Optimasi Pemilihan Rute Terpendek Angkutan Umum Sesuai Preferensi Pengguna dengan Algoritma A* Berbasis Google Maps

Hanny Fauzia[#1], Dr. Ir. Rinaldi Munir, M.T.[*2]

[#]*Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung*
Jl. Ganeca no. 10 Bandung
[1]13509042@std.stei.itb.ac.id

[*] *Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung*
Jl. Ganeca no. 10 Bandung
[2]*rinaldi@informatika.org*

Abstrak— **Fenomena kemacetan yang sering terjadi pada kota besar sangat mempengaruhi produktivitas kota tersebut karena waktu penuduk kota untuk bekerja menjadi terbuang di jalan. Salah satu pendekatan yang dapat digunakan untuk mengatasi masalah tersebut adalah dengan mengoptimalkan pemakaian kendaraan umum dengan menyediakan sarana pencarian rute secara otomatis dengan mengutamakan preferensi pengguna kendaraan umum.**

Pada makalah ini dilakukan studi penerapan pencarian rute optimal untuk kasus penggunaan kendaraan umum. Selanjutnya dilakukan studi beberapa penelitian terkait dan dipilih metode yang paling cocok untuk diterapkan pada kasus penggunaan kendaraan umum. Kemudian dilakukan analisis penerapan metode tersebut untuk menghasilkan rute yang optimal sesuai preferensi pengguna agar kenyamanan pemakai kendaraan umum dapat meningkat.

Dari makalah ini telah dikembangkan aplikasi AngkotFinder sebagai implementasi dua macam modifikasi A* dengan preferensi dan didapat bahwa modifikasi A* versi kedua lebih memenuhi preferensi pengguna dibanding versi pertama. Kedua modifikasi tersebut mengakibatkan waktu eksekusi A* meningkat menjadi sekitar 3.5 kali lipat.

Kata Kunci— ***Shortest path, optimal path,* A*, preferensi pengguna, angkutan umum.**

I. PENDAHULUAN

Jumlah penduduk di berbagai daerah di dunia selalu bertambah dan tidak terlihat adanya tanda penurunan [1]. Fenomena ini juga memicu semakin padatnya jalur transportasi yang tersedia, terutama pada kota–kota besar yang pada akhirnya akan menimbulkan kemacetan [2]. Masalah kemacetan ini sangat mempengaruhi produktivitas dari suatu kota karena waktu yang seharusnya bisa dilakukan untuk bekerja justru terbuang di jalan untuk menunggu kendaraan.

Jika dilihat secara seksama, sebenarnya solusi yang paling tepat untuk mengatasi masalah kemacetan adalah dengan memanfaatkan transportasi umum semaksimal mungkin. Penggunaan transportasi umum secara optimal akan menurunkan *space* yang dibutuhkan untuk suatu jalur transportasi dan akan menurunkan kemacetan secara signifikan. Sayangnya, kendaraan pribadi tetap lebih dinikmati secara umum karena trayek dari kendaraan umum bersifat tetap sehingga untuk pergi dari suatu tempat ke tempat lainnya biasanya dibutuhkan pergantian dari satu kendaraan dan kendaraan lain. Selain itu, trayek tersebut biasanya bersifat kompleks dan hanya sebagian orang saja yang mengetahui trayek kendaraan umum yang diperlukan untuk suatu rute perjalanan dari suatu tempat ke tempat lainnya.

Dengan dikembangkannya suatu aplikasi yang bersifat dapat diakses kapan saja dan dimana saja (aplikasi *mobile* via internet) untuk mencari rute kendaraan umum yang dibutuhkan untuk pergi dari suatu tempat ke tempat lainnya, maka diharapkan peminat dari kendaraan umum semakin meningkat. Selain itu, rute kendaraan umum yang dihasilkan sebaiknya dapat bersifat fleksibel, yaitu dapat mengikuti preferensi pengguna agar dapat meningkatkan kepuasan pengguna. Dengan meningkatnya pemakai kendaraan umum, maka diharapkan pula kemacetan yang sering terjadi pada kota-kota besar dapat ditekan semaksimal mungkin.

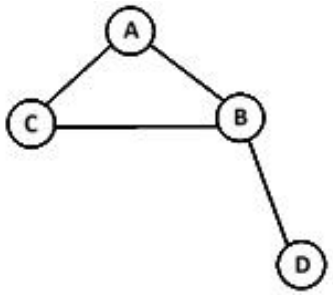
Beberapa pendekatan dengan bermacam-macam algoritma sudah diterapkan sebagai dasar dari aplikasi yang dirancang untuk memecahkan masalah pencarian rute kendaraan umum ini. Beberapa di antaranya adalah algoritma *K-Shortest Path* untuk mencari jalur terpendek rute bus sebanyak K untuk memenuhi preferensi pengguna [3], algoritma Djikstra dalam mencari jalur terpendek rute angkot di kota Bogor [4], dan algoritma A* dalam pembangunan aplikasi pencari rute angkot di kota Jakarta Selatan [5].

Makalah ini membahas kasus pencarian rute tersebut dengan pendekatan algoritma A* yang dinilai cukup fleksibel untuk modifikasi parameter yang ingin diperhitungkan dalam mencari rute kendaraan umum yang optimal dan sesuai dengan keinginan penumpang angkot, seperti jumlah pergantian angkot pada rute, perjalanan dengan kaki, dan waktu menunggu angkot.

