ISSN 2355-3286

# Pengukuran Beban Komputasi Algoritma Dijkstra, A*, dan Floyd-Warshall pada Perangkat Android

Michael Alexander Djojo, Karyono
Program Studi Sistem Komputer, Universitas Multimedia Nusantara, Tangerang, Indonesia
djojo.alexander@gmail.com, karyono@umn.ac.id

Diterima 2 Mei 2013
Disetujui 15 Mei 2013

**Abstrak—**Perkembangan teknologi di bidang komunikasi menciptakan berbagai kemudahan bagi pengguna untuk melakukan pertukaran informasi tanpa mengenal jarak secara geografis. Pada jaringan komunikasi, pertukaran informasi memerlukan pengaturan rute sehingga dicapai jalur terpendek untuk mengoptimalkan proses pengiriman data. Penelitian untuk mencari algoritma jalur terpendek masih terus dilakukan. Penelitian ini membandingkan algoritma Dijkstra, A*, dan Floyd-Warshall dari sisi waktu, beban komputasi dan penggunaan memori. Topologi yang digunakan dalam penelitian adalah topologi jaringan *mesh* karena dapat mewakili kondisi nyata. Aplikasi berbasis Android dapat digunakan sebagai simulator untuk memetakan *vertice* dan *edge* ke dalam kumpulan *node* dan *channel* yang saling berhubungan. Kompleksitas komputasi dalam pencarian jalur terpendek menjadi hal yang penting karena terdapat keterbatasan prosesor dan memori. Kompleksitas rute akan sebanding dengan skala jaringan *mesh*. Dari simulasi diperoleh nilai beban komputasi dan waktu simulasi yang sebanding dengan fungsi kuadrat jumlah simpul untuk ketiga algoritma tersebut. Hasil pengujian menunjukkan algoritma A* memiliki beban komputasi dan waktu simulasi yang paling kecil dibandingkan algoritma Dijkstra dan Floyd-Warshall tanpa mempengaruhi hasil pencarian rute terpendek. Hal ini disebabkan algoritma A* melakukan operasi pencarian dengan memanfaatkan nilai heuristik terhadap simpul tujuan, sehingga tidak semua simpul dilakukan pengecekan. Namun algoritma Dijkstra paling unggul dalam penggunaan memori. Floyd-Warshall menghasilkan nilai kompleksitas yang buruk pada proses pancarian jalur, semua data bobot kanal akan ditampung ke dalam matriks dua dimensi lalu diproses menggunakan operasi perulangan yang bertingkat.

**Kata kunci—***shortestpath*, *dijkstra*, *a**, *floyd-warshall*, *weighted graph*, *mesh*, *android*

## I. Pendahuluan

*Shortest Path Problem* merupakan salah satu masalah optimasi yang hingga saat ini masih terus dipelajari dengan aplikasi di berbagai bidang [1]. Algoritma Dijkstra, A*, dan Floyd-Warshall merupakan beberapa algoritma *shortest path* yang dipergunakan dalam kebutuhan pencarian jalur terpendek. Dijkstra dipergunakan dalam protokol routing *Open Shortest Path First* (OSPF) dalam menentukan rute terpendek pada proses pencarian jalur komunikasinya [2]. Algoritma A* dapat dimanfaatkan untuik menghasilkan perhitungan dan pemilihan rute dalam sistem penentuan lokasi dan rute perjalanan yang dikenal dengan *Traffic Navigational System* [3]. Selain itu dalam jaringan komunikasi, terdapat perangkat *switch* yang dapat memanfaatkan algoritma Floyd-Warshall sebagai penentuan jalur penyebaran informasi [1]. Pada jaringan komunikasi, ketepatan pemilihan rute perpindahan paket data dapat menjadi kunci dalam memperoleh optimasi kecepatan. Penelitian beban komputasi algoritma Dijkstra, A*, dan Floyd-Warshall menggunakan model jaringan *mesh* dilakukan untuk menemukan perbandingan dalam implementasinya pada perangkat seluler berbasis Android.

## II. Shortest Path Problem

*Shortest Path Problem* dapat didefinisikan untuk graf berarah, tanpa arah, atau gabungan. Permasalahan jalan terpendek yang difinisikan pada penelitian ini difokuskan pada keadaan topologi jaringan *mesh*, sehingga graf yang digunakan merupakan graf berarah. Graf berarah mengharuskan simpul untuk berturut-turut dihubungkan melalui kanal ke tujuan yang tepat.

