Seminar Nasional Sains & Teknologi Informasi (SENSASI)
SENSASI 2019

ISBN: 978-602-52720-2-8
Juli 2019
Hal: 259 – 264

Perbandingan Algoritma *Dijkstra* dan *Bellman-Ford* Dalam Pencarian Jarak Terpendek Pada SPBU

Akbar Serdano*, Muhammad Zarlis, Dedy Hartama

[1]Fakultas Ilmu Komputer dan Teknologi Informasi, Teknik Informatika S2, Univ. Sumatera Utara, Medan, Indonesia
Email: [1*] akbarserdano27@gmail.com

Abstrak

Jarak antar SPBU yang dilewati akan membentuk sebuah *graph*. Berdasarkan *graph* tersebut proses perhitungan dapat dilakukan menggunakan algoritma *dijkstra* dan *bellman-ford* untuk menentukan jarak spbu terpendek. Cara kerja algoritma *dijkstra* dan *bellman-ford* memakai strategi *greedy* yang cara kerjanya memilih sisi nilai bobot terkecil dengan menghubungkan jarak yang terpilih dengan jarak lain yang belum terpilih. Algoritma *dijkstra* dan *bellman-ford* membutuhkan parameter lokasi awal, dan tujuan sebagai masukan didalam proses. Hasil yang diberikan dari algoritma tersebut adalah memberikan jarak terpendek berserta rute dari lokasi awal ke tujuan. Penerapan algoritma *dijkstra* dan *bellman-ford* telah memberikan jarak terpendek sebagai solusi dalam penyelesaian masalah. Hasil penelitian ini menunjukkan bahwa algoritma *dijkstra* dapat memproses data lebih cepat dibandingkan algoritma *bellman-ford*.

Kata Kunci: *Dijkstra, Bellman-ford, Shortest Path Problem, Time Complexity, Node*

1. PENDAHULUAN

Kendaraan darat merupakan alat transportasi utama sebagai penunjang kegiatan sehari-hari yang membutuhkan bahan bakar. Untuk itu pengendara perlu mengetahui lokasi stasiun Pengisian Bahan Bakar Umum (SPBU) yang terpendek jika kehabisan bahan bakar untuk melakukan isi ulang. Pencarian lokasi SPBU tentunya berdasarkan jarak-jarak yang dapat dilewati sehingga dapat memperkirakan dan menentukan jarak yang terpendek menuju SPBU[1].

Dalam Ilmu komputer untuk menentukan jarak terpendek diperlukan suatu algoritma. Diantaranya adalah algoritma dijkstra dan bellman-ford yang sering digunakan untuk mencari solusi pencarian jarak terpendek. Kedua algoritma ini mudah untuk diimplementasikan kedalam permasalahan pencarian jalur terpendek[2]. Algoritma ini adalah algoritma mencari jarak terpendek dari sebuah lintasan yang saling terhubung menggunakan prinsip greedy dengan memilih berbobot minimum[3].

Dari penelitian yang telah dilakukan oleh peneliti yang dituliskan dalam jurnal atau karya ilmiah tentang pencarian SPBU terpendek dan penentuan jarak terpendek menggunakan algoritma *dijkstra*[4], dalam penelitian ini hanya menentukan jarak terpendek sebagai rekomendasi yang akan ditempuh. Maka dari itu dikembangkan penelitian baru menggunakan algoritma *dijkstra* dan algoritma *bellman-ford* untuk pencarian SPBU dengan jarak terpendek dan memberikan perbandingan dari kedua algoritma tersebut yang mana lebih baik. Kedua algoritma tersebut tentunya memiliki metode yang berbeda dalam menyelesaikan masalah sehingga perlu dilakukan perbandingan.

