ALGORITMA OPTIMASI DAN APLIKASINYA

Andi Hasad
andihasad@yahoo.com
Sekolah Pascasarjana IPB, Departemen Ilmu Komputer
Jl. Raya Darmaga, Kampus IPB Darmaga, Wing 20 Level 5-6,
Bogor - Jawa Barat, Indonesia,16680
Telp. 0251-8625584, Fax. 0251-8625584

I. Pendahuluan

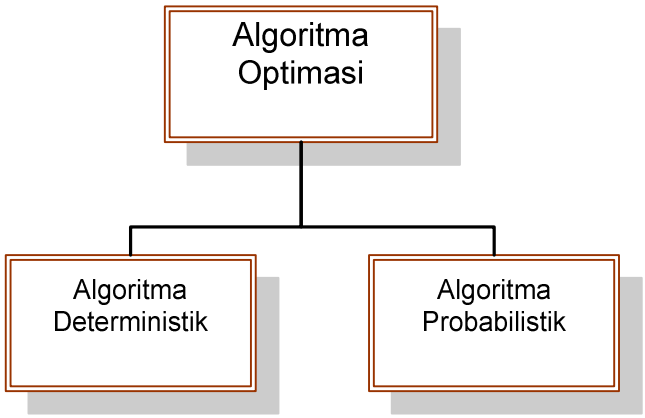
Dalam kehidupan sehari-hari, kita akan mudah menemukan permasalahan dari algoritma optimasi. Salah satu contohnya adalah bagaimana menyusun jadwal perkuliahan sehingga tidak bentrok ruangan, dosen, maupun mahasiswa dengan distribusi yang adil dan semaksimal mungkin memenuhi kebutuhan semua pihak. Untuk menyelesaikan berbagai permasalahan seperti itu, para ahli telah mengelompokkan berbagai algoritma ke dalam kelompok Algoritma Optimasi (AO). Setiap algoritma memiliki kelebihan dan kekurangan masing-masing dalam menyelesaikan suatu masalah, karena tidak ada satupun algoritma yang berlaku umum dan bisa digunakan untuk menyelesaikan semua jenis masalah. Olehnya itu, diperlukan kemampuan memilih AO yang paling tepat (sesuai) untuk menyelesaikan masalah yang dihadapi.

Para ahli mengelompokkan AO ke dalam dua kelompok yaitu :

1. AO Deterministik
2. AO Probabilistik

Perbedaan yang mendasar dari kedua kelompok tersebut adalah pada setiap langkah eksekusi di dalam AO Deterministik, hanya terdapat satu jalan untuk diproses, jika tidak ada jalan maka algoritma dianggap selesai. Dengan demikian AO ini selalu menghasilkan solusi yang tetap untuk suatu input yang diberikan. Algoritma ini biasanya digunakan untuk masalah yang ruang solusinya tidak terlalu besar, sedangkan AO Probabalistik digunakan untuk menyelesaikan ruang masalah dengan ruang solusi yang sangat besar, bahkan tak terbatas. Algoritma Probabilistik berusaha menemukan solusi yang “bagus” tanpa melebihi batasan waktu yang telah ditentukan. Solusi yang “bagus” belum tentu yang

paling **optimal,** namun sudah dapat diterima oleh *user*. Contohnya menyelesaikan masalah TSP (Travelling Salesman Problem) untuk jutaan lokasi, maka akan memerlukan 1000 tahun komputasi jika diselesaikan dengan AO Deterministik . Jika kita menginginkan solusi dalam satu hari maka solusi menggunakan AO Deterministik tidak mungkin dilakukan. Namun jika ada AO Probabilistik yang bisa memberikan solusi yang "bagus" (sedikit lebih besar daripada solusi paling minimum, tetapi bisa diterima oleh kita) dalam waktu satu hari, maka kita bisa menggunakan algoritma tersebut.