### *A. Dijkstra*

Metode pelabelan Dijkstra merupakan prosedur utama dalam algoritma tersebut [1]. Keluaran dari metode pelabelan berupa rangkaian alur dari simpul sumber ke suatu set simpul lain. Alur keluaran merupakan jarak terpendek dari simpul sumber yang dapat ditemukan oleh algoritma.

Tiga informasi penting yang diperlukan untuk setiap simpul dalam metode pelabelan diantaranya

1. label jarak,
2. label simpul sebelumnya, dan

3. set simpul yang berlabel permanen.

Label jarak menyimpan batas jarak terpendek dari sumber ke tujuan, sedangkan label simpul sebelumnya mencatat indeks simpul yang dilalui. Jika simpul belum ditambahkan ke dalam set simpul, maka dianggap belum terjangkau. Biasanya label jarak simpul yang belum terjangkau diatur bernilai tak terbatas. Ketika diketahui bahwa jalan terpendek dari suatu simpul ke simpul lain merupakan benar-benar jalan terpendek, maka simpul tersebut disebut label permanen. Namun, jika masih terdapat kemungkinan ditemukannya jalan terpendek yang berasal dari simpullain, maka simpul tersebut dianggap hanya berlabel sementara.

Nilai dari label jarak merupakan batas atas jarak jalur terpendek ke suatu simpul. Jika simpul untuk sementara diberi label dan nilai label jarak menjadi jarak jalan akhir yang terpendek maka simpul tersebut akan dimasukkan dalam set simpul berlabel permanen. Secara iteratif menambahkan simpul berlabel sementara dengan label jarak terkecil ke set simpul berlabel permanen, algoritma Dijkstra dapat menjamin optimalitas dari pencarian. Keuntungan dengan pelabelan pada algoritma Dijkstra adalah pencarian tersebut dapat dihentikan ketika semua simpul tujuan secara permanen telah terlabel.

### *B. Simulator*

Simulator adalah model dari suatu sistem yang digunakan untuk menguji dan mempelajari sistem sebelum diimplementasikan pada dunia nyata [4]. Simulator saat ini kebanyakan berupa perangkat lunak sehingga meminimalisasi biaya pemodelan. Perangkat lunak simulasi dapat juga digunakan sebagai permainan seperti simulasi penerbangan, simulasi berkendara, dan sebagainya.Simulator diharapkan dapat meminimalisasi resiko kegagalan dalam penerapannya di dunia nyata.

### *C. Android Sebagai Simulator*

Android adalah sistem operasi berbasis Linux yang dirancang untuk perangkat seluler dan tablet [5]. Aplikasi berbasis Android dapat digunakan sebagai simulator untuk menemukan beban komputasi dari algoritma pencarian jalur terpendek pada model jaringan *mesh*. Pemodelan jaringan *mesh* menggunakan aplikasi berbasis Android dilakukan dengan mengimplementasikannya pada komponen simpul untuk membentuk suatu susunan jaringan.

### *D. Perangkat Pembangun Aplikasi*

Dalam penelitian pemodelan algoritma *shortest path*, dibutuhkan perangkat pendukung pembuatan aplikasi yaitu perangkat keras dan lunak. Berikut adalah konfigurasi perangkat komputer yang digunakan selama proses pembuatan model dan simulasi.

Perangkat Keras

a. Prosesor: Pentium Dual Core T4200 2.00 GHz,

b. Memori: 2.93 GB,

c. Harddisk: 250 GB.

Perangkat Lunak

a. Sistem operasi: Windows 8 Pro,

b. Eclipse IDE for Java Developer version Helios Service Release 2,

c. Android SDK Manager revision 21.0.1.

Penelitian ini diimplementasikan pada perangkat seluler berbasis Android dengan konfigurasi perangkat sebagai berikut.

a. Prosesor: Dual Core 1.2 GHz Cortex A9,

b. Memori: 1 GB,

c. Harddisk: 32 GB,

d. Sistem operasi: Android versi 4.1.2 (Jelly Bean).

