Perbandingan Algoritma *Dijkstra* dan Algoritma *Bellman Ford* pada *Routing* Jaringan Komputer

Ginanjar Fahrul Muttaqin

Teknik Informatika Institut Teknologi Bandung, Ganeca 10,
e-mail: gin2_fm@yahoo.co.id

ABSTRAK

***Routing* adalah salah satu masalah dalam jaringan komputer. *Routing* berhubungan dengan pencarian jalur untuk pengiriman paket-paket data dalam jaringan. Makalah ini akan membahas bagaimana algoritma *Dijkstra* (pengembangan algoritma greedy) dan algoritma *Bellman Ford* (pengembangan program dinamis) diterapkan untuk menyelesaikan masalah ini. Makalah ini juga akan membahas bagaimana perbandingan, efisiensi dari kedua penggunaan algoritma *routing* tersebut.**

Kata kunci: *routing, link state, distance vector, cost*

1. PENDAHULUAN

Pengiriman paket-paket data adalah kunci utama dalam jaringan komputer. Pengirim dan penerima data berkomunikasi dalam sebuah topologi jaringan yang terdiri dari banyak simpul yang terhubung. Konsep ini sering disebut sebagai graf.

Hal yang paling penting adalah bagaimana mengirimkan paket data dari suatu simpul ke simpul lain secara efisien. Setiap paket yang dikirimkan dari simpul pengirim akan melewati simpul-simpul lain untuk mencapai simpul penerima. Dalam jaringan komputer istilah ini disebut sebagai penentuan *path* atau *route.* Penelusuran *path* itu dinamakan *routing*. Untuk menemukan jalur pengiriman paket-paket data tersebut digunakan beberapa algoritma yang populer, seperti algoritma *Dijkstra* dalam *link state routing* dan algoritma *Bellman Ford* dalam *distance vector routing*.

Kedua jenis *routing* tersebut berdasar pada algoritma sederhana yang telah disesuaikan dan dikembangkan. Algoritma *Dijkstra* sebenarnya adalah algoritma *greedy* karena pada dasarnya adalah pencarian jalur dengan meminimalkan *cost* pada setiap simpul yang ditemukan. Sedangkan algoritma *Bellman Ford* adalah pengembangan dari program dinamis karena pada dasarnya pencarian keputusan pada setiap tahap saling berhubungan.

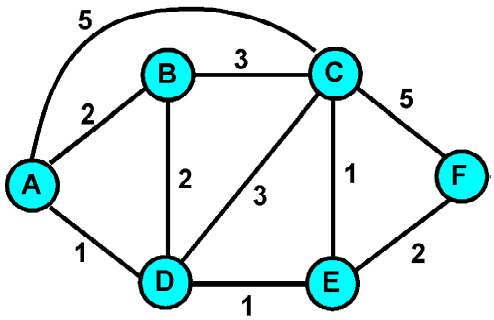
Jenis-jenis *routing* di atas memiliki keunggulan dan kelemahan sesuai dengan penggunaannya. Masing-masing tipe *routing*, baik *link state routing* maupun *distance vector routing* diterapkan untuk mendapatkan *path routing* yang efisien.

2. PRINSIP DASAR

2.1 *Routing* dalam Jaringan Komputer

Konsep dasar dalam *routing* adalah abstraksi dari sebuah graf yang memiliki beberapa komponen:

a. Simpul pengirim
   Simpul yang mengirimkan paket data ke simpul penerima. *Routing* dimulai saat simpul pengirim mengirimkan paket data dalam graf.
b. Simpul penerima
   Simpul yang menerima paket data dari simpul pengirim. *Routing* selesai setelah paket data diterima oleh simpul penerima.
c. *Forwarding Table*
   Sebuah tabel yang menyimpan informasi jalur pengiriman paket dari pengirim ke penerima pada setiap simpul yang dilewati saat *routing.*
d. *Cost*
   Besar usaha yang diberikan saat paket data dikirimkan dari sebuah simpul ke simpul tetangganya.
e. *Path*
   Informasi simpul-simpul yang dilalui pada saat pengiriman paket dari simpul pengirim ke simpul penerima



Gambar 1. Abstraksi Graf *Routing* pada Jaringan Komputer

Dengan mengamati gambar di atas, misalkan ada pengiriman paket data dari simpul A ke simpul F. Dalam hal ini:

a. Simpul pengirim adalah simpul A
b. Simpul penerima adalah simpul F
c. *Path* yang mungkin diperoleh:
   A → B → C → F, atau
   A → D → C → F, atau
   A → D → E → F, dst...
d. Jika *path* yang dipilih adalah A → B → C → F maka pada simpul B ada *forwarding table* F |(A,C), yang berarti untuk mengirimkan paket data dari A ke F melewati C.
e. *Cost* dari A → B = 2, B → C = 3, dst...