2. LANDASAN TEORI

2.1 Jarak Terpendek (*Shortest Path Problem*)

Proses penghitungan rute terpendek merupakan proses dimana untuk mendapatkan jarak yang terpendek atau dengan biaya yang sangat kecil pada suatu rute dari *node* awal ke *node* tujuan dalam sebuah jaringan. Pada proses penghitungan rute terpendek terdapat dua macam proses yaitu proses pemberian label dan proses pemeriksaan *node*. Metode pemberian label adalah metode untuk memberikan identifikasi pada setiap *node* dalam jaringan. Pada sebagian besar algoritma penghitungan rute terpendek, terdapat 3 label informasi yang dikelola untuk setiap *node* i pada proses pemberian label yaitu: label jarak d(i), parent *node* p(I), dan status *node* S(i)[5]. Algoritma untuk mencari jarak terpendek sudah banyak yang meneliti. Beberapa algoritma yang dapat digunakan untuk menyelesaikan penentuan jarak terpendek adalah algoritma *dijkstra*, algoritma bellmanford, algoritma a* , dan algoritma floyd–warshall[6].

2.2 Algoritma *Dijkstra*

Algoritma *dijkstra* merupakan salah satu bentuk algoritma greedy. Algoritma ini termasuk algoritma pencarian untuk menyelesaikan masalah jarak terpendek dengan satu tujuan pada sebuah lintasan yang tidak memiliki cost sisi *negative*[6]. Algoritma *dijkstra* menggunakan *adjacent list* untuk merepresentasikan sebuah jalur yang akan dilewati.

2.2 Algoritma *Bellman-ford*

Algoritma *bellman-ford* dikembangkan oleh Richard Bellman and Lester Ford, Jr. Algoritma ini sangat mirip dengan algoritma *dijkstra* namun algoritma ini mampu menangani bobot negatif pada pencarian jarak terpendek pada sebauh *graph* berbobot. Algoritma *bellman-ford* merupakan pengembangan dari algoritma *dijkstra*, algoritma *bellman-ford* akan benar jika dan hanya jika *graph* tidak terdapat cycle dengan nilai bobot negatif yang dicapai dari sumber. Ciri-ciri algoritma *bellman-ford*[6]:

1. Mengeksekusi walaupun terdapat *edge* dengan bobot negatif.
2. Harus directed *edge* (jika tidak *graph* akan memilik cycle dengan bobot negatif).
3. Iterasi i menemukan seluruh shortest path dengan menggunakan i *edge*.
4. Dapat mendeteksi cycle dengan nilai bobot negatif jika ada.

ISBN: 978-602-52720-2-8
Juli 2019
Hal: 259 – 264

3 METODE PENELITIAN

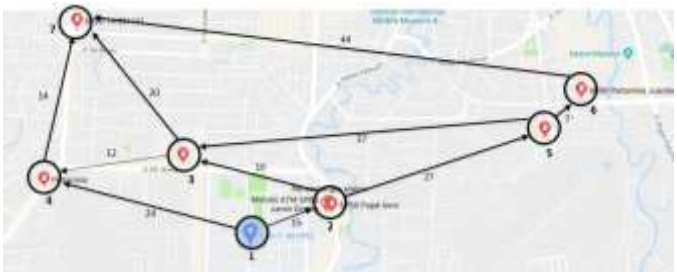
Pencarian jarak terpendek menggunakan algoritma *dijkstra* dan *bellman-ford* akan dikaji melalui kriteria unjuk kerja dari kedua algoritma tersebut. Ada empat kriteria dalam mengevaluasi kinerja algoritma yaitu :

1. *Completeness* : menilai bagaimana sebuah algoritma mampu menjamin ketersediaan solusi ketika solusi tersebut memang ada. Sehingga harus menjamin menemukan jarak terpendek dari titik start.
2. *Optimality* : menjukan kemampuan algoritma dalam menemukan solusi jarak terpendek secara optimal.
3. *Time Complexity* : menunjukkan seberapa lama sebuah algoritma menemukan solusi rute terpendek dalam pencarian jarak terpendek dari titik start ke lokasi tujuan.
4. *Space Complexity* : menunjukkan seberapa besar memory yang akan dibutuhkan oleh algoritma pencarian dalam menemukan jarak terpendek menuju lokasi SPBU.

