ISSN 2085-4552

Optimasi Pencarian Jalur dengan Metode A-Star

Studi Kasus: Area Gading Serpong, Tangerang

Veronica Mutiana, Fitria Amastini, Noviana Mutiara

Program Studi Teknik Informatika, Universitas Multimedia Nusantara, Tangerang, Indonesia
veronicamutiana@gmail.com, amastini22@gmail.com, novianamutiara28@gmail.com

Diterima 16 Desember 2013
Disetujui 30 Desember 2013

***Abstract*— High level of traffic density can lead to traffic jam those will make troublesome for driver to reach destination with alternative shortest path on time. Therefore, it is neccessary to make an agent that can choose optimal route without being stuck on traffic jam. In this paper, algorithm for choose optimal route is A* method for shortest path problem and use backtrack process when there is a traffic jam occurs on several roads. The design of algorithm is tested by using data which contain 100 locations or nodes and 158 roads or paths in Gading Serpong with an agent that can searching shortest path and sensor module that can send the traffic status based on number of vehicle on several particular node. Based on testing, A* method does not guarantee for path selection if agent is not full observable with environment and there is some case that can lead a worst case.**

***Index Terms*— A* Algorithm, Backtrack, Shortest Path, Traffic Density**

I. PENDAHULUAN

Agent yang dapat memilihkan jalur perjalanan memang sudah ada. Akan tetapi, dengan algoritma untuk mengecek kemacetan yang terjadi pada suatu jalan, jalur yang terpilih diharapkan merupakan jalur alternatif yang paling optimal atau jalur terpendek untuk mencapai tujuan. Pada *paper* ini dijelaskan algoritma pencarian jalur menggunakan metode *A** (*A-star*) dengan proses *backtrack*. Metode *A** untuk mencari jalur terpendek dari tempat awal ke tempat tujuan, dan jika terjadi macet, *agent* menggunakan proses *backtrack* untuk mencari jalur terpendek kedua, dan seterusnya.

Pembuktian optimalisasi dari algoritma ini menggunakan studi kasus data sampel berupa jalur-jalur di daerah Gading Serpong (didapat dari pembuatan graph berdasarkan peta Gading Serpong dari *Google Maps*) dengan data kepadatan yang acak (didapat dari modul sensor dan bukan sensor yang sebenarnya pada tiap tempat atau *node*) dan tipe kendaraan sampel adalah kendaraan beroda empat.

II. SOLUSI DAN ALGORITMA PEMILIHAN JALUR

Seperti pada [1], dalam pencarian jalur paling optimal (jalur terpendek tanpa terjebak kemacetan), diperlukan suatu *graph G(N, A)* dimana *N* adalah *node* atau tempat-tempat yang dilewati dan *A* adalah *arch* atau *link* atau jalur yang dilewati. Selain itu, diperlukan juga *cost* dari *node* asal sampai *node* tujuan, yang didapat dari jalur $a = (i,j) \in A$ dimana jumlah *node* $| N | = n$ dan jumlah jalur $| A | = m$ dari *node* asal *i* sampai *node* tujuan *j* sehingga didapat c_{ij} atau *cost* dari *i* ke *j*. *Cost* tersebut didapatkan tergantung dari jenis kendaraan dan jalur yang dilewati.

Algoritma standar dalam pencarian jalur terpendek [1] dengan beberapa penyesuaian adalah sebagai berikut:

Step 1 : *Initialization: Set i = node*(*i*); *cost*(*i*) = 0; *cost*(*j*) = ∞

$\forall j \neq i$; *shortestPath*(*i*) = *NULL*;

executableSetOfNode Q = {i}.

Step 2 : *Node Selection: Select and remove node(i) from Q.*

Step 3 : *Node Expansion: Scan each link emanating from node(i). For each link a = (i, j).*

then cost(j) = cost(i) + cost(a);

shortestPath(i) = a

Insert node(j) into Q

Step 4 : *Stopping Rule*:

If Q = ∅ then STOP; Else goto step 2.