II. STUDI LITERATUR

A. *Graf*

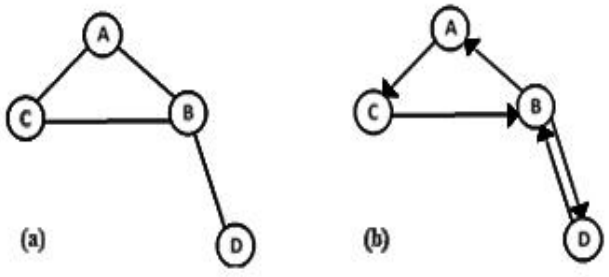
Pada makalah ini, struktur data lojik yang digunakan untuk merepresentasikan jalur lalu lintas adalah graf. Graf pada dasarnya terdiri dari dua komponen, yaitu titik/*node* (*vertex* atau *V*) dan busur (*edge* atau E) [6], atau dengan kata lain, suatu graf G dapat disimbolkan dalam notasi $G = (V,E)$. Komponen *V* pada graf berisi kumpulan titik yang ada pada graf, dan komponen *E* berisi kumpulan pasangan dua titik yang ada pada *V* yang terhubung satu sama lain pada graf. Contoh graf dapat dilihat pada Gambar 1 dengan: $V = \{A,B,C,D\}$ dan $E = \{(A,B),(A,C),(B,C),(B,D)\}$.



Gambar 1 Contoh graf sederhana

Jenis graf dapat dibagi menjadi dua, yaitu graf berarah dan graf tidak berarah [6]. Perbedaan secara semantik antara graf berarah dan tidak berarah adalah bahwa pada graf berarah urutan titik pada busur diperhitungkan, yang berarti busur (*A*,*B*) dan busur (*B*,*A*) adalah dua busur yang berbeda. Misalnya, pada graf di Gambar 2 b), terdapat busur (*A*,*C*) yang berarti *C* dapat dicapai dari *A*, namun *A* tidak dapat dicapai dari *C* karena tidak ada busur (*C*,*A*). Pada graf tidak berarah, urutan titik pada busur tidak diperhitungkan, yang berarti busur (*A*,*B*) = busur (*B*,*A*). Perbedaan tersebut dapat dilihat pada Gambar 2.

Struktur data graf dapat dipakai untuk memodelkan jalan pada dunia nyata dengan *V* mewakili kumpulan tempat yang ada pada kawasan yang dimodelkan, dan *E* adalah kumpulan jalan yang menghubungkan tempat-tempat tersebut. Graf berarah lebih realistis dalam memodelkan jalur lalu lintas karena suatu jalan dapat hanya memiliki satu arah untuk dipakai kendaraan.



Gambar 2 a) Contoh graf tidak berarah b) Contoh graf berarah

B. *Algoritma A**

Algoritma A* dikembangkan dengan mengkombinasikan pendekatan heuristik dari algoritma BFS (*Breadth-First Search*) dan pendekatan formal algoritma Djikstra yang dijamin dapat menemukan jalur terpendek pada tahun 1968 [7]. Algoritma A* memanfaatkan dua informasi yang dipakai kedua algoritma tersebut, yaitu jarak dari titik awal ke titik yang sekarang dievaluasi (digunakan pada algoritma Djikstra) dan estimasi jarak dari titik yang sekarang dievaluasi ke titik akhir (digunakan pada algoritma BFS). Berikut adalah representasi klasik dari algoritma A* [8]:

$$f'(n) = g(n) + h'(n) \quad (1)$$

$g(n)$: total jarak dari posisi awal ke titik *n*.
$h'(n)$: estimasi jarak dari titik *n* ke posisi tujuan.
$f'(n)$:estimasi jarak terpendek sejauh ini sampai ke titik *n*.

Ide dasar algoritma A* adalah mencari kumpulan busur yang saling terhubung pada graf dengan $f'(n)$ yang paling minimal hingga mencapai *node* tujuan. Fungsi heuristik $h'(n)$ yang dipakai dapat merupakan fungsi estimasi apapun, namun harus *admissible* (dapat diterima) untuk menjamin menghasilkan jarak yang optimal. Pengertian dari heuristik yang *admissible* adalah sebagai berikut [9]:

Sebuah fungsi heuristik $h'(n)$ adalah *admissible* jika untuk tiap *node* n,

$$h'(n) \leq h^*(n) \quad (2)$$

dimana $h^*(n)$ adalah *cost* yang sebenarnya untuk mencapai *node* tujuan dari *n*. Dengan kata lain, heuristik yang dapat diterima tidak pernah melebih-lebihkan (*overestimate*) biaya yang diperlukan untuk mencapai tujuan.

III. MODIFIKASI A* DENGAN BOBOT PREFERENSI

A. *Representasi Jalur Lalu Lintas Angkot dengan Graf Trayek Angkot*

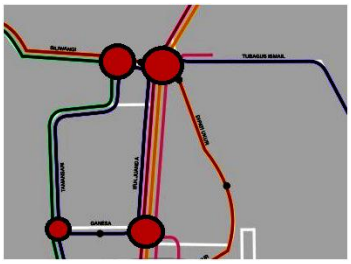
Sebagai ilustrasi keadaan trayek angkot di kota Bandung dapat dilihat pada Gambar 3. Tiap jalur yang berwarna menunjukkan trayek angkot. Tiap warna yang berbeda mewakili satu trayek angkot untuk jurusan tertentu. Sebagai contoh, pada Gambar 3, jalur berwarna hijau mewakili angkot jurusan Cicaheum – Ledeng, jalur berwarna biru tua mewakili angkot jurusan Sadang Serang – Caringin, jalur berwarna merah muda mewakili angkot jurusan Sadang Serang – Gede Bage, dan lainnya. Jalur putih menunjukkan jalur yang tidak dilalui angkot, atau dengan kata lain jalur untuk perjalanan dengan kaki.

Pada Gambar 3 dapat dilihat bahwa trayek angkot dapat saling *overlap* seperti trayek berwarna hijau dan biru tua. Namun, di suatu titik, pasti trayek angkot tersebut akan berpisah. Untuk memecahkan masalah ini, *node* yang diambil sebagai tempat pemberhentian adalah titik dimana jalur angkot yang saling *overlap* menjadi berpisah jalur. Gambar 3 yang sudah mencerminkan *node* tersebut dapat dilihat pada

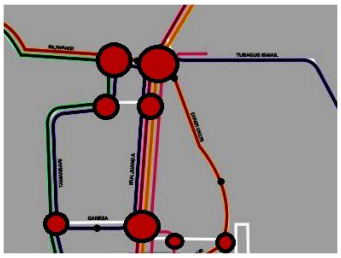
Gambar 4. Bulatan merah pada Gambar 4 menunjukkan *node* tersebut.