2.2 Algoritma Greedy

Algoritma greedy merupakan metode yang paling populer untuk memecahkan persoalan optimasi. Persoalan optimasi (*optimization problems*) adalah persoalan mencari solusi optimum.

Hanya ada dua macam persoalan optimasi:
1. Maksimasi (*maximization*)
2. Minimasi (*minimization*)

Prinsip greedy adalah "*take what you can get now!*". Algoritma greedy membentuk solusi langkah per langkah (*step by step*). Pada setiap langkah, terdapat banyak pilihan yang perlu dieksplorasi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Pada setiap langkah, kita membuat pilihan optimum lokal (*local optimum*) dengan harapan bahwa langkah sisanya mengarah ke solusi optimum global (*global optimum*).

Algoritma greedy adalah algoritma yang memecahkan masalah langkah per langkah. Mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan dan berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Elemen-elemen algoritma greedy:
1. Himpunan kandidat, C.
2. Himpunan solusi, S
3. Fungsi seleksi (*selection function*)
4. Fungsi kelayakan (*feasible*)
5. Fungsi obyektif

Dengan kata lain, algoritma greedy melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat,C yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan S dioptimisasi oleh fungsi obyektif.

2.3 Algoritma *Dijkstra* dalam *Link State Routing*

Algoritma *Dijkstra* pada dasarnya menggunakan prinsip *greedy*, yaitu dengan mencari minimum lokal untuk setiap tahap yang dilalui hingga didapat sebuah *link* antar simpul yang akan dijadikan pedoman untuk melakukan *routing*. Algoritma ini digunakan dalam *link state routing.*

Konsep dasar dalam *link state routing* adalah:

a. Menemukan tetangga dan mempelajari karakteristiknya.
b. Hitung *cost* untuk setiap tetangga.
c. Mengirimkan paket data hasil perhitungan *cost* tersebut kepada *router* tetangga.
d. Hitung rute terpendek untuk setiap router.

Untuk melakukan perhitungan tersebut perhatikan notasi-notasi berikut:

$$c(i,j) \rightarrow link\ cost\ dari\ simpul\ i\ ke\ simpul\ j \quad (1)$$

$$D(v) \rightarrow cost\ dari\ simpul\ pengirim\ ke\ simpul\ v \quad (2)$$

$$p(v) \rightarrow simpul\ tetangga\ sebelum\ v, cost\ terkecil \quad (3)$$

$$N \rightarrow himpunan\ simpul\ dengan\ shortest\ path \quad (4)$$

Jika *i* dan *j* tidak bertetangga maka nilai *c(i,j) = infty*. *D(v)* yang dihitung adalah cost terkecil.

Algoritma:

```
Initialization:
N = {A}
for all nodes v
      if v adjacent to A
            then D(v) = c(A,v)
      else D(v) = infty

loop
      find w not in N such that D(w) is
      a minimum

      add w to N
      update D(v) for all v adjacent to
      w and not in N:

      D(v) = min(D(v), D(w) + c(w,v))

      /* new cost to v is either old
      cost to v or known
      shortest path cost to w plus cost
      from w to v */

until all nodes in N
```

2.3 Program Dinamis

Program Dinamis adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah (*step*) atau tahapan (*stage*) sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan.

Pada penyelesaian persoalan dengan metode ini:

a. Terdapat sejumlah berhingga pilihan yang mungkin.
b. Solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya.
c. Kita menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

Prinsip Optimalitas: jika solusi total optimal, maka bagian solusi sampai tahap ke-k juga optimal. Prinsip optimalitas berarti bahwa jika kita bekerja dari tahap *k* ke tahap *k* + 1, kita dapat menggunakan hasil optimal dari tahap *k* tanpa harus kembali ke tahap awal. ongkos pada tahap *k* +1 = (ongkos yang dihasilkan pada tahap *k*) + (ongkos dari tahap *k* ke tahap *k* + 1).