Dari algoritma pencarian jalur terpendek tersebut, telah dikembangkan dengan menggunakan salah satu dari *heuristic search* yaitu algoritma *A**. Setelah mendapatkan jalur terpendek dari tempat awal sampai

tujuan, beberapa jalur yang terpilih mempunyai kemungkinan terjadi kemacetan, mulai dari kemacetan yang ringan sampai kemacetan yang parah sehingga kendaraan tidak dapat melewati jalur tersebut. Oleh karena itu, proses *backtrack* diimplementasikan dalam mengoptimalisasi algoritma pencarian jalur.

A. Algoritma *A**

Algoritma *A** merupakan salah satu dari *heuristic search*, adalah algoritma untuk mencari estimasi jalur dengan *cost* terkecil dari *node* awal ke *node* berikutnya sampai mencapai *node* tujuan. *A** memiliki suatu fungsi yang didenotasikan dengan *f(x)* untuk menetapkan estimasi *cost* yang terkecil dari jalur yang dilalui *node* x dengan rumus sebagai berikut.

$$f(x) = h(x) + g(x) \quad (1)$$

Fungsi *h(x)* adalah *hyphotesis cost* atau *heuristic cost* atau estimasi *cost* terkecil dari *node* x ke tujuan, yang disebut juga sebagai *future path-cost*. Fungsi *g(x)* adalah *geographical cost* atau *cost* sebenarnya dari *node x* ke *node* tujuan, yang disebut juga sebagai *past path-cost*.

Dengan metode atau algoritma *A**, *cost* untuk mencapai *node* berikutnya didapat dari fungsi *f(x)*, sehingga pada pemilihan jalur terpendek dapat langsung diketahui *node* berikutnya dengan *cost* terkecil sampai mencapai *node* tujuan tanpa kembali ke *node* yang sudah dikunjungi.

Berdasarkan algoritma standar pencarian jalur terpendek sebelumnya, jika ditambahkan dengan metode *A**, algoritma tersebut mengalami perubahan, khususnya saat perluasan *node* atau *Node Expansion*, yaitu saat memindai jalur atau *link*.

Step 3 : *Node Expansion: Scan the outgoing links of node i. For each link (i, j)*

If cost(i) + cij + estimatedCost(j,d)< F(j), then

cost(j) = cost(i) + cij;

F(j) = cost(i) + cij + estimatedCost(i,d);

shortestPath(j) = a; d = destinationNode

Insert node j into Q;

Karena *A** merupakan *best-first search*, semua *node* memenuhi pertidaksamaan

$$cost(i) + estimatedCost(i,d) \leq cost(d) \quad (2)$$

Node yang memenuhi pertidaksamaan di atas diperiksa sebelum algoritma diterminasi atau sebelum *node* tujuan diperiksa [2].

B. Proses *Backtrack*

Proses *backtrack* yang digunakan oleh modul *agent* tidak hanya dilakukan dalam melakukan pencarian jalur terpendek, tetapi juga dilakukan ketika menemui kemacetan pada jalur yang dilewati dan menemui jalan buntu, sehingga *agent* harus bergerak mundur dan memilih jalur alternatif lain [3], sehingga *stopping rule* pada algoritma pencarian jalur hanya jika pencarian mencapai *node* tujuan (*j*).

Proses ini dapat dilakukan oleh modul *agent* dengan menyimpan informasi mengenai *node* apa saja yang sudah dikunjungi. Kemudian, ketika modul *agent* menemui *node* dengan kondisi kemacetan yang parah atau saat modul *agent* menemui jalan buntu, *node* tersebut ditandai sebagai *node* yang tidak boleh dilewati dan mundur ke *node* sebelumnya untuk mencari jalur lain yang memiliki solusi.