## III. Rancangan Simulasi dan Hasil yang Dicapai

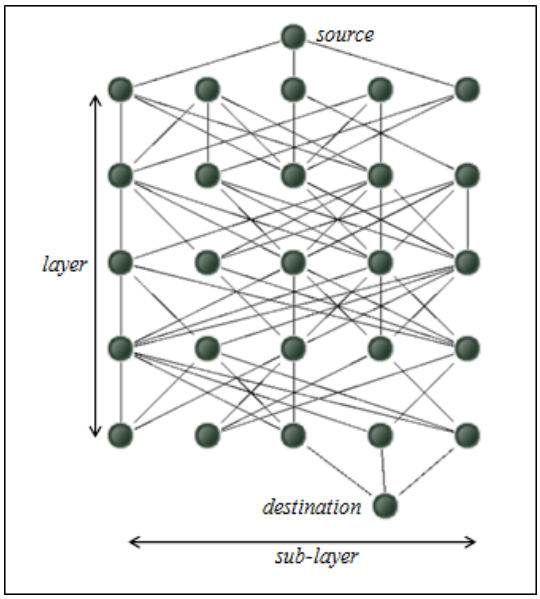
Penelitian terhadap algoritma Dijkstra, A*, dan Floyd-Warshall pada jaringan *mesh* ini dimodelkan menggunakan Eclipse IDE for Java Developer untuk menghasilkan aplikasi dan kemudian dapat diperoleh hasil beban komplesitasnya pada perangkat Android. Simpul terbagi menjadi tiga kategori, yaitu simpul *source*, *node*, dan *destination*. Simpul *source* merupakan titik awal pengiriman pesan dan simpul *destination* merupakan titik tujuan. Simpul*node* dibuat sebagai *array* yang disusun berlapis dengankanal penghubung yang memiliki bobot.

Jaringan *mesh* dibangkitkan menggunakan hubungan acak antar simpul, dimana terdapat simpul yang terhubung maupun yang tidak terhubung. Hal ini menyebabkan jaringan dapat terbentuk secara acak. Pembangkitan jaringan *mesh* dilakukan secara manual yaitu dengan mendefinisikan jumlah lapisan dan jumlah sub-lapisan simpul *node* yang membentuk jaringan penghubung dari simpul *source* ke *destination*. Nilai probabilitas yang digunakan untuk mendefinisikan hubungan antar *node* adalah 65% sehingga meminimalisasi kemungkinan tidak adanya jalan keluar.

### *A. Rancangan Jaringan Mesh*

Jaringan *mesh* dipetakan ke dalam model simulasi dengan konsep berlapis. Jaringan dibangkitkan dengan menerima masukan berupa jumlah lapisan dan sub-lapisan. Kanal pada jaringan *mesh* terbentuk secara urut mengikuti lapisan simpul dari titik *source* ke *destination*. Beban kanal yang diberikan memiliki rentang antara 1 hingga 10 yang dibangkitkan secara acak. Banyaknya simpul yang dibentuk mempengaruhi

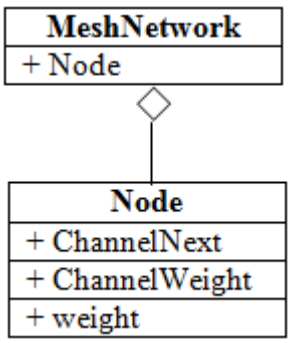
jumlah ketersediaan kanal.Kemungkinan terbentuknya kanalantar dua *node*yang digunakan penulis dalam membangkitkan jaringan *mesh* adalah 65%. Dari beberapa percobaan yang dilakukan, nilai persentase ini dianggap sebagai persentase optimal dalam menciptakan keacakan jaringan dengan jumlah yang relatif wajar. Banyaknya jumlah kanal menentukan banyak *gate* pada setiap simpul.




Gambar 1. Pemetaan topologi jaringan *mesh*

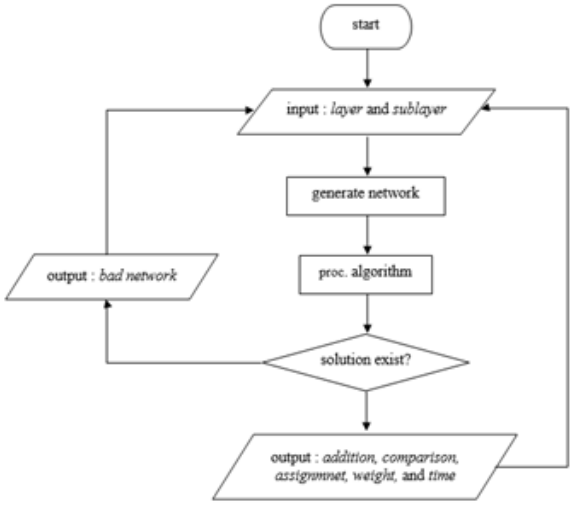
*B. Implementasi Rancangan pada Android*

Setiap simpul dimodelkan menjadi *class* yang digunakan sebagai objek dalam membentuk rangkaian jaringan *mesh* yang dapat dilihat pada *class diagram* berikut.