Gambar 1. Pengelompokan Algoritma Optimasi

II. Tinjauan Pustaka

2.1 Algoritma

Algoritma adalah setiap prosedur komputasi yang terdefinisi dengan baik yang mengambil beberapa nilai, atau seperangkat nilai-nilai, sebagai masukan dan menghasilkan beberapa nilai, atau seperangkat nilai-nilai, sebagai output. Sebuah algoritma merupakan langkah komputasi yang mengubah input ke output. Kita juga dapat melihat sebuah algoritma sebagai alat bantu untuk memecahkan masalah (Cormen, 2001). Algoritma adalah urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis. Langkah-langkah logis berarti kebenarannya harus dapat ditentukan, benar atau salah (Munir, 1999), sedangkan menurut kamus besar bahasa Indonesia terbitan balai pustaka, Departemen Pendidikan Nasional, edisi ketiga tahun 2007, algoritma adalah prosedur sistematis untuk memecahkan masalah matematis dalam langkah-

langkah terbatas. Masalah merupakan sesuatu yang harus diselesaikan (dipecahkan). Pernyataan masalah menentukan secara umum yang diinginkan pada hubungan input / output dan algoritma menggambarkan prosedur komputasi tertentu untuk mencapai yang hubungan input / output.

Di dalam Ilmu Komputer, pertanyaan yang sering ditanyakan bukanlah bukanlah bagaimana cara menyelesaikan suatu persoalan, melainkan bagaimana menyelesaikan persoalan dengan baik (efisien). Sebagai contoh adalah menyelesaikan persoalan pengurutan/*sorting*. Ada berbagai macam algoritma yang digunakan untuk proses pengurutan, namun yang menjadi persoalan bukanlah bagaimana menemukan cara untuk mengurutkan, tetapi mencari cara yang efisien dalam mengurutkan suatu deret (*sequence*). Berikut adalah bagaimana kita mendefinisikan secara resmi masalah pengurutan :

Input : sebuah urutan angka n ($a_1, a_2, \ldots, a_n$)

Output : sebuah permutasi ($a'_1, a'_2, \ldots, a'_n$) pada urutan input :$a'_1 \leq a'_2 \leq \cdots \leq a'_n$. Sebagai contoh, diberikan urutan input 3, 5, 9, 7, 4. Sebuah algoritma pengurutan kembali menghasilkan output urutan 3, 4, 5, 7, 9.

Pengurutan merupakan operasi fundamental dalam ilmu komputer (program banyak menggunakannya sebagai langkah menengah) dan sebagai outputnya banyak dihasilkan algoritma yang baik.

Pada dasarnya sebuah algoritma dapat dibangun dari tiga buah struktur dasar, yaitu :

1. Runtunan (*Sequence)*

   Sebuah *Sequence* terdiri dari satu atau lebih instruksi. Tiap instruksi dikerjakan secara berurutan sesuai dengan urutan penulisannya, yakni sebuah instruksi dilaksanakan setelah instruksi sebelumnya selesai dilaksanakan.
2. Pemilihan (*selection)*

Struktur *selection* terletak pada kemampuannya yang memungkinkan pemroses mengikuti jalur aksi yang berbeda berdasarkan kondisi yang ada. Tanpa struktur *selection*, kita tidak mungkin menulis algoritma untuk permasalahan praktis yang demikian kompleks.

3. Pengulangan (*repetition)*

   Struktur *repetition* memungkinkan komputer melakukan pengulangan pada sebuah pekerjaan tanpa mengenal lelah. Struktur ini biasa juga disebut *loop*, dan bagian algoritma yang diulang disebut *loop body*

2.2 Optimal

Optimal adalah terbaik, paling menguntungkan (Kamus Besar Bahasa Indonesia Balai Pustaka, 2007). Algoritma yang optimal dapat diartikan urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis dalam langkah-langkah terbatas, yang paling baik dan menguntungkan, yang menggambarkan prosedur komputasi tertentu untuk mencapai yang hubungan input / output.