Selain titik dimana rute angkot yang saling *overlap* mulai berpisah, perlu juga dimasukkan titik dimana jalur angkot berpapasan dengan jalur yang hanya bisa dilewati oleh perjalanan dengan kaki sebagai *node* untuk algoritma ini. Hal tersebut dibutuhkan karena rute terpendek yang dihasilkan sebagai solusi memungkinkan campuran antara perjalanan dengan angkot dan juga dengan kaki. Gambar 4 yang sudah ditambah dengan *node* perjalanan kaki tersebut dapat dilihat pada Gambar 5.

Gambar 3 Potongan Trayek Angkot di Kota Bandung [10]



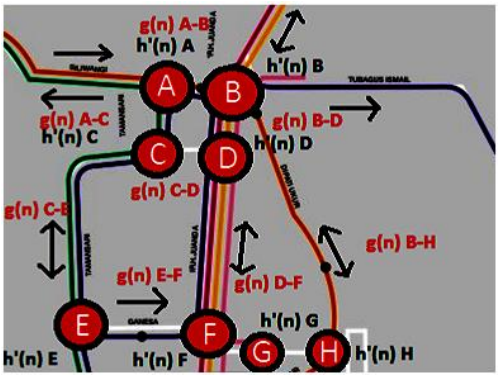
Gambar 4 Gambar trayek angkot yang sudah mengandung node untuk A*



Gambar 5 Gambar III-2 dengan node perjalanan kaki

Dapat dilihat pada Gambar 5 bahwa satu busur yang menghubungkan tiap *node* merah dilewati oleh ≥ 1 macam angkot. Ini artinya, pada tiap busur perlu dilengkapi informasi angkot apa saja yang melewati busur tersebut. Ilustrasi dari representasi data tersebut dapat dilihat pada Gambar 6.

Pada Gambar 6 dapat dilihat bahwa informasi yang disimpan untuk tiap busur pada graf tersebut adalah angkot apa yang melewati busur tersebut beserta arahnya, dan $g(n)$ dari busur tersebut, yang dibahas lebih lanjut pada subbab berikutnya. Untuk busur putih yang mewakili perjalanan kaki, cukup disimpan nilai $g(n)$-nya karena arahnya pasti dua arah dan tidak ada angkot yang melewatinya. Selain itu, pada tiap busur yang dilewati angkot, didefinisikan juga busur dua arah yang mewakili perjalanan dengan kaki. Terakhir, informasi yang disimpan pada tiap *node* merah adalah nama dari tiap *node*, atau dengan kata lain koordinat dari *node* tersebut dan $h'(n)$ atau nilai dari fungsi heuristik untuk *node* tersebut.



Gambar 6 Representasi lalu lintas angkot dengan graf

B. *Modifikasi f'(n) A* dengan Preferensi*

Fungsi heuristik $h'(n)$ yang dipakai pada makalah ini adalah jarak tegak lurus dari suatu *node* ke *node* tujuan. Dasar pemilihan fungsi tersebut adalah karena jarak tegak lurus antara dua titik adalah *admissible* karena mencerminkan jarak terpendek antara dua titik sehingga nilainya pasti lebih kecil dari jarak yang sebenarnya (bersifat optimis). Sedangkan $g(n)$ yang dipakai yang menggambarkan jarak yang sudah ditempuh sejauh ini didapat dari jumlah akumulatif panjang semua busur yang terhubung dari *node* awal sampai *node* yang sekarang dievaluasi.

Untuk menentukan nilai $f'(n)$ atau *path-cost function* yang menentukan bobot dari tiap *node*, pada dasarnya ada 4 hal yang perlu diperhitungkan selain meminimalkan jarak tempuh yang sesuai dengan rumusan masalah, yaitu meminimalkan jumlah pergantian angkot, jarak perjalanan dengan kaki, waktu tunggu angkot, dan kepadatan jalan. Salah satu pendekatan yang dapat digunakan untuk memodifikasi $f'(n)$ yang sudah ada pada adalah dengan menambahkan faktor pengali yang nilainya dipengaruhi oleh 4 faktor tersebut (jumlah pergantian angkot, jarak perjalanan dengan kaki, waktu tunggu angkot, dan kepadatan jalan). Modifikasi dari Persamaan (1) tersebut adalah sebagai berikut:

$$f'(n) = (g(n) + h'(n)) \times (p(n)) \quad (3)$$

Faktor pengali $p(n)$ yang ditambahkan pada Persamaan (3) adalah fungsi yang nilainya dipengaruhi oleh 4 faktor tambahan tersebut. Penentuan besar dari p disusun oleh 4 faktor berikut:

i. Meminimalkan jumlah pergantian angkot ($gt(n)$)

Jika *node* yang sekarang dihitung nilai $g(n)$-nya (untuk selanjutnya disebut *node* sekarang) adalah *node* awal, semua busur yang langsung terhubung dengan *node* awal diberi bobot $gt(n) = 0$. Setelah memilih busur dengan $f'(n)$ terkecil, jenis angkot

yang dipakai dicatat pada *node* tujuan busur tersebut. Misalnya, busur A-B dengan angkot Cicaheum-Ledeng adalah busur dengan $f'(n)$ terkecil, maka *node* A dimasukkan ke dalam *closed-list* dan pada *node* B disimpan data 'Cicaheum-Ledeng' yang mewakili angkot yang dipakai sampai *node* B. Jika ternyata busur yang dipilih adalah busur perjalanan kaki, maka tidak perlu dilakukan penyimpanan jenis angkot pada *node* B.