Karakteristik program dinamis:

a. Persoalan dapat dibagi menjadi beberapa tahap (*stage*), yang pada setiap tahap hanya diambil satu keputusan.
b. Masing-masing tahap terdiri dari sejumlah status (*state*) yang berhubungan dengan tahap tersebut. Secara umum, status merupakan bermacam kemungkinan masukan yang ada pada tahap tersebut.
c. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya.
d. Ongkos (*cost*) pada suatu tahap meningkat secara teratur (*steadily*) dengan bertambahnya jumlah tahapan.
e. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos pada tahap tersebut.
f. Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusan yang dilakukan pada tahap sebelumnya.
g. Adanya hubungan rekursif yang mengidentifikasikan keputusan terbaik untuk setiap status pada tahap *k* memberikan keputusan terbaik untuk setiap status pada tahap *k* + 1.
h. Prinsip optimalitas berlaku pada persoalan tersebut.

Dengan kata lain sesuai dengan namanya program dinamis akan tergantung dari keseluruhan perhitungan ongkos untuk menentukan solusi optimal.

2.4 Algoritma *Bellman Ford* dalam *Distance Vector Routing*

Algoritma *Bellman Ford* menggunakan prinsip program dinamis. Algoritma ini akan manghitung setiap *vector* dalam *router*. Setiap *router* akan *best distance* yang akan dipilih. Setiap *router* ini akan mengirimkan informasi tersebut ke setiap tetangganya. Informasi ini akan digunakan untuk perhitungan selanjutnya. Algoritma ini digunakan dalam *distance vector routing.*

Konsep dasar dalam *distance vector routing:*

a. Setiap saat, masing-masing simpul akan mengirimkan hasil perhitungan *distance vector* untuk setiap tetangganya.
b. *Asynchronous.*
c. Setiap kali simpul menerima *distance vector* dari tetangganga maka simpul tersebut akan memperbarui nilai *distance table* yang dia miliki.
d. *Cost* dari simpul pengirim ke penerima dihitung dengan membandingkan *cost* dari pengirim ke *v* + *cost* dari *v* ke simpul penerima, untuk setiap *v* simpul tetangga pengirim.

Untuk melakukan perhitungan tersebut perhatikan notasi-notasi berikut:

$$D_x(y) \rightarrow estimasi\ cost\ dari\ x\ ke\ y \quad (5)$$

$$c(x,v) \rightarrow cost\ dari\ x\ ke\ tetangga\ v \quad (6)$$

$$D_x \rightarrow distance\ vector\ x\ untuk\ setiap\ tetangga \quad (7)$$

$$D_x = [D_x(y): y \in N] \quad (8)$$

$$D_x(y,z) \rightarrow estimasi\ cost\ dari\ x\ ke\ y\ melewati\ z \quad (9)$$

$$D_x(y,z) = c(x,z) + \min_w\{D_z(y,w)\} \quad (10)$$

atau $D_x(y) = \min_v\{c(x,v) + D_v(y)\}$ (11)

Algoritma:

```
Initialization:
for all adjacent nodes v:
      Dx(*,v) = infty
      /* the * operator means "for all
      rows" */
      Dx(v,v) = c(x,v)
for all destinations, y
      send minwD(y,w) to each neighbor
      /* w over all x's neighbors */

loop
wait (until I see a link cost change to
neighbor v
or until I receive update from neighbor
v)
```

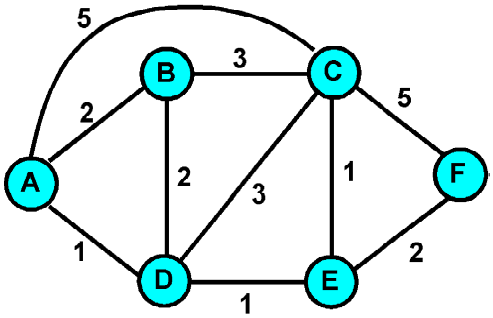
```
if (c(x,v) changes by d)
/* change cost to all dest's via
neighbor v by d */
/* note: d could be positive or
negative */
      for all destinations y: Dx(y,v) =
      Dx(y,v) + d

else if (update received from v wrt
destination y)
/* shortest path from v to some y has
changed */
/* v has sent a new value for its minw
DV(y,w) */
/* call this received new value is
"newval" */
      for the single destination y:
      Dx(y,v) = c(x,v) + newval

if we have a new minw Dx(y,w)for any
destination y
      send new value of minw Dx(y,w) to
      all neighbors
forever
```