C. Traffic Density atau Tingkat Kepadatan Jalan

Kepadatan jalan tiap *node* dalam periode *T* tertentu dicari untuk menentukan *node-node* yang mengalami kemacetan dan *action* yang dilakukan *agent* terhadap tingkat kemacetan tertentu. Dengan mengetahui tingkat kepadatan jalan dari *node* yang menjadi solusi pemilihan jalur, jalur yang terpilih semakin optimal karena *agent* mengetahui *node* tujuan yang tingkat kemacetannya cukup tinggi, sehingga dapat melakukan proses *backtrack* saat terjadi kemacetan.

Kepadatan jalan ditentukan berdasarkan jumlah kendaraan yang melewati suatu *node X* yang didapat oleh modul sensor dalam periode *T* tertentu [4].

$$density = V / T \quad (3)$$

Agent modul sensor mengirim data jumlah kendaraan yang dibedakan menjadi 3 tipe kendaraan : *small vehicle* (V_1) seperti mobil sedan dan mobil famili, *medium vehicle* (V_2) seperti mobil van, dan *big vehicle* (V_3) seperti bus dan truk pengangkut material. Berdasarkan *Passenger Car Equivalents* (variabel untuk tingkat lalu lintas) [5], data jumlah kendaraan dari modul sensor untuk tipe *medium vehicle* dikalikan 1.5 dan tipe *big vehicle* dikalikan 2 karena ukurannya lipat lebih besar dibandingkan *small vehicle*. Jika waktu periode *T* adalah tiap 60 detik, maka persentase tingkat kepadatan jalan (*Dp*) didapat dari rumus berikut.

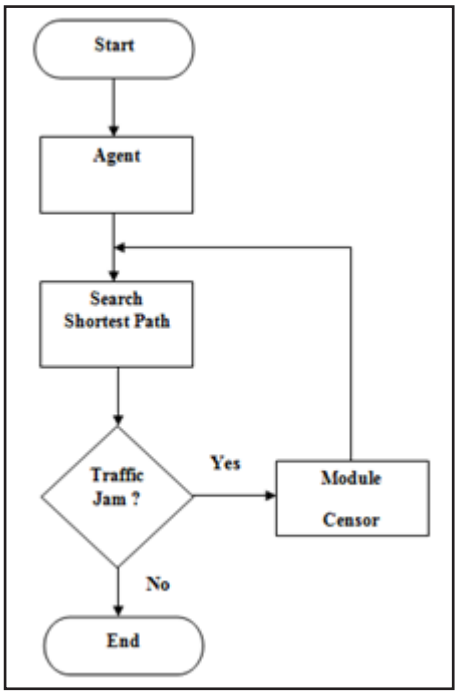
$$Dp = ((V_1 + 1.5V_2 + 2V_3) / 60) \% \quad (4)$$

III. PERANCANGAN ALGORITMA

Terdapat dua model perancangan yang diterapkan, yaitu model sistem secara umum dan model alur algoritma berdasarkan dari model sistem secara umum yang telah dijabarkan lebih detail.

A. Model Sistem secara Umum

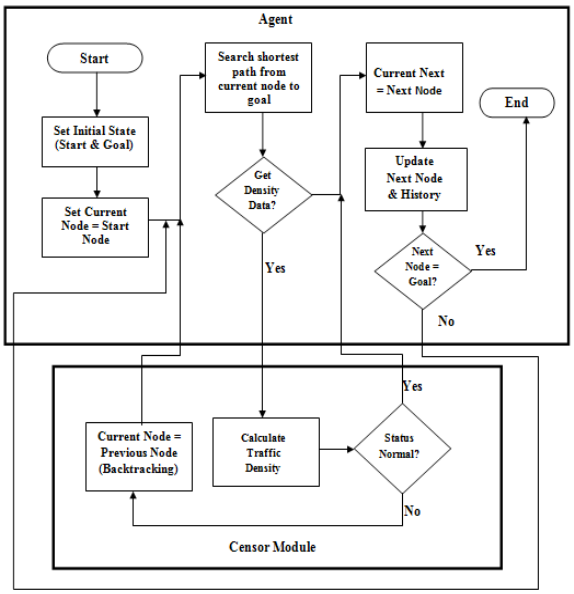
Pada model sistem secara umum, *agent* yang sudah mengetahui *node* tujuan mencari jalur terpendek dimulai dari *node* awal. Ketika *agent* melakukan pencarian jalur terpendek yang akan dilewati, terdapat mekanisme pengecekan kepadatan lalu lintas pada jalan yang akan dilewati. Jika terdapat kepadatan lalu lintas, *agent* akan menerima lagi data kepadatan lalu lintas di jalan lain dari modul sensor, sehingga *agent* dapat menentukan pencarian jalur terpendek lain yang lalu lintasnya tidak mengalami kepadatan.