2.3 Algoritma Optimasi

Algoritma Optimasi (*Optimization Algorithms)* dapat didefenisikan sebagai algoritma untuk menemukan nilai x sedemikian sehingga menghasilkan f(x) yang bernilai **sekecil** atau **sebesar** mungkin untuk suatu fungsi f yang diberikan, yang mungkin disertai dengan beberapa batasan pada x. Dimana x bisa berupa skalar atau vektor dari nilai-nilai kontinu maupun diskrit.

Global Optimization didefenisikan sebagai suatu cabang ilmu dari matematika terapan dan analisa numerik yang membahas optimasi dengan kriteria yang bersifat tunggal, ganda atau bahkan mungkin konflik. Kriteria diekspresikan sebagai himpunan fungsi matematika F = {f1, f2,…, fn} yang disebut fungsi-fungsi obbjektif. (Thomas Weise, 2007)

Algoitma optimasi sedikit berbeda dengan algoritma pencarian (*search algorithmn*). Pada algoritma pencarian terdapat suatu kriteria tertentu yang menyatakan apakah elemen x_i merupakan solusi atau bukan. Sebaliknya pada

algoritma optimasi mungkin tidak terdapat kriteria tersebut, melainkan hanya fungsi-fungsi objektif yang menggambarkan bagus tidaknya suatu konfigurasi yang diberikan. Algoritma optimasi bisa dikatakan sebagai generalisasi dari algoritma pencarian atau dengan kata lain, algoritma pencarian adalah kasus khusus dari algoritma optimasi.

