Penerapan Algoritma *Backtracking* sebagai *Routing Algorithm* pada Jaringan Komputer

Mukhtar Hudaya

Laboratorium Ilmu dan Rekayasa Komputasi
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
E-mail: if12078@students.if.itb.ac.id

Abstrak

Dalam jaringan komputer, *routing* adalah proses untuk menentukan jalur komunikasi antara sumber dan tujuan dari sebuah paket. *Routing* adalah fungsi utama dari *network layer* dalam *OSI Reference Model*. Pada kebanyakan jaringan komputer, sebuah paket data memerlukan banyak loncatan antarhost untuk mencapai tujuan. Selama terdapat jalur fisik antara kedua mesin, sebuah algoritma *routing* harus dapat memastikan bahwa data yang dikirim dari mesin sumber diterima oleh mesin tujuan.
***Backtracking* adalah algoritma yang berbasis pada algoritma DFS (*Depth-First Search*) yang dapat mencari solusi sebuah persoalan dengan lebih mangkus. Algoritma ini dapat menemukan solusi sebuah persoalan tanpa perlu memeriksa semua kemungkinan solusi dan hanya mempertimbangkan pencarian yang mengarah ke solusi.**
Makalah ini memaparkan penerapan algoritma *backtracking* sebagai algoritma *routing* pada jaringan komputer.

Kata kunci: *Routing algorithm, backtracking,* DFS, jaringan komputer.

1. Pendahuluan

Algoritma routing dapat dibagi menjadi dua kelas: *nonadaptive* dan *adaptive*. Algoritma *nonadaptive* tidak mendasarkan keputusan routing pada keadaan lalu lintas data dan topologi jaringan saat ini. Pemilihan jalur komunikasi yang digunakan antarmesin pada algoritma ini ditentukan dari awal dan ditanamkan ke router pada saat jaringan diaktifkan. Algoritma routing ini disebut juga static routing. Kebalikannya, algoritma adaptive menentukan jalur komunikasi berdasar kondisi jaringan saat ini, seperti topologi yang digunakan dan juga kondisi lalu lintas data. Algoritma adaptive (dynamic routing) memperoleh informasi untuk proses routing secara lokal, dari router terdekat atau dari semua router yang ada di jaringan. Algoritma routing yang akan dibahas pada makalah ini adalah algoritma adaptive atau dynamic routing.

Algoritma routing harus dapat mendeteksi adanya cacat pada sebuah host yang akan dilalui paket data dan menentukan jalur lain yang dapat dilalui untuk mencapai mesin tujuan. Cacat tersebut dapat berupa mesin yang mati atau ketidakmampuan mesin tersebut untuk meneruskan paket data ke mesin lain.

Selain untuk memastikan tersampaikannya paket dari sumber ke tujuan, pemilihan algoritma routing juga harus mempertimbangkan besarnya biaya (memori dan *bandwidth*) yang diperlukan.

Dipilihnya *backtracking* sebagai algoritma routing juga berdasar pertimbangan di atas: efektivitas untuk memperoleh solusi (dalam hal ini *route* dari sumber ke tujuan) dan efisiensi dalam penggunaan sumber daya komputasi.

2. Prinsip Kerja Algoritma *Backtracking*

Backtracking menggunakan prinsip DFS untuk mencari sebuah solusi persoalan. Ruang solusi diorganisasikan dalam struktur pohon dengan simpul pohon menyatakan status persoalan.

Algoritma *backtracking* mencari solusi dengan menelusuri pohon dengan prinsip DFS. Di setiap simpul yang dikunjungi, algoritma akan mengecek apakah lintasan yang tercipta mengarah ke solusi (memenuhi fungsi pembatas). Jika lintasan tidak mengarah ke solusi, simpul tersebut akan ‘dibunuh’ dan menjadi simpul mati. Simpul mati ini tidak akan diperluas lagi. Jika pembentukan lintasan berakhir dengan simpul mati, proses pencarian akan membangkitkan simpul anak yang lain. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, pencarian solusi dilanjutkan dengan melakukan runut-balik (*backtrack*) ke simpul hidup terdekat, yaitu orang tua. Simpul ini akan menjadi *expand-node* yang baru.

Pencarian akan berhenti bila solusi sudah ditemukan atau tidak ada lagi simpul hidup untuk runut-balik.

Algoritma *backtracking* secara signifikan memperbaiki kompleksitas algoritma yang dijumpai pada algoritma *brute force*. Jika jumlah simpul dalam pohon ruang status

adalah 2^n atau n!, maka pada kasus terburuk, *backtracking* memerlukan waktu $O(p(n)2^n)$ atau O(q(n)n!) dengan p(n) dan q(n) adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul [3].