Jika *node* sekarang bukan *node* awal, busur yang dilewati angkot yang sama dan arah sama dengan angkot yang dicatat pada *node* tersebut diberi bobot $gt(n) = 0$. Busur angkot selain busur tersebut diberi bobot $gt(n) = [0..1]$ dan bobot $gt(n)=1$ untuk *node* perjalanan kaki. Jika *node* tersebut tidak menyimpan jenis angkot (dicapai dengan perjalanan kaki), maka semua busur yang terhubung dengan *node* tersebut diberi bobot $gt(n) = 0$.

ii. Meminimalkan jarak perjalanan dengan kaki ($jk(n)$)

Busur yang dilewati angkot dan langsung terhubung dengan *node* sekarang diberi bobot $jk(n) = 0$. Sedangkan busur yang tidak dilewati angkot dan langsung terhubung dengan *node* sekarang diberi bobot $jk(n) = [0..1]$.

iii. Meminimalkan waktu tunggu angkot ($wt(n)$)

Busur yang diperhitungkan adalah busur perjalanan kaki dan busur angkot yang masih beroperasi sampai 1 jam dari waktu saat ini. Tiap busur perjalanan kaki yang terhubung dengan angkot sekarang diberi $wt(n) = 0$. Busur yang dilewati angkot yang sama jenisnya dengan yang dicatat *node* sekarang juga diberi $wt(n)=0$. Busur selain itu yang dilewati angkot diberi nilai antara 0 sampai dengan 1, yang menunjukkan frekuensi kemunculan angkot (angkot dengan $wt(n)=1$ adalah angkot dengan frekuensi kemunculan terendah dan angkot dengan $wt(n)=0$ adalah angkot dengan frekuensi kemunculan tertinggi).

iv. Meminimalkan melewati jalan yang padat ($pd(n)$)

Kepadatan jalan diperhitungkan secara *real-time* dengan memperhitungkan keadaan jalan pada waktu pengguna memasukkan kueri pencarian. Bobot yang digunakkan bervariasi mulai dari 0 untuk busur yang lancar, sampai 1 untuk busur yang macet tidak bergerak.

Setelah $g(n)$, $h'(n)$, $gt(n)$, $jk(n)$, $wt(n)$ dan $pd(n)$ ditentukan, maka dapat ditentukan nilai $f'(n)$ sebagai berikut, yang didapat dari penjabaran faktor $p(n)$ yang ada pada Persamaan (3) :

$$p(n) = 1 + (gt(n) + jk(n) + wt(n) + pd(n)) \qquad (4)$$

$pj(n)$ = Panjang jalan dari *node* sekarang ke *node* n (km)
$gt(n)$ = Bobot ganti angkot sampai ke *node n* (0,[0..1],[0..1])
$jk(n)$ = Bobot perjalanan kaki sampai ke *node* n (0,[0..1])
$wt(n)$ = Bobot waktu tunggu angkot ([0..1])
$pd(n)$ = Bobot kepadatan ruas jalan ([0..1])

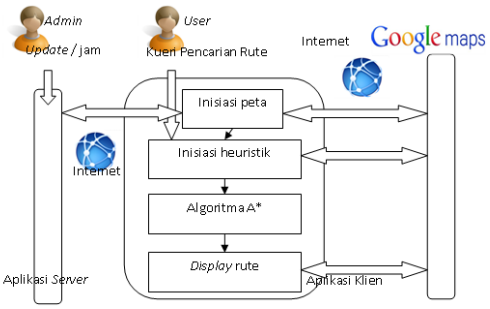
Nilai $jk(n)$ hanya mempunyai dua pilihan, yaitu 0 untuk nilai minimum, dan [0..1] untuk nilai maksimal yang dapat disesuaikan dengan preferensi pengguna. Nilai $gt(n)$ mempunyai tiga pilihan, yaitu 0 untuk nilai minimum, [0..1] untuk pergantian angkot yang dapat disesuaikan dengan preferensi pengguna, dan [0..1] untuk pergantian menjadi perjalanan kaki, dengan nilai lebih besar daripada nilai $gt(n)$ untuk pergantian angkot. Misalnya, jika seorang pengguna lebih memilih perjalanan kaki lebih menyusahkan daripada mengganti angkot karena sedang membawa banyak barang berat, dapat memilih bobot $gt(n) = 0{,}1$ dan bobot $jk(n) = 1$. Hal ini berarti, busur terbaik yang dipilih adalah busur dengan $gt(n) = 0$, $jk(n) = 0$, $wt(n) = 0$, dan $pd(n) = 0$ (tidak perlu mengganti angkot, tidak perlu berjalan, waktu tunggunya minimal, dan jalannya lancar), sehingga $g(n) = pj(n)$.

Nilai $p(n)$ pada Persamaan (3) bersifat lokal karena hanya bergantung pada *node n* yang sedang dievaluasi. Karena itu, dibuat lagi satu variasi peletakkan $p(n)$ seperti pada Persamaan (5) agar bobot $p(n)$ bersifat kumulatif dari *node-node* yang sebelumnya dilewati sampai ke *node n*. Peletakkan $p(n)^2$ pada $g(n)$ dan $p(n)$ pada $h'(n)$ adalah untuk mengecilkan peran $h'(n)$ dalam menentukan $f'(n)$ karena dimungkinkan mengambil jalan yang sedikit memutar (busur dengan $h'(n)$ yang lebih besar) untuk menekan perjalanan kaki dan mempertahankan angkot yang sedang dipakai, dan untuk menekankan peran $g(n)$ yang bersifat kumulatif dari *node-node* yang perlu dilalui sampai ke *node n*.

$$f'(n) = (g_{parent}(n) + (g(n) \times p(n)^2)) + (h'(n) \times (p(n)) \qquad (5)$$

C. *Arsitektur Umum Aplikasi*

Aplikasi yang dibuat dalam *platform* Android dengan alasan populer dalam penggunaan dan bersifat *open-source* sehingga pilihan fitur yang bisa dipakai relatif lebih banyak. Arsitektur aplikasi dapat dilihat pada Gambar 7. Tugas dari aplikasi *server* adalah memberikan data matriks keterhubungan jalan tiap satu jam (dilakukan oleh admin) untuk diakses aplikasi klien. Sedangkan eksekusi algoritma A* dilakukan secara lokal pada aplikasi klien yang menerima kueri dari *user* lalu menampilkan hasil rute pencarian. Google Maps API digunakan untuk menampilkan *map* serta menyediakan nilai fungsi heuristik untuk tiap *node*.