4. HASIL DAN PEMBAHASAN

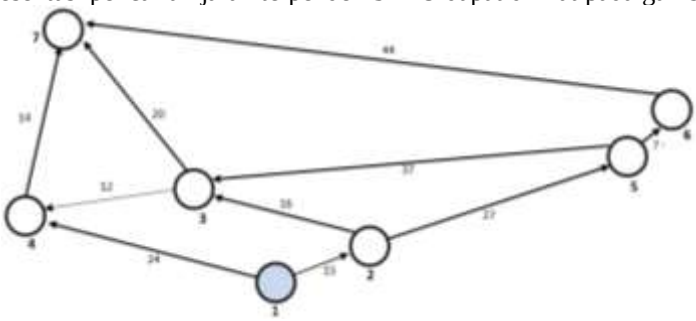
Dalam menentukan jarak SPBU terpendek dilakukan perhitungan dari jarak lokasi awal ke lokasi masing-masing SPBU. Proses perhitungan ini akan menghasilkan jarak SPBU terpendek dari posisi awal. Proses ini membutuhkan parameter lokasi awal dan parameter tujuan SPBU sekitar sehingga dihasilkan perbandingan jarak antar SPBU.

Perhitungan dilakukan dengan menentukan panjang pada masing-masing jalan yang dilalui pada SPBU sekitar. Parameter yang perlu diperhatikan adalah kendaraan yang akan digunakan beserta jarak pada jalur yang akan dilalui. Perhitungan ini dilakukan dengan representasi ke dalam sebuah *graph*. Jalan antar posisi asal dengan SPBU digambarkan dengan *edge*, sedangkan titik SPBU beserta posisi asal digambarkan dengan *vertex*. Berikut Skema Sistem dalam Algoritma *Dijkstra* dan *Bellman-ford* :



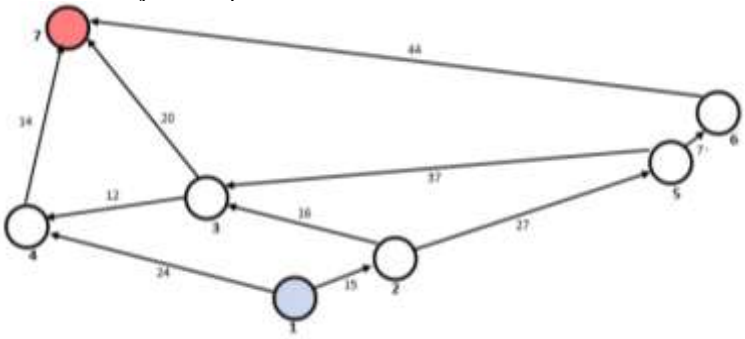
Gambar 1. Skema Sistem dalam Algoritma *Dijkstra* dan *Bellman-ford*

Notasi *graph* hasil representasi pencarian jarak terpendek SPBU dapat dilihat pada gambar dibawah ini.



Gambar 2. Awal Notasi *Graph*

Berikut *graph* dalam menentukan jarak terpendek dari titik 1 ke titik 7.

