list yang berisi pengendara-pengendara yang berada dalam radius pengguna.

Kemudian list tersebut lah yang akan dilanjutkan dengan algortima A* dimana akan dikalkulasi juga berdasarkan heuristiknya seperti waktu menuju tempat penjemputan. Karena bisa aja jarak yang terdekat menjadi yang paling lama untuk menjemput dikarenakan macet atau sesuatu terjadi dijalanan ia berada.

B. Mencari Pengendara Terdekat dalam Radius

Kemudian setelah mendapatkan beberapa pengendara yang berada dalam radius dengan pengguna, kemudian dilakukan pemilihan dengan mencari pengendara terdekat dengan pengguna. Kasus ini dapat diselesaikan dengan menggunakan salah satu algoritma *Branch and Bound* yaitu A*.

Gambar 4.3 Pengendara terpilih merupakan pengendara terdekat dengan pengguna
Sumber : www.dribble.com

Seperti Algoritma A* yang diajarkan pada perkuliahan, pertama-tama ditentukan terlebih dahulu suatu heuristic yang bagus agar dapat hasil yang optimal. Kemudian cari hitung kalkulasi tiap pengendara dengan pengguna menggunakan 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.

Setelah mendapatkan perhitungan dari tiap pengendara maka dilanjutkan dengan menggunakan algoritma pencarian minimum, bisa didapatkan pengendara yang mempunyai jarak minimum serta wakt yang paling singkat berdasarkan heuristiknya dengan pengguna. Algoritma ini cukup mangkus, yaitu memiliki kompleksitas O(N).

Pseudocode untuk menghitung formula algoritma A* untuk menentukan jarak tiap pengendara dengan pengguna adalah sebagai berikut.

```
      // A*
1:    initialize the open list
2:    initialize the closed list
3:    put the starting node on the open list (you can leave its f at zero)
-
4:    while the open list is not empty
5:        find the node with the least f on the open list, call it "q"
6:        pop q off the open list
7:        generate q's 8 successors and set their parents to q
8:        for each successor
9:            if successor is the goal, stop the search
10:           successor.g = q.g + distance between successor and q
11:           successor.h = distance from goal to successor
12:           successor.f = successor.g + successor.h
-
13:           if a node with the same position as successor is in the OPEN list \
-                 which has a lower f than successor, skip this successor
14:           if a node with the same position as successor is in the CLOSED list \
-                 which has a lower f than successor, skip this successor
15:           otherwise, add the node to the open list
16:       end
17:       push q on the closed list
18:   end
```

Gambar 4.4 Psedocode Algoritma A*
Sumber : http://web.mit.edu/eranki/www/tutorials/search/

Arti dari pseudocode diatas adalah akan dibuat sebuah *Queue* yang berisi hasil perhitungan formula dengan algortima A*.

V. KESIMPULAN

Algoritma pencarian pengendara terdekat jadi dilakukan secara bertahap, tahap yang pertama yaitu mencari pengendara dalam lingkup yang sama dengan menggunakan algoritma *closest pair* dan *Branch and Bound(A*)* diatas. Karena kemangkusan algortima, algortima *closest pair* digunakan untuk mencari pengendara-pengendara dalam radius tertentu, sedangkan untuk tahap kedua jika sudah didapat beberapa pengendara dalam radius tertentu kemudian dicari yang terpendek terharap pemesan menggunakan algoritma *Branch and Bound(A*)* yang cukup mangkus dalam hal mencari jarak terdekat antara 2 titik. Akan tetapi, algoritma ini cukup magnkus jika dijalankan dalam program skala kecil. Jika dalam skala besar, algortima ini akan berjalan lambat, sehingga diperlukan algoritma lain untuk mencari pengendara yang lebih mangkus dalam skala besar untuk mencapai kebutuhan yang sudah dijelaskan.

Kekurangan algortima-algoritma tersebut terkadang menghasilkan perhitungan yang salah hal itu dikarenakan ada kalanya jika terdapat jalur yang tidak dapat dilewati atau semacamnya dapat membuat hasil perhitungan dengan algortima A* mencari tidak akurat. Maka dari itu diharapkan nantikan akan ada algoritma baru yang lebih mangkus, untuk menangani berbagai persoalan yang terjadi.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa yang telah memampukan penulis untuk menyelesaikan makalah ini dengan baik. Selanjutnya penulis juga mengucapkan terima kasih kepada orang tua penulis serta rekan-rekan di sekitar penulis yang terus memberikan dukungan dan nasihat sehingga menjadi motivasi bagi penulis dalam menyelesaikan kewajiban dalam perkuliahan di ITB.

Penulis berterima kasih pada Dr.Ir. Rinaldi Munir, MT ; Dr. Masayu Leylia Khodra, ST.MT ; Dr.Nur Ulfa Maulidevi, ST., M.Sc., 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.

REFERENCES

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

[2] https://onbuble.wordpress.com, diakses 15 Mei 2017 pukul 20.15

[3] http://go-jek.com, diakses pada 15 Mei pukul 20.15

[4] https://www.quora.com/What-are-technologies-behind-Go-Jek-app#, diakses pada 17 Mei pukul 09.43

[5] https://www.quora.com/What-are-some-thoughts-about-GO-JEK, diakses pada 17 Mei pukul 10.00

[6] http://tipsdaftar.blogspot.com/2015/10/sejarah-berdirinya-gojek-dan-pendiri.html, diakses pada 17 Mei pukul 10.25

[7] http://www.cs.mcgill.ca/~cs251/ClosestPair/ClosestPairDQ.html, diakses pada 17 Mei pukul 11.55

[8] http://www.gojakgojek.com/2016/03/sudah-tahu-cara-kerja-gojek-atau-sistem-kerja-gojek.html, diakses pada 17 Mei 14.20

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, 17 Mei 2017

Taufan Mahaputra 13515028