3. Analisa Penerapan Algoritma

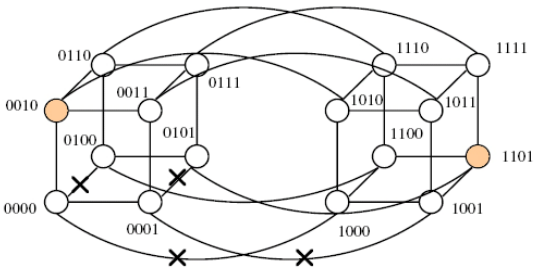
Algoritma routing yang berdasar pada teknik pencarian graf telah terbukti memberikan fleksibilitas yang paling baik[1]. Jika digunakan metode *exhaustive traversal*, paket data dikirim melalui semua *node* yang dapat dikunjungi sampai *node* tujuan ditemukan. Dengan metode ini, sebuah *node* pada jaringan dapat dikunjungi lebih dari satu kali, sehingga dapat menghasilkan beban yang tidak perlu bagi jaringan.

Pendekatan yang dilakukan untuk mencegah redundansi kunjungan paket data ke sebuah *node* adalah dengan menyimpan informasi status pada *header* pesan yang dikirim.

Misalkan sebuah paket mempunyai struktur (*R*, *TD*, *message*). *R* adalah vektor *n*-bit yang berfungsi sebagai *routing tag.* Jika bit *i* pada *R* terdefinisi, maka paket data harus melalui dimensi *i*. Jika *R* = 0, maka paket data telah mencapai tujuan. *TD* adalah himpunan dimensi yang telah dilalui oleh paket data. Pada *host* sumber, *TD* diinisialisasi dengan nilai 0. Selama perjalanan, setiap melalui *node* antara sumber dan tujuan, nilai *R* dan *TD* diperbaharui.

Proses routing pada titik-titik antara (*intermediate node*) melalui beberapa tahap. Tahap pertama adalah mentransmisikan paket data melalui jalur terpendek (*shortest path*) ke *node* tujuan. Jika jalur tertutup oleh komponen (*node* / *host*) yang cacat, maka himpunan dimensi yang pernah dilalui oleh paket data sampai saat ini dapat digunakan untuk menghitung alamat semua *node* yang pernah dilalui oleh paket data ini. Semua *node* tetangga dari *node* tersebut tidak lagi dijadikan kandidat sebagai jalur yang dilalui paket data. Jika semua *node* tetangga telah dilalui atau semua tertutup oleh komponen yang cacat, maka paket data akan melakukan *backtrack* ke *node* sebelumnya.

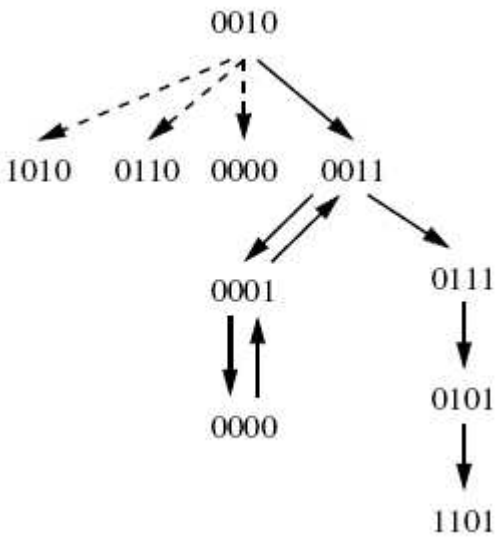
Misalkan, kita ingin mengirim paket data dari *node* 0010 ke *node* 1101 seperti terlihat pada **Gambar 1**.



Gambar 1: Graf yang menunjukkan jalur komunikasi pada sebuah jaringan komputer.

Awalnya, paket data akan melalui dimensi 0 untuk mencapai *node* 0011, kemudian melalui dimensi 1 untuk mencapai *node* 0001. Kerusakan pada jalur komunikasi yang seharusnya dipilih, yaitu jalur dari 0001 ke 0101 dan dari 0001 ke 1001 menyebabkan paket data terkirim ke *node* 0000. Di *node* 0000, kerusakan terdeteksi pada dua jalur komunikasi. Daftar dimensi yang telah dilalui oleh paket data tersimpan pada header paket. Daftar ini dapat digunakan untuk menyusun ulang semua *node* yang telah dilalui paket data. Dari informasi ini, *node* 0000 dapat menentukan bahwa *node* 0010 pernah dilalui oleh paket data. Hal ini menyebabkan *node* 0000 untuk melalukan *backtrack* dengan mengirim ulang paket data ke *node* 0001. Di *node* 0001, semua kemungkinan jalur telah diperiksa sehingga menyebabkan paket data backtrack ke *node* 0011 dengan isi pesan (1110, [0, 1, 0, 0, 1], *message*). Algoritma routing pada *node* 0011 akan memilih dimensi terkecil untuk melakukan pergerakan menuju *node* tujuan, yaitu *node* 2, sehingga paket yang dikirim ke *node* 0111 berisi (1010, [0, 1, 0, 0, 1, 2], *message*). *Node* 0111 akan memilih dimensi 1 dan mengirim pesan (1000, [0, 1, 0, 0, 1, 2 ,1], *message*) ke *node* 0101. *Node* 0101 kemudian mengirim paket data ke *node* 1101 yang merupakan *node* tujuan.