2.4 Klasifikasi Algoritma Optimasi

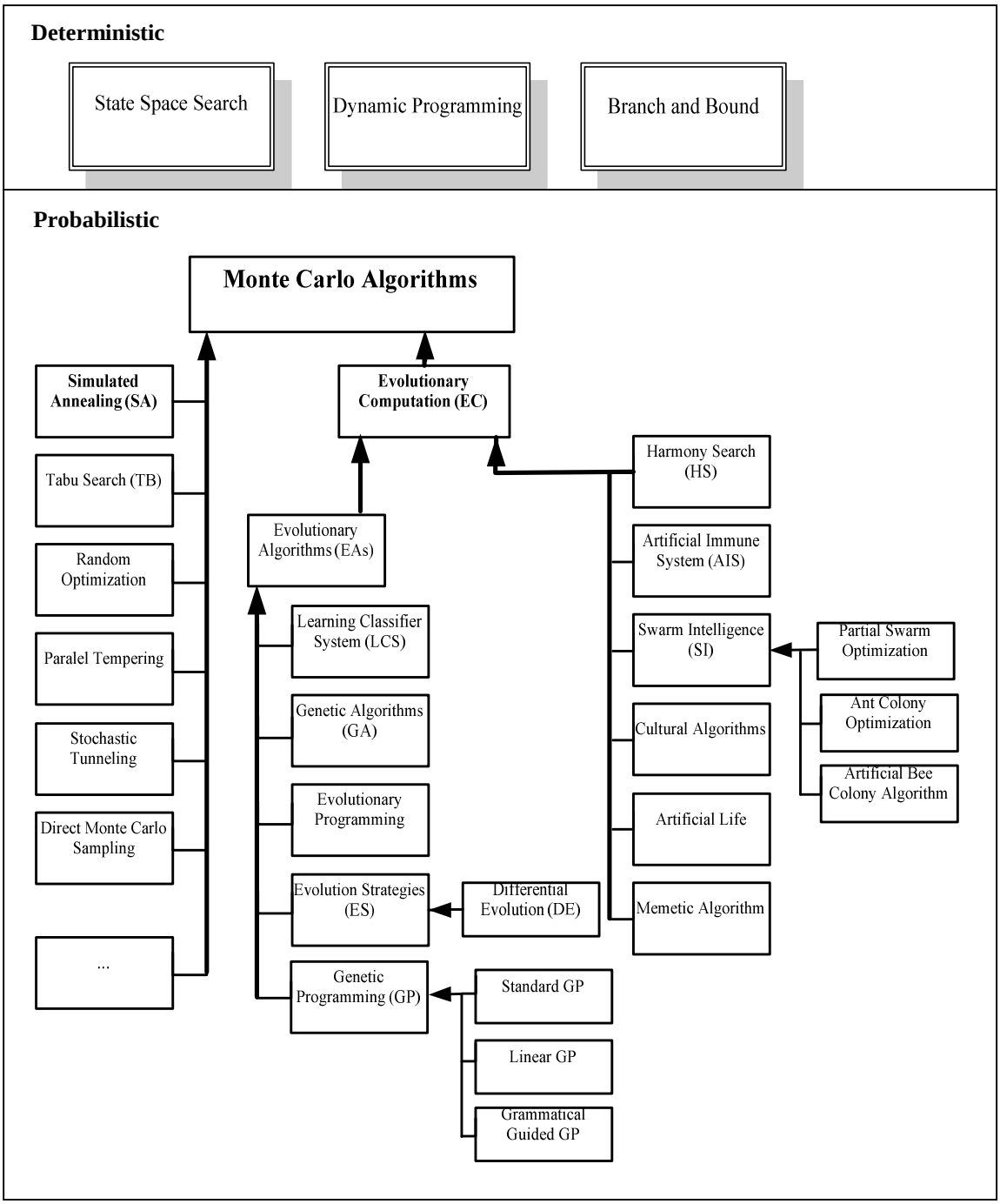
2.4.1 Berdasarkan Metode Operasinya

Berdasarkan metode operasinya, algoritma optimasi dapat dibagi ke dalam dua kelas besar, yaitu algoritma deterministik *(deterministic)* dan probabilistik *(probabilistic).* Pada setiap langkah eksekusi dalam algoritma deterministik, terdapat maksimum satu jalan untuk diproses. Jika tidak ada jalan, berarti algoritma sudah selesai. Algoritma deterministik sering digunakan untuk masalah yang memiliki relasi yang jelas antara karakteristik calon solusi dengan utilitasnya. Dengan demikian, ruang pencarian dapat dieksplorasi menggunakan, misalnya, metode *branch and bound* atau *divide and conquer scheme.* Tetapi, jika relasi antara suatu kan-didat solusi dan *"fitness"*-nya tidak dapat dipahami atau diamati (kandidat-kandidat solusi yang bertetangga mungkin sangat berbeda dalam hal utilitas atau dimensi ruang pencarian yang sangat besar), maka masalah ini akan lebih sulit diselesaikan secara deterministik. Bayangkan jika kita harus mencari suatu solusi dari $10^{1000000}$ kandidat solusi secara deterministik. Tentu membutuhkan usaha dan waktu yang banyak. Hal ini sama sulitnya dengan mencari jarum di tumpukan jerami.

Untuk permasalahan dengan ruang pencarian yang sangat besar, biasanya para praktisi lebih sering menggunakan algoritma probabilistik. Hampir semua algoritma probabilistik menggunakan konsep dasar dari metode Monte Carlo (MC). Metode MC bersandar pada proses pengambilan sampel secara acak yang berulang-ulang *(repeated random sampling)* unruk menghasilkan solusi.

Demikianlah perbedaan signifikan antara algoritma deterministik dan algoritma probabilistik. Untuk permasalahan tertentu, kita bisa menggunakan algoritma deterministik. Tetapi, untuk masalah yang lain mungkin saja kita harus menggunakan algoritma probabilistik. Bagaimanapun tidak ada satupun algoritma

optimasi yang sesuai untuk semua masalah. Biasanya suatu algoritma sangat efektif dan efisien untuk beberapa masalah, tetapi algoritma tersebut mungkin saja memiliki performansi yang kurang atau bahkan sangat buruk untuk permasalahan yang lain.



Gambar 2. Klasifikasi Algoritma berdasarkan metode operasinya.