Gambar 2. *Class diagram* jaringan *mesh*

Aplikasi menerima *input* berupa parameter karaktersitik jaringan *mesh* yang akan dibangkitkan. Parameter tersebut diantaranya, *layer* yang menentukan banyaknya lapisan dalam jaringan dan *sublayer* yang menentukan banyaknya simpul pada setiap lapisan. Kedua parameter tersebut merupakan input berupa angka nonnegatif yang bersifat wajib untuk diisi. Validasi akan dilakukan untuk memastikan bahwa kedua parameter tersebut telah terisi oleh pengguna untuk dapat melanjutkan proses pencarian menggunakan algoritma shortest path. Gambar 3 menunjukkan flow chart aplikasi Android yang dirancang.




Gambar 3. *Flow chart* aplikasi pada perangkat Android

*C. Implementasi Algoritma Shoertest Path*

Algoritma Dijkstra, A*, dan Floyd-Warshall ditempatkan pada setiap simpul untuk dapat menemukan rute jaringan dengan bobot yang paling kecil.

Beberapa operasi yang dilakukan pada simpul diantaranya

1. membandingkan bobot kanal (*comparison*),
2. menjumlahkan bobot kanal (*addition*), dan
3. menyimpan informasi indeks simpul dan jumlah bobot (*assignment*).

Jumlah komputasi diperoleh dengan melakukan penghitungan terhadap banyak operasi yang dilakukan selama simulasi dijalankan [6].

Penghitungan kompleksitas komputasi dilakukan dengan cara menyisipkan beberapa baris perintah tambahan di antara *source code* algoritma yang menyebabkan penambahan nilai variabel untuk setiap operasi yang dilakukan. Variabel-variabel tersebut diantaranya penjumlahan (*addition*), pemasukan nilai (*assignment*), perbandingan (*comparison*), dan waktu simulasi (*time*). Algoritma Dijkstra, A*, dan Floyd-Warshall memiliki variabel yang terpisah sehingga penghitungan kompleksitas komputasi untuk setiap algoritma tersebut dapat dilakukan secara independen.

Simulasi untuk setiap algoritma tidak dilakukan secara bersamaan, tetapi secara bertahap dengan merancang penjadwalan eksekusi algoritma. Penjadwalan tersebut dilakukan agar tidak membebani perangkat saat melakukan simulasi secara bersama-

sama. Setiap algoritma akan diterapkan pada model jaringan *mesh* yang seragam. Proses penjadwalan tidak akan mempengaruhi bentuk hubungan simpul dan bobot kanal yang telah dibangkitkan oleh simulator.

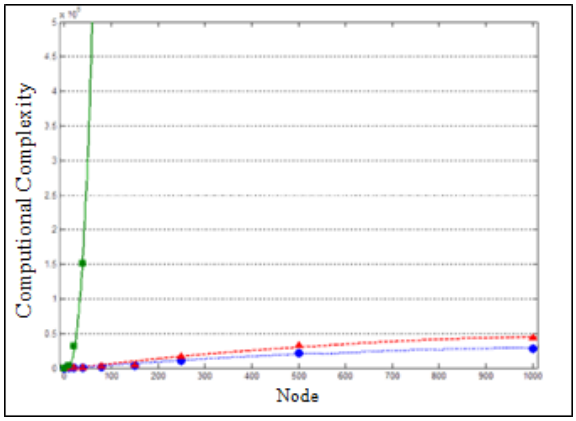
## IV. Hasil

Pengujian yang dilakukan terhadap perangkat seluler berbasis Android memperoleh hasil pengukuran algoritma shortest path sebagai berikut.

Tabel 1. Kompleksitas komputasi *shortest path* pada perangkat Android

| Node | Dijkstra | A* | Floyd-Warshall |
|---|---|---|---|
| 10 | 141 | 107 | 3547.8 |
| 20 | 357.2 | 269.2 | 31307.6 |
| 40 | 810.4 | 610.8 | 150703.6 |
| 80 | 2234.1 | 1578.7 | 1114818.2 |
| 150 | 4932.8 | 3521.1 | 6872435.8 |
| 250 | 16824.4 | 10388.8 | 32171572.6 |
| 500 | 32512.1 | 21890.7 | 253975134.2 |
| 1000 | 37771.3 | 27026.3 | 2014604822 |

Algoritma A* pada perangkat Android memiliki nilai kompleksitas komputasi yang paling kecil dapat dilihat pada Gambar 4.