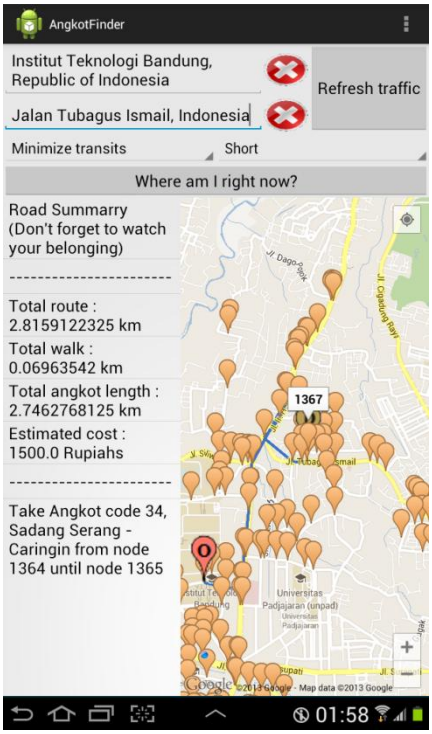
Gambar 7 Arsitektur Umum Aplikasi

Pertama-tama dilakukan inisiasi peta yaitu menyiapkan matriks keterhubungan tiap *node* yang diambil dari aplikasi *server*, partisi *node* secara lokal di aplikasi klien, dan *display* peta bandung dengan Google Maps API. Pengguna kemudian memasukkan kueri pencarian berupa titik *origin* dan *destination* pada peta yang sudah di-*display*, preferensi perjalanan kaki dan pergantian angkot, dan jarak maksimal perjalanan kaki yang diinginkan. Aplikasi kemudian melakukan inisialisasi nilai fungsi heuristik untuk tiap *node* yang ada melewati Google Maps API. Kemudian dilakukan algoritma A* berdasarkan data masukan tersebut yang kemudian menghasilkan 1 rute terbaik sesuai preferensi pengguna. Rute tersebut kemudian di-*display* pada aplikasi klien dengan bantuan Google Maps API.

IV. IMPLEMENTASI DAN PENGUJIAN

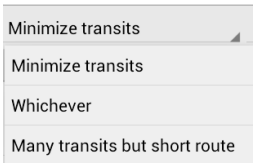
A. *Implementasi*

Antarmuka klien hanya terdiri dari satu layar untuk mempermudah navigasi. Antarmuka klien dapat dilihat pada Gambar 8 dan terdiri dari dua *textfield* untuk memasukkan titik asal dan tujuan, dilengkapi dengan tombol untuk menghapus *input* tersebut. Fitur *autocomplete* pada Gambar 9 memudahkan pengguna untuk memilih alamat asal dan tujuan. Preferensi pengguna dapat dimasukkan lewat *dropdown* preferensi transit dan jalan kaki, serta jarak maksimal perjalanan kaki yang dapat dipilih yang dapat dilihat pada Gambar 10 dan Gambar 11.



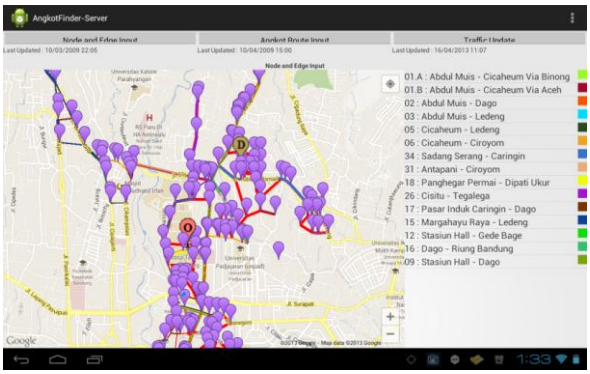
Gambar 8 Antarmuka Klien

Gambar 9 Autocomplete Input Alamat O-D

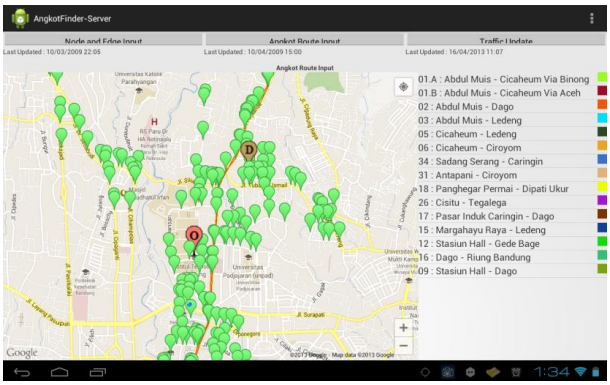


Gambar 10 Dropdown Preferensi Transit

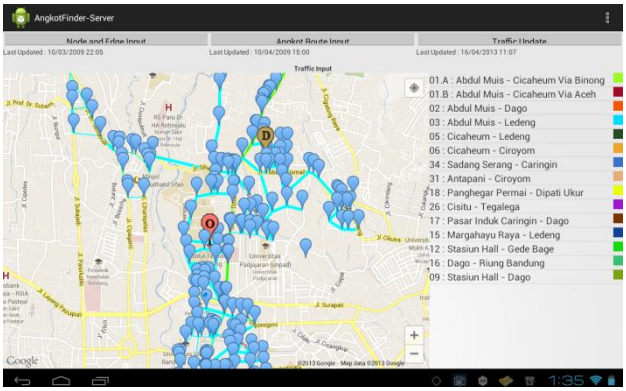
Gambar 11 Dropdown Jarak Maksimal Perjalanan Kaki



Gambar 12 Antarmuka Server mode Input Node dan busur



Gambar 13 Antarmuka Server mode Input Rute Angkot



Gambar 14 Antarmuka Server mode Input Kepadatan Jalan

Antarmuka *server* AngkotFinder mempunyai tiga mode. Mode *input node* dan busur, *input* rute angkot, dan *input* kepadatan jalan dapat dilihat pada Gambar 12, Gambar 13, dan Gambar 14 secara berurutan.