2.4.2 Berdasarkan akurasi dan kecepatan

Pengklasifikasian algoritma optimasi di atas secara sekilas, sudah cukup memadai jika dipandang secara teori. Perbedaan prinsip dasar dan diikuti algoritma sangat jelas terlihat. Tetapi, para praktisi yang sedang menghadapi Suatu masalah cenderung memilih algoritma optimasi berdasarkan dua hal : akurasi dan kecepatan.

Cara klasifikasi lain yang bisa cligunakan adalah membedakan algoritma optimasi menjadi dua, yaitu: optimasi **online** dan optimasi **offline**.

• Optimasi online

Sesuai dengan namanya, optimasi ini ditujukan untuk permasalahan yang membutuhkan solusi dalam waktu cepat dan biasanya permasalahan tersebut terjadi secara berulang-ulang. Misalnya, penentuan lokasi robot *(robot localization),* penyeimbangan beban *(load balancing),* komposisi layanan untuk proses bisnis dari *IT system* yang sedang berjalan di suatu perusahaan, atau perbaruan jadwal pekerjaan mesin *(machine job schedule)* di suatu pabrik setelah datangnya suatu permintaan baru. Pada permasalahan tersebut, biasanya kita hanya memiliki waktu sangat singkat, mulai dari 10 mili detik hingga beberapa menit saja. Permasalahan tersebut juga sering terjadi berulang-ulang, misalnya permintaan barang ke suatu pabrik bisa datang kapan saja dan berkali-kali (misal 100 permintaan dalam sehari). Untuk mendapatkan solusi yang baik, biasanya para praktisi melakukan kompromi antara optimalitas dan kecepatan. Hampir semua algoritma optimasi probabilistik bisa digunakan untuk masalah ini karena waktu prosesnya bisa dikontrol secara penuh oleh user. Pada algoritma optimasi ini, akurasi tidak harus yang paling baik (optimum global).

• Optimasi offline

Optimasi jenis ini ditujukan untuk permasalahan yang membutuhkan solusi tidak dalam waktu cepat dan biasanya masalah ini terjadi dalam periode waktu yang lama. Misalnya, optimasi dalam proses perancangan *(design optimization), data mining,* pembuatan jadwal kuliah setiap semester, dan sebagainya. User bisa memiliki waktu lama, hingga berhari-hari, untuk menunggu

proses optimasi menghasilkan solusi yang seoptimal mungkin (kalau bisa optimum global). Untuk masalah seperti ini dan ruang masalahnya yang tidak terlalu besar, algoritma optimasi yang bersifat deterministik biasanya menghasilkan solusi yang lebih baik dibandingkan optimasi probabilistik.

2.4.2 Berdasarkan Analog dan Nama

Bardasarkan analogidan nama yang digunakan, algoritma optimasi menjadi dua : algoritma maksimasi dan algoritma minimasi.

- Algoritma Maksimasi

Algoritma ini menggunakan analogi memaksimalkan sesuatu pada dunia nyata. Salah satu contohnya adalah algoritma *Hill Climbing* (HC), menggunakan analogi pendekatan bukit untuk mencapai puncak tertinggi

- Algoritma Minimasi

Algoritma ini menggunakan analogi meminimalkan sesuatu pada dunia nyata. Salah satu contohnya adalah algoritma *Simulated Annealing* (SA) yang mensimulasikan proses annealing pada pembuatan materi yang terdiri dari butir kristal atau logam. Tujuan dari proses ini adalah menghasilkan struktur kristal yang baik dengan menggunakan energi seminimal mungkin. Salah satu studi kasus yang menggunakan *Simulated Annealing* yaitu *job shop scheduling,* yang akan dijelaskan lebih lanjut pada bagian pembahasan dari makalah ini.