Proses *backtracking* pada pengiriman paket data di atas diilustrasikan pada **Gambar 2**.



Gambar 2: Pohon dinamis yang menggambarkan proses *backtracking* dalam pengiriman paket data dari 0010 ke 1101

Pseudocode untuk routing dengan algoritma *backtracking* dapat dilihat pada **Gambar 3**. Algoritma tersebut adalah algoritma yang akan diterapkan oleh tiap *node* saat memperoleh kiriman paket data dari *node* lain.

```
function min(input X: vektor biner) : integer
{fungsi untuk memperoleh nilai minimum dari komponen vektor biner X
}

function notVisited(anyNode: node) : boolean
{fungsi untuk untuk mengecek apakah node anyNode sudah pernah dikunjungi atau belum. Fungsi mengembalikan nilai true bila anyNode belum dikunjungi, false bila sebaliknya
}

function noFault(anyDimension: integer) : boolean
{ fungsi untuk mengecek apakah path dari node saat ini dengan dimensi anyDimension cacat atau tidak. Fungsi mengembalikan nilai true bila tidak cacat, false bila cacat.
}

function getNode() : node
{fungsi untuk memperoleh node berdasar dimensi pergerakan dan node saat ini
}

procedure stop()
{ prosedur untuk menghentikan proses / eksekusi algoritma, dilakukan misalnya karena data sudah dikirim ke node lain oleh node ini.
}

procedure BacktrackRouting (input msg: PaketData)
{ PaketData adalah data dengan struktur (R, TD, message), dimana R adalah (r_{n-1}, r_{n-2}, ..., r_1, r_0).
  Notasi ⊕ adalah operasi XOR (exclusive OR) pada vektor biner.
  e^i merepresentasikan sebuah vektor bit berukuran n dengan bit i bernilai 1 dan bit-bit yang lain bernilai 0.
  Notasi & adalah operasi penggabungan list.
}

Deklarasi
  j, n, h : integer
Algoritma:
  if R = 0 then node tujuan sudah tercapai
  for j := 0 to n – 1 and not b do begin
    if notVisited(getNode(r_j)) then
      kirim (R ⊕ e^i, TD&j, message ) melalui dimensi j.
      stop()
    endif
  endfor

  if notFault(min(R)) and notVisited(getNode(min(R))) then
  begin
    kirim (R ⊕ e^h, TD&h, message ) melalui dimensi h.
      stop()
  endif

  if node saat ini memperoleh paket dari node dengan dimensi g then begin
    kirim ulang (R ⊕ e^g, TD&g, message ) ke node sebelumnya
    {backtrack ke node sebelumnya}
  endif
```

Gambar 3: Pseudocode *routing algorithm* dengan menerapkan algoritma *backtracking*

4. Kesimpulan

Secara teoretis, penggunaan algoritma *backtracking* untuk algoritma routing pada jaringan komputer dapat selalu menyampaikan paket yang dikirim ke mesin tujuan, meskipun terdapat komponen yang cacat (*fault*) pada jaringan komputer, selama masih terdapat jalur fisik antara mesin sumber dan mesin tujuan.

Penerapan algoritma *backtracking* seperti diilustrasikan pada makalah ini juga cukup hemat dari sisi penggunaan sumber daya komputasi. Algoritma ini juga mencegah dilaluinya sebuah *node* lebih dari satu kali tanpa perlu, sehingga beban jaringan dapat dikurangi.

5. Referensi

[1] Jose Duato, "Interconnection Networks: An Engineering Approach", Morgan Kaufmann Publisher, 2003.

[2] Andrew S. Tanenbaum, "Computer Network", Prentice Hall, 2003.

[3] Rinaldi Munir, "Diktat Kuliah IF2251: Strategi Algoritmik", Program Studi Teknik Informatika, STEI ITB, 2006.