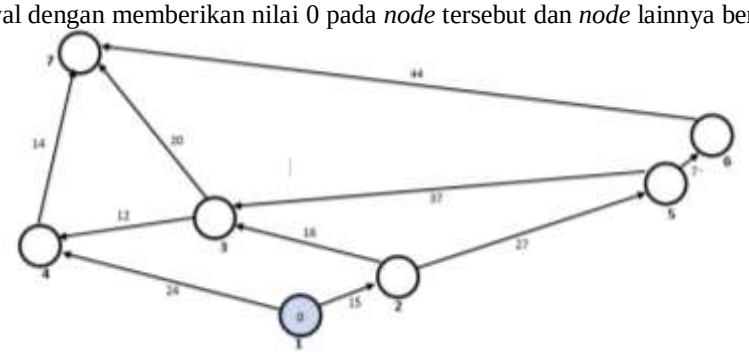


Gambar 3. *Graph* untuk Menentukan Jarak Terpendek dari Titik 1 Ke Titik 7

Ada beberapa jarak yang mungkin dilalui misalnya 1 -> 4 -> 7, atau 1 -> 3 -> 7, dan lain-lain. Tetapi cara yang efektif adalah :

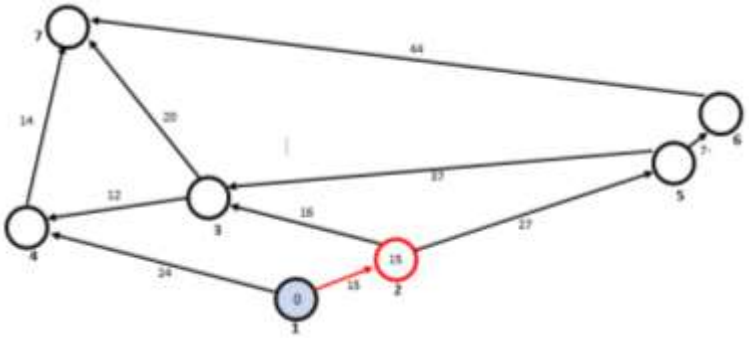
ISBN: 978-602-52720-2-8
Juli 2019
Hal: 259 – 264

a. Menentukan *node* awal dengan memberikan nilai 0 pada *node* tersebut dan *node* lainnya bernilai kosong.



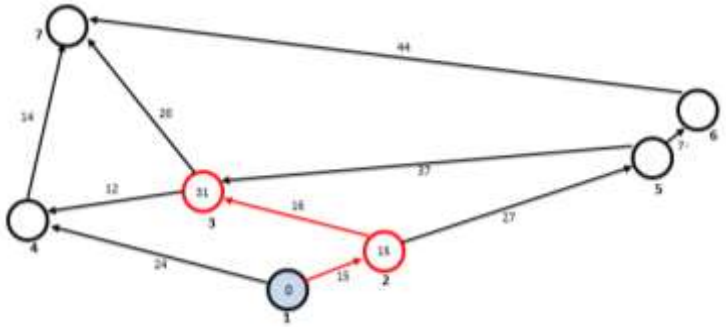
Gambar 4. Inisialisasi Node Awal dan Node Lainnya

b. Menentukan nilai bobot yang paling kecil untuk meneruskan ke *node* selanjutnya. Pada gambar diketahui bahawa untuk melangkah ke *node* 2 memiliki bobot terkecil, kemudian memberikan nilai atau label pada *node* 2 adalah 15.

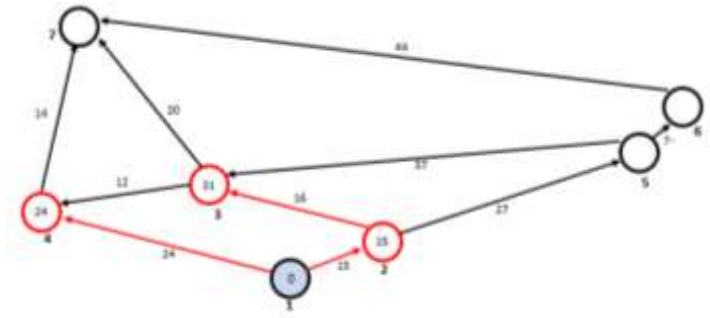


Gambar 5. Menentukan Nilai Terkecil dari *Node* untuk Berpindah ke *Node* Selanjutnya

c. *Node* 3 dengan *node* 4 memiliki nilai yang berbeda yang mana *node* 4 memiliki bobot minimum dari pada *node* 3. Maka ditentukanlah *node* yang mana paling efektif dari kedua *node* tersebut.