3. ANALISIS

Untuk lebih jelasnya lihat contoh topologi jaringan di bawah ini:



Gambar 2. *Routing* dalam Sebuah Topologi Jaringan

3.1 *Link State Routing*

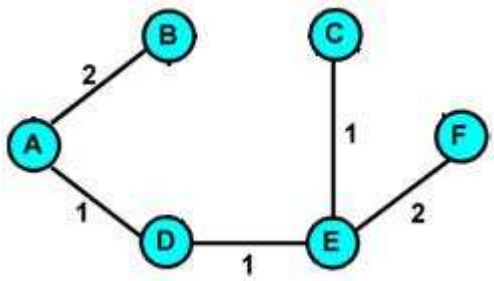
Dengan menggunakan algoritma yang telah dibahas akan didapat tahapan seperti yang tergambar dalam tabel berikut:

Tabel 1 Tabel Tahapan Pembentukan *Link State*

step	N	D(B),p(B)	D(C),P(C)	D(D),P(D)	D(E),P(E)	D(F),p(F)
0	A	2,A	5,A	1,A	infty	infty
1	AD	2,A	4,D		2,D	infty
2	ADE	2,A	3,E			4,E
3	ADEB		3E			4E
4	ADEBC					4E
5	ADEBCF					

**bagian yang dilingkari adalah minimum lokal*

Dengan perhitungan tersebut didapatkan *link state* seperti ini:



Gambar 3. Graf *Link State*

Forwarding table yang dihasilkan pada simpul A adalah:

Tabel 2 Tabel *Forwarding Table* Simpul A

tujuan	link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

Dengan ini dapat disimpulkan untuk mendapatkan rute dari simpul A ke simpul F dengan melewati simpul-simpul A → D → E → F.

3.2 *Distance Vector Routing*

Dengan menggunakan persamaan ini:

Persamaan (11), $D_x(y) = \min_v \{ c(x, v) + D_v(y) \}$, *Distance vector* dari simpul A ke simpul F:.

$$D_A(F) = \min_v \{ c(A, v) + D_v(F) \}$$
$$= \min \{c(A,B) + D_B(F), c(A,D) + D_D(F), c(A,C) + D_C(F)\}$$
$$= \min \{2 + D_B(F), 1 + D_D(F), 5 + D_C(F)\} \quad (12)$$

$$D_B(F) = \min_v \{ c(B, v) + D_v(F) \}$$
$$= \min \{c(B,A) + D_A(F), c(B,D) + D_D(F), c(B,C) + D_C(F)\}$$
$$= \min \{2 + D_A(F), 2 + D_D(F), 3 + D_C(F)\}$$
$$= \min \{2 + D_D(F), 3 + D_C(F)\} \quad (13)$$

$$D_C(F) = \min_v \{ c(C, v) + D_v(F) \}$$
$$= \min \{c(C,B) + D_B(F), c(C,D) + D_D(F), c(C,E) + D_E(F), c(C,F) + D_F(F)\}$$
$$= \min \{3 + D_B(F), 3 + D_D(F), 1 + D_E(F), 5\}$$
$$= \min \{3 + D_D(F), 1 + D_E(F), 5\} \quad (14)$$

$$D_D(F) = \min_v \{ c(D, v) + D_v(F) \}$$
$$= \min \{c(D,A) + D_A(F), c(D,B) + D_B(F), c(D,C) + D_C(F), c(D,E) + D_E(F)\}$$
$$= \min \{1 + D_A(F), 2 + D_B(F), 3 + D_C(F), 1 + D_E(F)\}$$
$$= \min \{ 1 + D_E(F)\} \quad (15)$$

$$D_E(F) = \min_v \{ c(E, v) + D_v(F) \}$$
$$= \min \{c(E,D) + D_D(F), c(E,C) + D_C(F), c(E,F) + D_F(F)\}$$
$$= \min \{1 + D_D(F), 1 + D_C(F), 2\}$$
$$= \min \{2\}$$
$$= 2 \text{ (simpul F)} \quad (16)$$

Dengan melangkah mundur didapat:
Persamaan (15)
$$D_D(F) = \min \{ 1 + D_E(F)\}$$
$$= \min \{ 1 + 2\}$$
$$= 3 \text{ (simpul E)}$$

Persamaan (14)
$$D_C(F) = \min \{3 + D_D(F), 1 + D_E(F), 5\}$$
$$= \min \{3 + 3, 1 + 2, 5\}$$
$$= \min \{6, 3, 5\}$$
$$= 3 \text{ (simpul E)}$$