III. Aplikasi Algoritma Optimasi

3.1 Kriteria Kebaikan suatu algoritma

Kriteria kebaikan suatu algoritma ditentukan oleh 5 hal , yaitu :

1. *Correctness (kebenaran)*
2. *Amount of work done*
3. *Amount of space used*
4. *Simplicity/ clarity (Kesederhanaan)*
5. *Optimality (optimalitas)*

Suatu algoritma dikatakan optimal pada *worstcase*-nya jika tidak ada algoritma lain yang sekelas yang melakukan lebih sedikit operasi dasar atau Algoritma optimal adalah yang terbaik, dimana dalam *worstcase*-nya melakukan perbandingan 2 entri dalam list yang paling sedikit. Cara memeriksa optimalitas algoritma adalah dengan menentukan batas bawah dari jumlah operasi yang diperlukan untuk menyelesaikan masalah, jika sembarang algoritme yang melakukan operasi sejumlah batas bawah tersebut berarti optimal. Untuk memeriksa optimalitas algoritme dapat dilakukan dengan cara sebagai berikut :

- Membuat algoritme yang efisien (A). Analisis A dan tentukan sebuah fungsi W, sedemikian sehingga A mengerjakan paling banyak W(n) langkah pada *worst-case*nya. Dhi, n adalah ukuran input.
- Untuk beberapa fungsi F, buktikan teori yang menyatakan bahwa sembarang algoritme dalam kelas yang dipelajari, terdapat ukuran input n yang menyebabkan algoritme tersebut melakukan paling sedikit F(n) langkah .
  - ✓ Jika W = F, maka algoritme A optimal pada *worst-case*nya.
  - ✓ Jika W ≠ F, mungkin terdapat algoritme yang lebih baik atau ada batas bawah yang lebih baik.

Prinsip dari algoritma yang optimal berikut ini adalah mengganti halaman yang tidak akan terpakai lagi dalam waktu lama, sehingga efisiensi pergantian halaman meningkat (*page fault* yang terjadi berkurang) dan terbebas dari anomali Belady. Algoritma ini memiliki *page fault rate* paling rendah di antara semua algoritma di semua kasus. Akan tetapi, optimal belum berarti sempurna karena algoritma ini ternyata sangat sulit untuk diterapkan. Sistem tidak dapat mengetahui halaman-halaman mana saja yang akan digunakan berikutnya.

Gambar 3. Contoh Algoritma yang Optimal

3.2 Optimalitas Algoritma

Salah satu cara untuk memeriksa optimalitas suatu algoritma adalah dengan membuat algoritma yang efisien. Efisiensi suatu algoritma ditentukan oleh 2 hal, yaitu :

1. Kecepatan (Waktu)

   Dalam segi kecepatan, faktor-faktor yang mempengaruhinya adalah banyak langkah, tipe data dan operator-operator.
2. Space memori (Alokasi Memori)

   Space (alokasi) memori dipengaruhi oleh struktur data dinais, procedure / function call dan recursif.

❖ **Banyak langkah**

Berikut digambarkan pengaruh banyak langkah dalam menentukan efisiensi sebuah algoritma yaitu membandingkan kecepatan dari 2 tipe data ; data integer dan data real. Kedua data tersebut memiliki dua nilai yang sama dengan operator yang sama tapi dengan variabel yang berbeda, maka terdapat perbedaan kecepatan/proses penyelesaiannya.

Contoh :

→ 500 + 65 = 565 (**lebih cepat dari**)

→ 500.0 + 65.0 $= 0.5*10^3 + 0.65*10^2$

$= 0.5*10^3 + 0.065*10^3$

$= (0.5+ 0.065)*10^3$

$= 0.565*10^3$

$= 565.0$

Banyak langkah dalam suatu algoritma dinyatakan dengan banyaknya operasi aritmatika dan logika yang dilakukan. Dengan demikian hal ini bergantung pada statement dan jenis algoritma :

- sequensial
- branching

- looping
- subroutine call (bisa memanggil prosedur dan bisa memanggil fungsi)

Kompleksitas waktu diukur dari banyak langkah (waktu tempuh) dan memori, dapat dituliskan :

Kompleksitas waktu = banyak langkah + memori

❖ **Operator**

Urutan penggunaan operator/penempatan operator bisa mempengaruhi efisiensi. Contoh perkalian (*) lebih lama daripada penjumlahan (+) , namun dalam perkembangan teknologi, perbedaan penggunaan operator maupun tipe data dasar tidak terlalu signifikan. Yang perlu diperhatikan adalah banyaknya operasi aritmatika dan logika yang dilakukan.