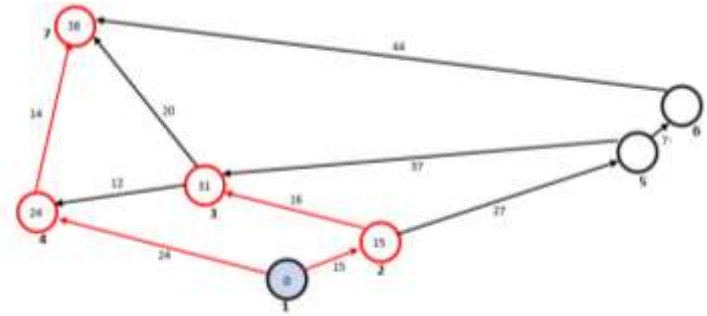
Gambar 6. Menentukan *Node* Kedua untuk Berpindah *Node* Berikutnya



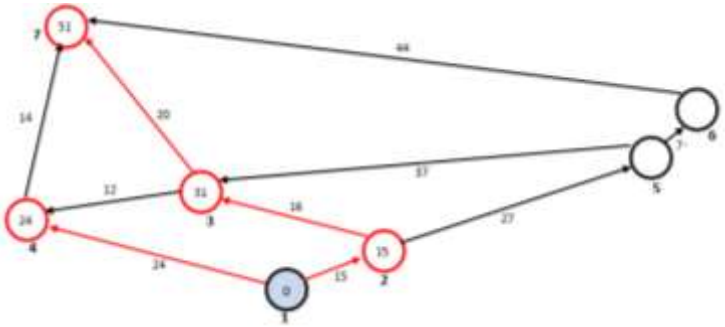
Gambar 7. Menentukan *Node* yang Nilai Bobotnya Terkecil yang akan Dilalui

d. Pada *node* 7 terdapat 2 jarak yang memiliki bobot berbeda, dimana *node* 7 bisa dilalui *node* 3 dan *node* 4 yang memiliki bobot minimum. Selanjutnya dilakukan pencarian jarak yang efektif dari kedua jarak tersebut.

ISBN: 978-602-52720-2-8
Juli 2019
Hal: 259 – 264



Gambar 8. Menentukan *Node* Keempat yang dapat Dilintasi *Node* Selanjutnya



Gambar 9. Menentukan *Node* yang Nilai Bobotnya Terkecil yang akan Dilewati Hingga ke Titik Tujuan

Penentuan jarak terpendek pada gambar diatas dengan menggunakan algoritma *dijkstra* menghasilkan rute terpendek 1 -> 4 -> 7. Berikut perhitungan penyelesaiannya dapat dilihat pada Tabel 1.

Tabel 1. Hasil Perhitungan Algoritma *Dijkstra* dan *Bellman-ford*

No.	*Vertex* Sumber	Tujuan	Jarak dengan *Vertex* lainnya		
1	1	1 -> 2	0	15	15
2		1 -> 4	0	24	24
3	2	2 -> 3	15	16	31
4		2 -> 5	15	27	42
5	3	3 -> 4	31	12	43
6		3 -> 7	31	20	51
7	4	4 -> 7	24	14	38
8	5	5 -> 3	42	37	79
9		5 -> 6	42	7	49
10	6	6 -> 7	49	44	93

Setelah dilakukan perhitungan maka dapat mengukur kinerja dari masing-masing agloritma untuk mendukung keputusan. Ada empat jenis kreteria untuk mengevaluasi kinerja dari algoritma tersebut dalam menentukan pengambilan keputusan untuk menentukan jarak terpendek.