Persamaan (13)
$$D_B(F) = \min \{2 + D_D(F), 3 + D_C(F)\}$$
$$= \min \{2 + 3, 3 + 3\}$$
$$= \min \{5, 6\}$$
$$= 5 \text{ (simpul D)}$$

Persamaan (12)
$$D_A(F) = \min \{2 + D_B(F), 1 + D_D(F), 5 + D_C(F)\}$$
$$= \min \{2 + 5, 1 + 3, 5 + 3\}$$
$$= \min \{7, 4, 8\}$$
$$= 4 \text{ (simpul D)}$$

Dengan ini dapat disimpulkan rute yang dilalui dari simpul A ke simpul F adalah A → D → E → F dengan total *cost* minimum = 4.

3.3 Perbandingan *Link State* dan *Distance Vector*

Meskipun pada contoh di atas didapatkan *path* yang sama dengan dua jenis algoritma *routing*, akan tetapi pada dasarnya dua jenis algoritma tersebut memiliki perbedaan mendasar, perhatikan tabel berikut:

Tabel 3 Tabel Perbandingan *Link State dan Distance Vector*

Link State *(Algoritma Dijkstra)*	*Distance Vector* *(Algoritma Bellman Ford)*
Dengan *x* simpul dan *y* link, ada *O*(*xy*) pesan dikirim.	Terjadi pengiriman pesan hanya dengan tetangga simpul saja.
Dibutuhkan $O(n^2)$ proses untuk *O*(*xy*) pesan.	Waktu proses cenderung lama. Kemungkinan *infinit loop*. Karena sifat rekursif.
Setiap simpul hanya menghitung *node table* sendiri.	Setiap informasi *node table* pada setiap simpul digunakan simpul lain.
Kemungkinan kesalahan pada *link cost.*	*Path* akan salah jika *path cost* pada suatu node salah.

Bagaimanapun setiap algoritma akan lebih cocok untuk kondisi yang berbeda. Sebuah topologi jaringan yang membutuhkan efisiensi tidak tepat jika menggunakan *distance vector algorithm*, sedangkan sebuah topologi jaringan yang cenderung memiliki banyak simpul (*host*) tidak cocok menggunakan *link state algorithm.* Kedua jenis *routing* ini dimanfaatkan sesuai kondisi untuk mendapatkan *route* yang efektif dan efisien dalam pengiriman paket-paket data.

4. KESIMPULAN

Strategi Algoritma memiliki banyak teori algoritma dasar yang sebenarnya potensial untuk dikembangkan. Setiap tipe algoritma memiliki karakteristik tersendiri sesuai dengan nilai efisiensi proses maupun waktu, mekanisme pemecahan masalah, dan metode pengembangan algoritma.

Sebagai contoh algoritma *greedy* dan program dinamis telah dimanfaatkan untuk pemecahan masalah *routing* dalam sebuah topologi jaringan komputer. Konsep dasar algoritma *greedy* digunakan dalam algoritma *dijkstra* untuk dimanfaatkan dalam *link state routing algorithm.* Sedangkan program dinamis menjadi dasar pengembangan algoritma *Bellman Ford* yang digunakan dalam *distance vector algorithm.*

Kedua tipe *routing* tersebut memiliki kekurangan dan kelebihan masing-masing. Penggunaan keduanya tergantung kebutuhan dan keadaan sebuah topologi.

Dengan mengenal konsep dasar algoritma yang digunakan, dan mempelajari karakteristik untuk setiap algoritma *routing*, maka seseorang dapat dengan mudah menganalisis algoritma mana yang sesuai dengan keadaan dan kebutuhan sebuah topologi jaringan.

Untuk itulah makalah ini dibuat, agar pemahaman tentang *routing* dan karakteristik dasar algoritma yang digunakan bertambah.

Semoga pembuatan makalah ini bermanfaat baik bagi penulis maupun bagi pembaca. Terimakasih.

DAFTAR REFERENSI

[1] James F. Kurose *and* Keith W. Ross, "*Computer Networking*", 2000, Hal 270 - 280.

[2] Andrew S. Tanenbaum, "*Computer Networks*", 2003, Pearson Education, Inc. Hal. 264 – 280.

[3] Rinaldi Munir, "*Diktat Kuliah IF3051 Strategi Algoritma*", i*nformatika ITB*, 2009, Bab Program Dinamis.