B. *Tujuan Pengujian*

Tujuan pengujian AngkotFinder terdiri dari beberapa hal, yaitu:

1. Menguji kesesuaian preferensi pengguna pada rute yang dihasilkan aplikasi, baik untuk algoritma A* dengan preferensi versi I (Persamaan (3)) dan versi II (Persamaan (5)) untuk mengetahui versi mana yang lebih baik.
2. Mengukur pengaruh penambahan faktor preferensi pengguna terhadap waktu eksekusi algoritma pencarian.

C. Batasan Pengujian

Batasan pengujian AngkotFinder terdiri dari beberapa hal, yaitu:

1. Daerah yang diperhitungkan dalam pencarian rute adalah wilayah Bandung Utara sebagai studi kasus.
2. Faktor yang diperhitungkan dalam pencarian rute adalah preferensi penguna (banyak ganti angkot dan jarak maksimal perjalanan kaki), waktu menunggu angkot, dan kepadatan jalan saat pencarian rute, sesuai dengan rumusan masalah.
3. Faktor yang tidak diperhitungkan dalam pencarian rute adalah ongkos total yang diperlukan saat menaiki angkot, waktu terpendek, dan kemungkinan angkot keluar dari trayeknya (biasanya untuk menghindari jalan macet).

D. Rancangan Kasus Uji

Rancangan kasus uji AngkotFinder terdiri dari dua, yaitu untuk pengujian kebenaran rute pencarian dan kinerja perangkat lunak. Rancangan ini dibuat agar pengujian AngkotFinder dapat dilakukan secara sistematis dan terstruktur.

1. Kasus Uji Kebenaran Perangkat Lunak

   Pengujian kebenaran rute pencarian AngkotFinder dilakukan dengan membandingkan hasil rute pencarian dari aplikasi dengan hasil rute dari *survey* rute angkot dari pengguna rutin angkot sebanyak 18 rute. Pemberian skor untuk tiap rute dapat dilihat pada Tabel I. Kasus uji ini dibuat untuk memenuhi tujuan pengujian nomor satu. Bobot preferensi yang digunakan adalah sebagai berikut:

   i) Preferensi Ganti Angkot
      (1) Banyak Ganti Angkot :
      Bobot maksimal $gt(n)$: 0.1.
      (2) Mana Saja :
      Bobot maksimal $gt(n)$: 0.3.
      (3) Ganti Angkot Minimal:
      Bobot maksimal $gt(n)$: 0.5.
   ii) Preferensi Perjalanan Kaki
      (1) Perjalanan Kaki Pendek :
      Bobot maksimal $jk(n)$:
      Bobot maksimal $gt(n)$ + 0.5.
      (2) Perjalanan Kaki Sedang :
      Bobot maksimal $jk(n)$:
      Bobot maksimal $gt(n)$.
      (3) Perjalanan Kaki Panjang :
      Bobot maksimal $jk(n)$:
      Bobot maksimal $gt(n)$ – 0.5.

   Bobot tersebut dipilih secara heuristis, namun tetap dengan ketentuan bahwa besar kecilnya bobot ditentukan oleh preferensi pengguna. Misalnya, jika pengguna tidak keberatan banyak ganti angkot, maka dipilih bobot maksimal $gt(n)$ yang kecil, misalnya 0.1. Lalu, untuk pengguna yang ingin meminimalkan ganti angkot, maka bobot maksimal $gt(n)$ yang dipilih adalah 0.5, angka yang lebih besar dari 0.1.

   Bobot untuk kepadatan jalan juga dipilih secara heuristis, tetapi tetap dengan ketentuan jalan lancar mempunyai bobot $pd(n)$ yang kecil, dam semakin padat jalan, bobot $pd(n)$-nya semakin besar. Bobot $pd(n)$ untuk busur perjalanan kaki disamakan dengan bobot jalan padat dengan asumsi kecepatan berjalan orang dewasa sama dengan kecepatan kendaraan pada jalan yang padat. Berikut bobot $pd(n)$ yang dipakai saat pengujian berdasarkan kecepatan kendaraan pada busur tersebut:

   i. 40-57 km/jam : Lancar, $pd(n) = 0.1$
   ii. 26-40 km/jam : Sedang, $pd(n) = 0.3$
   iii. 17-26 km/jam : Padat, $pd(n) = 0.5$
   iv. <17 km//jam : Macet, $pd(n) = 0.7$.

Tabel I

Kriteria	<= rute *survey*	> rute *survey*
Jarak Total	1	0
Jarak total jalan kaki	1	0
Jumlah total ganti angkot	1	0
Waktu tunggu angkot	1	0
Total kepadatan jalan	1	0

2. Kasus Uji Kinerja Perangkat Lunak

   Kinerja AngkotFinder juga diuji dengan membandingkan waktu eksekusi pencarian dengan dan tanpa memperhitungkan preferensi pengguna. Kasus uji ini dibuat untuk memenuhi tujuan pengujian nomor dua.

E. Hasil dan Analisis Pengujian

Hasil pengujian kebenaran perangkat lunak dapat dilihat pada
Tabel II. Dapat dilihat bahwa hasil pengujian cukup bervariasi jika dikelompokkan per faktor preferensi.