Gambar 4. Perbandingan kompleksitas komputasi pada Android

Selain membandingkan dari segi komputasi, dapat diperoleh grafik perbandingan algoritma yang diuji dengan parameter waktu simulasi sebagai berikut.

Tabel 2. Waktu simulasi *shortest path* pada perangkat Android

| Node | Dijkstra | A* | Floyd-Warshall |
|---|---|---|---|
| 10 | 0.4 | 0.2 | 3547.8 |
| 20 | 0.9 | 0.2 | 31307.6 |
| 40 | 1 | 0.4 | 150703.6 |
| 80 | 1 | 0.6 | 1114818.2 |
| 150 | 1.4 | 1 | 6872435.8 |
| 250 | 8.2 | 3 | 32171572.6 |
| 500 | 13.9 | 12.1 | 253975134.2 |
| 1000 | 30.4 | 30.2 | 2014604822 |

*satuan dalam milisecond*

Berdasarkan hasil pengukuran, beban komputasi yang diperoleh dapat dilakukan pendekatan dengan fungsi kuadrat. Penambahan jumlah simpul (n-*node*) akan menghasilkan pengingkatan beban komputasi yang signifikan. Hal ini berarti, semakin banyak lokasi dan jarak yang harus diukurakan membutuhkan peningkatan memori dan prosesor yang signifikan pula.

Seperti halnya pada pengukuran beban komputasi, dalam pengukuran waktu simulasi diperoleh hasil yang dapat dilakukan pendekatan menggunakan fungsi kuadrat.

## V. Kesimpulan dan Saran

### *A. Kesimpulan*

Aplikasi berbasis Android mampu membangkitkan model jaringan *mesh*. Model jaringan mesh yang dihasilkan merupakan graf berbobot dengan beban kanal yang dibangkitkan secara acak pada rentang nilai 1 hingga 100. Parameter yang digunakan sebagai pembanding diantaranya, banyaknya operasi penjumlahan (*addition*), perbandingan (*comparison*), pemasukan nilai (*assignment*), dan waktu simulasi. Penelitian dilakukan dengan mengukur nilai parameter tersebut untuk algoritma Dijkstra, A*, dan Floyd-Warshall pada jaringan *mesh*. Jaringan memiliki jumlah simpul, kanal, dan beban kanal yang dibangkitkan secara acak.

Hasil penelitian mendapatkan bahwa algoritma A* memiliki kompleksitas komputasi yang paling kecil dibandingkan algoritma Dijkstra dan Floyd-Warshall. Pendekatan hasil dilakukan menggunakan fungsi kuadrat, sehingga diperoleh grafik perbandingan kompleksitas komputasi dan waktu simulasi antara ketiga algoritma yang diuji.

Algoritma *shortest path* kemudian diimplementasikan kepada perangkat seluler berbasis Android. Aplikasi dirancang dengan memodelkan jaringan mesh kedalam rangkaian objek. Aplikasi tersebut mampu melakukan penghitungan beban komputasi yang dihasilkan oleh algoritma *shortest path* selama proses pencarian jalur terpendek. Hasil pengujian memperoleh hasil bahwa algoritma A* unggul dalam hal kompleksitas komputasi maupun waktu simulasi.

### *B. Saran*

Terdapat banyak algoritma *shortest path* yang dapat diimplementasikan pada simulator berbasis

Android terutama dengan menggunakan model topologi *mesh*. Dengan melakukan pengujian terhadap algoritma yang berbeda dapat diperoleh perbandingan kompleksitas komputasi antar algoritma tersebut. Perbandingan beban komputasi algotirma dapat digunakan sebagai referensi dalam pengembangan aplikasi yang membutuhkan pencarian jalur terpendek.

## Daftar Pustaka


[1] Faramroze Engineer. 2005. *Fast Shortest Path Algorithms for Large Road Networks*. New Zealand: University of Auckland.

[2] Suherman, Eman, dkk.. 2011. *Simulasi Algoritma Dijkstra pada Protokol Routing Open Shortest Path*. Semarang: Universitas Diponegoro.

[3] Liu, Jingang dan Yujun Liu. 2010. *Application of A* Algorithm in Traffic Navigational System*. Information Engineering and Electronic Commerce (IEEC), 2010 2nd International Symposium.

[4] Banks, Carson, Nelson,& Nicol. 2001. *Discrete Event System Simulation.* Prentice Hall.

[5] *www.openhandsetalliance.com*, *Open Handset Alliance*. Diakses 18 Oktober 2013.

[6] Ruohonen, Keijo. 2008. Graph Theory. Canada: University of British Columbia.