Gambar 1 Model Sistem secara Umum

B. Model Alur Algoritma

Berdasarkan dari model sistem secara umum, model sistem algoritma menggambarkan detail dari kinerja *agent* dan modul sensor.



Gambar 2 Model Alur Algoritma

Agent menginisialisasi *node* awal dan tujuan, dimana *node* awal adalah *node* dimana *agent* tersebut berada sekarang. Kemudian, *agent* akan mencari jalur terpendek. Jika *agent* sudah mendapatkan data kepadatan lalu lintas yang akan dilewati, modul sensor akan mengkalkulasi persentase kepadatan jalan. Jika modul sensor memberikan status normal pada jalan yang akan dilewati, *agent* akan memilih jalan tersebut. Jika modul sensor mengirimkan status bahwa jalan tersebut mengalami kepadatan lalu lintas, *agent* akan melakukan proses *backtrack* [3], yaitu kembali ke *node* sebelumnya dan mencoba untuk mencari jalan terpendek lainnya yang status kepadatannya normal. *Agent* akan terus mencari jalur terpendek yang memiliki status normal atau jalan yang tidak mengalami kepadatan lalu lintas dan setelah menemukan jalur yang tingkat kepadatannya masih normal, *agent* memperbarui histori berupa informasi seperti *node* sebelum, nama jalan yang dilewati, *node* sesudah, dan jarak yang ditempuh.

C. *Rule* untuk Tingkat Kepadatan Jalan

Semakin besar persentase kepadatan yang ditunjukkan oleh suatu *node*, berarti tingkat kepadatan jalan pada *node* tersebut semakin tinggi, dan meningkatkan kemungkinan terjadinya kemacetan. Agar *agent* dapat memilih jalur secara optimal, modul sensor pada setiap *node* akan mengirim status kemacetan berdasarkan jumlah kendaraan beroda empat jika suatu *node* mengalami kemacetan. Asumsikan terdapat tiga status yang mungkin dikirim oleh suatu *node*, dimana range persentase status tersebut merupakan 1/3 dari 100% tingkat kepadatan jalan

sesuai rumus (4), yaitu normal, sedikit padat, dan sangat padat. Untuk tiap status, *action* atau reaksi yang akan dilakukan *agent* adalah sebagai berikut.

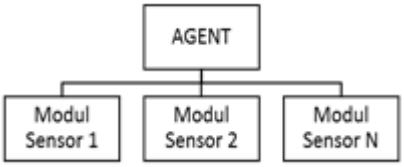
Tabel 1. *Rule* mengenai reaksi *Agent* terhadap persentase kepadatan jalan dari modul sensor

Density Percentage (D_p)	Status	*Action* atau Reaksi Agent
>0% - <=33.3%	*Normal*	Lewat jalur ini
>33.3% - <66.7%	*Jammed*	Bandingkan dengan jalur lain, apabila *density percentage* jalur terpendek kedua <=33.3% dan beda jarak dengan *route* sebelumnya < 1 kilometer, maka agent akan melewati jalur tersebut
>66.7%	*Heavy Jammed*	Cari jalur lain (proses *backtrack*)