Tabel II

Algoritma	Akurasi per Faktor Preferensi (%)				
	Jarak Total	Jumlah Ganti Angkot	Jarak Total Jalan Kaki	Waktu tunggu angkot	Total kepadatan jalan
A* I	83.33	44.44	44.44	66.67	50
A* II	55.55	38.89	61.11	66.67	72.22

Analisis hasil uji tersebut berdasarkan tiap faktor preferensi adalah sebagai berikut:

1. Jarak Total Perjalanan

   Ketepatan algoritma dalam meminimalkan jarak total perjalanan pada
   Tabel II adalah 83.33% dan 55.55% untuk A* versi I dan II secara berurutan. Hal ini dapat dijelaskan karena peletakkan bobot $p(n)$ untuk A* versi II (Persamaan (5)) lebih memberatkan pada bobot $g(n)$ dibandingkan bobot $h'(n)$ ($p(n)^2$ untuk $g(n)$ dan $p(n)$ untuk $h'(n)$), sehingga busur yang lebih diprioritaskan cenderung adalah busur dengan $g(n)$ terpendek, padahal belum tentu busur tersebut paling dekat dengan titik tujuan ($h'(n)$). Hal tersebut menyebabkan jarak total dari rute yang dihasilkan A* versi II cenderung lebih besar daripada yang dihasilkan A* versi I yang bobot $p(n)$-nya mempengaruhi $g(n)$ dan $h'(n)$ dengan sama rata, sehingga A* versi I dapat menekan jarak total rute yang dihasilkan dengan lebih baik.

2. Preferensi Pergantian Angkot

   Dari
   Tabel II dapat disimpulkan bahwa persentase ketepatan algoritma dalam meminimalkan jumlah pergantian angkot dari pasangan *O-D* data uji adalah sekitar 44.44% dan 38.89% untuk A* versi I dan II secara berurutan, dan angka tersebut dinilai kurang memuaskan. Ketepatan yang rendah ini disebabkan karena pemilihan busur angkot dititikberatkan pada angkot dengan frekuensi tertinggi ($wt(n)$ kecil) dan angkot yang sudah dinaiki sebelumnya ($gt(n)=0$). Padahal, angkot dengan frekuensi tertinggi pada suatu busur belum tentu akan membawa pengguna sampai pada titik tujuan.

3. Preferensi Jarak Perjalanan Kaki

   Akurasi algoritma terhadap peminimalan jarak perjalanan kaki pada data uji adalah 44.44% untuk A* versi I, yang dinilai kurang memuaskan, dan 61.11 % untuk A* versi II yang dinilai cukup memuaskan. Perbedaan yang besar ini sebenarnya disebabkan oleh sifat dari pembobotan $p(n)$ pada versi II yang bersifat kumulatif, yaitu bobot dari *node* sebelumnya akan terbawa kepada *node* anaknya dan semakin membesar. Oleh karena itu, jika ada dua busur jalan kaki pada *open-list*, busur yang dipilih adalah busur yang bobot perjalanan kakinya paling kecil dari titik awal sampai busur tersebut dilewati. Sedangkan letak $p(n)$ pada A* versi I tidak menyebabkan bobot $p(n)$ tersalur secara kumulatif dan hanya bergantung pada busur yang saat itu sedang dievaluasi. Hal tersebut menyebabkan busur perjalanan kaki yang dipilih semata-mata hanya yang panjangnya paling pendek, walau bisa saja total jarak perjalanan kaki sampai ke busur itu tidak yang paling pendek.

4. Waktu tunggu angkot

   Akurasi algoritma terhadap penekanan waktu tunggu angkot pada data uji bernilai 66.67% untuk versi I dan II, yang dinilai cukup memuaskan. Hal ini disebabkan oleh bobot $wt(n)$ pada faktor pengali *path cost function* yang menyebabkan busur angkot yang dipilih adalah selalu busur angkot dengan frekuensi tertinggi pada busur tersebut.

5. Total kepadatan jalan

   Dari
   Tabel II dapat disimpulkan bahwa presentase ketepatan algoritma dalam memilih jalur yang lancar dari pasangan *O-D* data uji adalah sekitar 50% untuk versi I dan 72.22% untuk versi II. Perbedaan persentase yang cukup besar ini sebenarnya penjelasannya sama seperti pada poin 3 sebelumnya untuk faktor preferensi perjalanan kaki. Bobot $p(n)$ untuk A* versi II bersifat kumulatif dan transitif dari *node-node parent* untuk sebuah *node*, sehingga *node* yang dipilih adalah *node* yang total bobot kepadatan jalannya paling kecil. Sedangkan bobot $p(n)$ untuk A* versi I hanya bergantung pada *node* yang saat ini dievaluasi, sehingga jika hanya faktor kepadatan jalan yang dilihat, *node* yang dipilih adalah *node* yang paling kecil bobot kepadatan jalannya, walau bisa saja bobot total kepadatan jalan sampai pada *node* tersebut bukan yang minimal.

Tabel III

Rata-rata waktu eksekusi data uji (detik)			
Djikstra	A* (Tanpa preferensi)	A* (Dengan preferensi) I	A* (Dengan preferensi) II
1.2984	1.3419	4.6744	4.574

Hasil pengujian kinerja perangkat lunak dapat dilihat pada Tabel III. Dari Tabel III didapat kesimpulan bahwa urutan waktu eksekusi yang paling singkat adalah dimulai dari algoritma Djikstra dengan waktu eksekusi 1.2984 s, lalu A* 1.3419 s atau 1.03 kali lipat dari Djikstra, untuk algoritma A* dengan preferensi versi I adalah 4.6744 s atau sekitar 3.6 kali lipat dari Djikstra, dan untuk versi II adalah 4.574 s atau sekitar 3.52 kali lipat dari Djikstra. Algoritma A* pada pengujian ini memiliki waktu lebih lambat daripada algoritma

Djikstra membuktikan bahwa heuristik yang dipakai pada makalah ini, yaitu jarak *eucledian* adalah *admissible* karena hanya menambah waktu eksekusi sebesar 0.03 kali lipat walau diperlukan komputasi tambahan untuk menghitung jarak *eucledian* setiap kali suatu *node* dihitung nilai *path-cost*-nya.