1. Kelengkapan (*Completeness*)
   Tahapan ini dimana mengukur kinerja algoritma dari sisi kelengkapan untuk menyelesaikan solusi. Kelengkapan tersebut dari ketersediaan jarak yang dimulai dari titik awal menuju simpul yang berhubungan. Ketersedian solusi dari algoritma mampu menentukan jarak terpendek dari setiap simpul pada titik koordinat yang telah ditentukan pada Google Maps. Kelengkapan termasuk mencapai dari titik simpul akhir dari *graph* yang digambarkan.
2. Optimalitas (*Optimality*)
   Tahapan ini merupakan kemampuan dari algoritma untuk menemukan solusi jarak terpendek yang optimal. Kreteria ini dimana menjamin ketersedian solusi jarak terpendek, sedangkan pada kerteria optimalitas menemukan solusi jarak terpendek dari ketersediaan solusi jarak terpendek. Dari solusi yang dihasil maka dapat membandingkan hasil dari ketersediaan pada masing-masing solusi. Untuk menemukan jarak yang menuju titik akhir dari simpul yang telah digambarkan dalam bentuk *graph*. Berikut contoh hasil jarak terpendek dari titik 1 ke 7.
3. Kompleksitas Waktu (*Time complexity*)
   Evaluasi dari komplesitas waktu menunjukan waktu yang dibutuhkan oleh algoritma dalam menemukan solusi jarak terpendek. Kompleksitas waktu pada penentuan jarak SPBU terpendek dapat dilihat dari suatu program. Berikut hasil perbandingan kompleksitas dari algoritma *dijkstra* dan algoritma *bellman-ford*.

ISBN: 978-602-52720-2-8
Juli 2019
Hal: 259 – 264

Tabel 2. Perbandingan Waktu yang Dibutuhkan Algoritma Diijkstra dan Bellman-ford

No.	*Dijkstra* (s)	*Bellman-ford* (s)
1	0,5	1,6
2	0,5	1,5
3	0,5	1,4
4	0,5	1,5
5	0,5	1,4
6	0,5	1,4
7	0,5	1,4
8	0,5	1,5
9	0,5	1,4
10	0,5	1,5
Jumlah	5	14,6
Rata-rata	0,5	1,46

4. Kompleksitas Waktu Algoritma *Dijkstra*
   Untuk himpunan simpul pada sisi jumlah simpul sebesar V dan jumlah sisi sebesar E, dapat menentukan kompleksitas waktu dari algoritma *dijkstra* dengan notasi Big-O, yaitu O (| V | 2 + | E |) = O (| V | 2), jika algoritma merepresentasikan simpul dan sisi dalam keadaan bentuk list maupun array. Algoritma dapat lebih efesien dengan menyimpan *graph* dalam bentuk adjacency list dan menggunakan binary heap sebagi priority queue. Didapatkan dalam notasi Big-0, yaitu O ((| E | + | V |)log | V |).
5. Kompleksitas Waktu Algoritma *Bellman-ford*
   Secara kompleksitas waktu algoritma *bellman-ford* memiliki satu kemungkinan, yaitu dengan notasi Big-O diberikan O(|V.E|). V adalah jumlah *vertex* atau simpul dalam *graph* berbobot, lalu E adalah *edge*s yang merupakan jumlah sisi dalam *graph*. Oleh sebab itu jumlah sisi ataupun simpul yang sangat besar akan menyebabkan algoritma ini akan berjalan lebih lama dibandingkan dengan algoritma *dijkstra*. Berikut penjabaran perhitungan kompleksitas yaitu algoritma *bellman-ford*:
   (i) Tahap inisialisasi mempunyai kompleksitas O(V).
   (ii) Tahap kedua yaitu melakukan pencarian jarak atau jalan terpendek terhadap suatu simpul s mempunyai kompleksitas O(V.E)
   (iii) Tahap ketiga yaitu pengecekan ada atau tidak sisi *negative* pada jarak, menpunyai kompleksitas O(E).
   Dari semua kompleksitas tersebut dapat ditentukan kompleksitas waktu algoritma ini adalah O(V.E).
6. Kompleksitas ruang (*Space complexity*)
   Besar beban memori akan mempengaruhi kinerja dari algoritma untuk mencari solusi jarak terpendek. Penggunaan memori akan ditentukan dengan variable yang didefinisikan. Variable akan menggunakan ruang pada memori yang akan mempengaruhi kinerja dari algoritma.