D. Rancangan Program

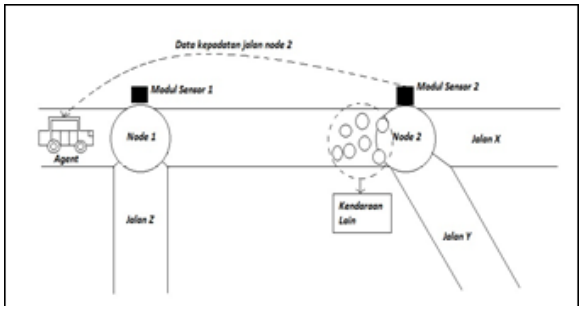
Program dibuat untuk mensimulasikan pengujian berdasarkan rancangan algoritma yang sudah dibuat, yaitu pencarian jalur oleh *Agent* menggunakan algoritma *A** dan proses *backtrack* serta modul sensor yang dapat mengirimkan persentase kepadatan jalan kepada *agent*.

Tiap *node* memiliki modul sensor sehingga banyaknya *node* yang mengalami kemacetan adalah 0 < jumlah *node* macet < jumlah *node* yang dilewati. Hubungan antara modul sensor di tiap *node* dengan *agent* adalah sebagai berikut.



Gambar 3. Hubungan antara *Agent* dan Modul Sensor

Dengan mempertimbangkan hubungan antara *agent* dan modul sensor dan algoritma, dimana modul sensor akan mengirim data kemacetan suatu *node* kepada *agent* sesuai dengan jalur yang dilewati *agent*, maka simulasi yang dilakukan dapat digambarkan sebagai berikut.



Gambar 4. Gambaran Simulasi

IV. PENGUJIAN

Pengujian optimasi pemilihan jalur berdasarkan algoritma yang sudah dirancang direpresentasikan dengan menggunakan program simulasi yang sudah dibuat. Inisiasi data sampel, perancangan program simulasi, dan hasil dari pengujian algoritma yang telah dibuat adalah sebagai berikut.

A. Inisiasi Data

Data sampel yang digunakan, seperti yang telah disebutkan sebelumnya, yaitu tempat dan jalur di daerah Gading Serpong yang didapatkan dengan melakukan analisis terhadap peta Gading Serpong pada aplikasi web *Google Maps*. Data pemerintah tidak menjadi acuan dalam pengujian karena daerah Gading Serpong termasuk daerah yang baru dan masih dalam tahap pembangunan sehingga data pemerintah masih minim. Tempat-tempat tersebut direpresentasikan sebagai *node*, sedangkan nama jalan direpresentasikan sebagai *path*. Panjang jalan didapatkan menggunakan *Distance Measurement Tools* dari aplikasi web tersebut. Bunderan, persimpangan, dan portal yang ada di Gading Serpong juga direpresentasikan sebagai *node*, dengan mempertimbangkan posisinya terhadap *node* lain, sehingga semua *node* dapat terhubung. Dari analisis tersebut, didapat data sebanyak 100 *node* dan 158 *path*, dimana tiap *node* dihubungkan oleh *path*.

Data *node* dan *path* tersebut merupakan *knowledge base* dari program. Inisiasi pertama adalah *node* awal (*i*) dan *node* tujuan (*j*). Inisiasi jumlah *node* yang dilewati (*N*), jarak dari *node* awal ke *node* tujuan (X_{ij}), jumlah *node* yang mengalami kemacetan (*t*) merupakan input acak dari modul sensor. *Heuristic cost* untuk pencarian dengan *A** diinisiasi $H(x) = 0$.

B. Hasil Pengujian

Pengujian dilakukan untuk setiap status dari tingkat kepadatan jalan, yaitu Normal (*nl*), *Jammed* / 'sedikit macet' (*j*), dan *Heavy Jammed* / 'sangat macet' (*hj*), dimana persentasenya (D_p) didapat dari jumlah kendaraan kecil (V_1),

kendaraan sedang (V_2), dan kendaraan besar (V_3) yang didapat secara acak berdasarkan rumus (4). Untuk jumlah *node* macet $t = 0$, persentase kepadatan jalannya berstatus Normal (*nl*) untuk membandingkan hasil dari *t* selanjutnya.