Sedangkan untuk algoritma A* dengan preferensi, baik versi I maupun versi II mempunyai waktu eksekusi yang didapat lebih besar daripada A* tanpa preferensi. Hal ini wajar karena penambahan penghitungan preferensi pengguna pada A* pasti akan menambah langkah komputasi pada A* untuk mendapatkan nilai fungsi *cost path*, yang pada akhirnya meningkatkan waktu eksekusi algoritma A*. Rata-rata waktu eksekusi untuk kedua versi algoritma A* dengan preferensi tersebut adalah sekitar 5 detik dan merupakan waktu yang wajar untuk digunakan pada aplikasi AngkotFinder.

V. KESIMPULAN

1. Algoritma A* dapat dimodifikasi untuk mengoptimalkan rute terpendek penggunaan angkutan umum sesuai dengan preferensi pengguna.
2. Perangkat lunak AngkotFinder yang mengimplementasikan modifikasi algoritma A* untuk memenuhi preferensi pengguna angkutan umum berhasil dibangun.
3. Penambahan faktor preferensi pengguna pada *path cost function* Algoritma A* membuat waktu eksekusi A* meningkat sekitar 3.5 kali lipat, tetapi pada aplikasi ini waktu eksekusi rata-rata setelah pengalian tersebut masih dalam batas wajar untuk pemakaian pengguna, yaitu sekitar 5 detik.
4. Akurasi algoritma A* dengan preferensi pengguna untuk kedua versi dapat dilihat pada Tabel IV. Simbol + berarti akurasi baik, o untuk akurasi sedang, dan – untuk akurasi kurang. Dapat disimpulkan bahwa algoritma A* versi II lebih memuaskan untuk memenuhi preferensi pengguna karena umumnya pengguna lebih menyukai rute yang lebih sedikit perjalanan kakinya dan tidak macet, dibanding dengan rute yang jarak totalnya lebih sedikit namun dengan perjalanan kaki yang lebih banyak dan melewati jalan yang lebih padat yang cenderung dihasilkan oleh algoritma A* versi I.

Tabel IV

Algoritma	Jarak Total	Ganti Angkot	Jarak Jalan Kaki	Waktu Tunggu Angkot	Kepadatan Jalan
A* I	+	-	-	o	-
A* II	-	-	o	o	+

5. Penambahan bobot faktor preferensi pengguna pada pencarian ini bersifat umum, yaitu dapat diaplikasikan pada *path-cost function* algoritma pencarian rute apapun.

VI. SARAN

1. Perlu dilakukan modifikasi pada langkah di algoritma A* (tidak hanya sekedar modifikasi *path-cost function*) untuk meminimalkan jumlah pergantian angkot pada pencarian rute. Misalnya, dengan memodifikasi algoritma A* biasa menjadi algoritma A* dengan pencarian dua arah (*bi-directional search*) untuk menjamin busur angkot yang dipilih juga melewati titik tujuan sehingga pemilihan busur angkot tersebut dapat terus dipertahankan sampai rute pencarian berhasil ditemukan.
2. Untuk meningkatkan performansi perangkat lunak dalam segi waktu eksekusi, dapat digunakan heuristik untuk A* yang lebih mangkus daripada jarak *eucledian*.

REFERENSI

[1] M. Rosenberg, "What is Geography? : Current World Population," 2012. [Online]. Available: http://geography.about.com/od/obtainpopulationdata/a/worldpopulation.htm. [Diakses 29 November 2012].

[2] S. Bambang, "1-2-3 Langkah: Langkah Kecil yang Kita Lakukan Menuju Transportasi yang Berkelanjutan," *Majalah Transportasi Indonesia*, vol. I, pp. 89-95, 2004.

[3] Q. Wu dan J. Hartley, "USING K-SHORTEST PATHS ALGORITHMS TO ACCOMMODATE USER PREFERENCES IN THE OPTIMIZATION OF PUBLIC TRANSPORT TRAVEL," Nottingham Trent University, Nottingham, 2004.

[4] D. Ertanto, "SISTEM INFORMASI UNTUK MELIHAT RUTE TERPENDEK DAN JALUR ANGKOT BERBASIS SMS," Institut Pertanian Bogor, Bogor, 2006.

[5] A. Lutris, M. dan G. , "PERANCANGAN APLIKASI PENENTUAN RUTE ANGKUTAN KOTA BERBASIS ANDROID MENGGUNAKAN ALGORITMA A* UNTUK DAERAH JAKARTA BARAT," Universitas Bina Nusantara, Jakarta, 2012.

[6] T. H. Cormen, "Introduction to Algorithms," London, MIT Press, 2009, pp. 1168-1169.

[7] A. Patel, "Introduction to A*," Stanford University, 18 Juni 2009. [Online]. Available: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html. [Diakses 27 Desember 2012].

[8] C. W. Armstrong, "Path Finding - A* Algorithm," 14 Juli 2011. [Online]. Available: http://www.edenwaith.com/products/pige/tutorials/a-star.php. [Diakses 27 Desember 2012].

[9] S. Russel dan P. Norvig, Artificial Intelligence : A Modern Approach, New Jersey: Pearson Education , Inc., 2003.

[10] "Peta Angkot Bandung 2011 – Versi 1.0," 14 Maret 2011. [Online]. Available: http://petaangkot.wordpress.com/2011/03/14/peta-angkot-bandung-2011-versi-1-0/. [Diakses 6 Januari 2013].