Pada algoritma *dijkstra*, kompleksitas waktu yang dimiliki lebih kecil dibandingkan algoritma *bellman-ford*. Untuk kasus terburuk, kedua algoritma ini dapat memiliki kompleksitas waktu yang sama. Kompleksitas waktu algoritma menjadi sorotan paling penting ketika menghitung waktu program. Program yang baik akan mempunyai kompleksitas waktu yang paling kecil sehingga dengan cepat melakukan perhitungan dalam menyelesaikan suatu masalah.

4. KESIMPULAN

Berdasarkan hasil yang dilakukan, maka dapat disimpulan sebagai berikut:
1. Algoritma *dijkstra* dan *bellman-ford* bertujuan untuk menentukan jarak terpendek dalam suatu masalah. Kedua algoritma ini masuk ke dalam kategori strategi greedy yang cara kerjanya memilih sisi bobot yang terecil yang mengghubungkan jarak yang sudah terpilih dengan jarak yang lain belum terpilih.
2. Dari analisa kompleksitas waktu terhadap kedua algoritma, *dijkstra* membutuhkan waktu untuk memproses data lebih pendek daripada algoritma *bellman-ford*.
3. Dalam mengukur suatu kinerja setiap algoritma dapat menggunakan empat kreteria yaitu kelengkapan (*completeness*), optimalitas (*optomality*), kompleksitas waktu (*time complexity*) dan kompleksitas ruang (*space complexity*). Setiap kreteria berfungsi untuk menganalisis kinerja dari algoritma untuk mendapatkan hasil solusi yang terbaik. Dalam kedua algoritma tersebut memiliki kesamaan dalam kelengkapan (*completeness*), optimalias (*optomality*), dan kompleksitas ruang (*space complexity*), tetapi dalam kompleksitas waktu (*time complexity*) berbeda saat mengeksekusi program.

REFERENCES

[1] S. Arifianto, "Sistem Aplikasi Penentuan Rute Terpendek pada jaringan Multi Moda Transportasi umum menggunakan Algoritma Dijkstra," *Sist. Apl. Penentuan Rute Terpendek pada Jar. Multi Moda Transp. umum menggunakan Algoritm. Dijkstra*, 2012.
[2] Y. M. Zhang and L. Ma, "The optimal path of logistics distribution in electronic commerce," in *2nd International Conference on Information Science and Engineering, ICISE2010 - Proceedings*, 2010.
[3] A. Gusmão, S. H. Pramono, and Sunaryo, "Sistem Informasi Geografis Pariwisata Berbasis Web Dan Pencarian Jalur Terpendek Dengan Algoritma

ISBN: 978-602-52720-2-8
Juli 2019
Hal: 259 – 264

Dijkstra," *J. Electr. Electron. Commun. Control. Informatics, Syst.*, 2013.
[4] W. E. Y. R, D. Istiadi, and A. Roqib, "Pencarian Spbu Terdekat Dan Penentuan Jarak Terpendek Menggunakan Algoritma Dijkstra," *J. Nas. Tek. Elektro*, 2015.
[5] Y. Purwananto, D. Purwitasari, and W. A. Wibowo, "implementasi dan Analisis Algoritma Pencarian Rute Terpendek di Kota Surabaya," *J. Penelit. dan …*, 2005.
[6] B. K. Shivani Sanan, Leena jain, "Shortest Path Algorithms: A Comparison," *Oper. Res.*, vol. 2, no. 7, pp. 316–320, 2013.