Setelah melakukan pengujian sebanyak 200 kali dengan inisiasi *i* dan *j* yang sama, serta persentase kepadatan jalan dari modul sensor (D_p) secara acak dengan batas maksimal $t = 5$, skenario dari pengujian yang didapat adalah sebagai berikut.

1. Pencarian dengan *A** dimana semakin banyak *node* yang mengalami kemacetan (*random*), semakin sedikit jumlah *node* atau jarak tempuh semakin pendek. Dengan kata lain, hasilnya lebih optimal dibanding hanya menggunakan *A** saja.

   Hal ini disebabkan karena *node* acak berstatus 'sangat macet' yang tidak dilalui *agent* (ada *constraint backtrack*) memperbesar jarak tempuh *agent*, walaupun *node* tersebut merupakan *node* terdekat dari *current node* (posisi dari *agent*).

Tabel 2 Hasil pengujian dari skenario pertama

i = Universitas Multimedia Nusantara *j* = Summarecon Mall Serpong				
	D_p (%)	Status	N	X_{ij} (m)
t=0	0	nl	77	26304
t=1	40	j	77	26304
	80	hj	34	10241
t=2	38	j	34	10241
	77	hj	21	5914
t=3	35	j	21	5914
	77	hj	19	5060
t=4	45	j	19	5060
	89	hj	19	5060
t=5	45	j	19	5060
	98	hj	17	4808

2. Pencarian dengan *A** dimana semakin banyak *node* yang mengalami kemacetan (*random*), semakin banyak jumlah *node* atau jarak tempuh semakin panjang.

   Hal ini disebabkan karena *node* dengan *constraint backtrack* tersebut memutus hubungan dengan *node* terpendek berikutnya sehingga *node* yang terpilih menjauhi *node* tujuan / *goal*.

Tabel 3 Hasil pengujian dari skenario kedua

i = Universitas Multimedia Nusantara *j* = Summarecon Mall Serpong				
	D_p (%)	Status	N	X_{ij} (m)
t=0	0	nl	77	26304
t=1	36	j	77	26304
	78	hj	77	26822
t=2	60	j	75	26822
	79	hj	92	31086
t=3	56	j	75	26822
	77	hj	93	31241
t=4	54	j	75	26822
	99	hj	95	32826
t=5	59	j	75	26822
	100	hj	95	32826

3. Pencarian dimana jumlah *node* dan jarak tempuh berubah-ubah.

   Hal ini disebabkan karena *node* dengan *constraint backtrack* yang terpilih dapat memperkecil atau meperbesar jarak tempuh, seperti yang dijelaskan pada skenario pertama dan kedua.

Tabel 4 Hasil pengujian dari skenario ketiga

i = Universitas Multimedia Nusantara *j* = Summarecon Mall Serpong				
	D_p (%)	Status	N	X_{ij} (m)
t=0	0	nl	77	26304
t=1	34	j	77	26304
	74	hj	75	26822
t=2	51	j	77	26304
	93	hj	32	9951
t=3	44	j	75	26822
	94	hj	33	10051
t=4	60	j	75	26822
	87	hj	69	24304
t=5	41	j	75	26822
	90	hj	67	24052

4. Banyaknya jumlah *node* yang mengalami kemacetan tidak mengubah jumlah *node* maupun jarak tempuh.

   Hal ini disebabkan karena *node* dengan *constraint backtrack* yang terpilih kemungkinan merupakan *node* di luar jalur yang dilalui agent, sehingga *node* yang macet ini tidak memberikan fungsi apapun terhadap pencarian.

Tabel 5 Hasil pengujian dari skenario keempat

i = Universitas Multimedia Nusantara *j* = Summarecon Mall Serpong				
	D_p(%)	Status	N	X_{ij} (m)
t=0	0	nl	77	26304
t=1	34	j	77	26304
	90	hj	77	26304
t=2	53	j	77	26304
	98	hj	77	26304
t=3	66	j	77	26304
	78	hj	77	26304
t=4	63	j	77	26304
	82	hj	77	26304
t=5	37	j	77	26304
	82	hj	77	26304

5. Pencarian dimana *constraint* dari backtrack pencarian (*node* yang macet tidak dapat dilalui) menyebabkan siklus pencarian selamanya (*Infinity Loop*).

   Hal ini disebabkan karena adanya kemungkinan *node* yang macet merupakan satu-satunya *node* yang dapat dilalui oleh *agent* (tidak ada jalur lain).

Tabel 6 Hasil pengujian dari skenario kelima

i = Universitas Multimedia Nusantara *j* = Summarecon Mall Serpong				
	D_p(%)	Status	N	X_{ij} (m)
t=0	0	nl	77	26304
t=1	36	j	77	26304
	78	hj	∞	∞

V. SIMPULAN DAN SARAN

Berdasarkan hasil pengujian, dapat disimpulkan bahwa penggunaan metode *A** tidak menjamin bahwa rute yang terpilih adalah rute terbaik atau paling optimal (dilihat dari $t = 0$), karena *agent* tidak bersifat *full observable* setelah proses *backtrack* dilakukan (menutup *node* yang macet), sehingga hanya menghitung jarak dari *node* yang dilaluinya saja.

Selain itu, *constraint node* yang macet juga mempengaruhi hasil pencarian menggunakan *A**, dimana hasilnya dapat lebih optimal (skenario pertama) atau dapat lebih buruk (skenario kedua atau kelima).

Saran yang diberikan untuk mengembangkan solusi pencarian jalur yang lebih optimal yaitu menambahkan *knowledge base* untuk *agent* sehingga *agent* dapat mempelajari jalur terpendek sesuai input dari user, dan *constraint* yang dapat mengoptimalkan pencarian menggunakan *A**. Selain itu, diperlukan perbandingan pengujian di daerah lain atau perbandingan algoritma ini dengan algoritma lain sebagai pembuktian bahwa algoritma lebih optimal.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Yustinus Eko Soelistio, S.Kom, M.M. dan Seng Hansun,S.Si., M.Cs yang telah memberikan pengarahan dalam menulis jurnal ini, keluarga: orang tua dan saudara-saudara, rekan-rekan mahasiswa Universitas Multimedia Nusantara yang turut serta menjadi penyemangat dalam penelitian yang dilakukan penulis. Semoga jurnal ini dapat bermanfaat dalam memberi informasi bagi para pembaca.

DAFTAR PUSTAKA

[1] L. Fu, D. Sun, dan L.R. Rilett, "Heuristic Shortest Path Algorithm for Transportation Application : State of The Art," Computers & Operations Research, vol. 33, hal. 3324-2243, 2006.

[2] Masoud Nosrati, Ronak Karimi, dan Hojat Allah Hasanvand, "Investigation of the * (Star) Search Algorithms : Characteristics, Methods, and Approaches," World Applied Programming, vol. 2, no. 4, hal. 251-256, April 2012.

[3] Rengga Dionata Putra, Muhammad Aswin, dan Waru Djuriatno, "Pencarian Rute Terdekat Pada Labirin Menggunakan Merode *A**," Jurnal EECCIS, vol. 6, no. 2, Desember 2012.

[4] Cecil Ozkurt dan Fatih Camci, "Automatic Traffic Density Estimation and Vehicle Classification for Traffic Surveillance Systems using Neural Networks," Mathematical and Computational Applications, vol. 14, no. 3, hal.187-196, 2009.

[5] John van Rijn, "Road Capacity," belum terbit.