- Operator aritmatika : +,-,*,/,^,div,mod
- Operator logika : AND,OR,NOT masing-masing 1.
- Operator adalah jika hasil perhitungannya termasuk dalam himpunan itu sendiri.

  2 < 5 bukan operator tapi konstanta logika karena tidak menghasilkan nilai yang sejenis
- Operator : H x H → H

x = 2<5
x = True } Tidak ada operation (0 operation)
y = 5

y = 5+0 → 1 operation

y = 2+3*5 → 2 operation

y = 3*5+2 → 2 operation

❖ Sequensial

Statement s1 dgn banyak langkah n(s1)

Statement s2 dgn banyak langkah n(s2)

banyak langkah = n(s1)+n(s2)

Assigment dgn konstanta mempunyai waktu tempuh 0

x = 0
y = 1
n = x+y
} 1 operation

Built in subroutine call mempunyai waktu tempuh 1

Sin(x) → 1 op
Sin(x*pi/1000) → 3 op

Branching (percabangan)

If (kondisi) Then statement s1
Else statement s2

Contoh :

Jika n(kondisi) = waktu tempuh kondisi → 2 op
n(s1) = waktu tempuh statement s1 → 5 op
n(s2) = waktu tempuh satement s2 → 3 op

Maka :

waktu tempuh = n(kondisi) + max(n(s1),n(s2))

= 2 + 5

= 7

Loop

For Loop
For var ← awal to akhir step diff.
Statement S(var)

Jika statement dalam inner loop tidak bergantung pada var, maka statement tersebut diulang sebanyak

$$t = \begin{cases} \left\lceil \frac{akhir - awal}{step} \right\rceil kali & \rightarrow \text{jika non integer} \\ \left\lfloor \frac{akhir - awal}{step} \right\rfloor + 1 & \rightarrow \text{jika integer} \end{cases}$$

Misalnya waktu tempuh untuk statement tersebut adalah Ts, maka waktu tempuh dengan loop tsb adalah t*Ts.

Waktu tempuh untuk control loop adalah t*1.

Jadi waktu tempuh untuk loop tersebut adalah t * Ts + t = t (Ts+1)

Contoh :

1. for i ← 2 to 30 step 5

   x ← x+1
   y ← x+y
   } Ts = 2

   Berapa waktu tempuhnya ?

   $$t = \left\lceil \frac{30 - 2}{5} \right\rceil = 6$$

   T = t (Ts+1)

   = 6 (2+1)

   = 18

2. n=20

   for i ← 2 to 2*n step 5

   x ← x+1
   y ← x+y
   } Ts = 2

   Hitunglah berapa waktu tempuhnya ?

   Looping →

   $$t = \left\lceil \frac{40 - 2}{5} \right\rceil = 8$$

   T_{for} = t (Ts+1)

= 8 (2+1)

= 24

Waktu tempuh perkalian 2*n → $T_{2*n} = 1$

Jadi waktu tempuhnya = T = 24 + 1

= 25

3. for i←1 to 10

x ← x+1 → 1 op

if x>=1 then

x ← x-2 } 2 op

y ← x^2 }

else

y ← x+y → 1 op

} max(2,1)

Berapa waktu tempuhnya ?

$$t = \left\lfloor \frac{10-1}{1} \right\rfloor + 1 = 10$$

Waktu tempuh dlm percabangan = max(2,1)

Ts = 1 + max(2,1) = 3

T = t (Ts+1) = 10 (3+1) = 40

Jika statement tergantung nilai var

For var ← awal to akhir step diff

S(var)

$$T = \left\lceil \frac{akhir - awal}{step} \right\rceil + Ts(awal) + Ts(awal + step) + ...$$
$$+ Ts(awal + \left\lfloor \frac{akhir - awal}{step} \right\rfloor step)$$

Contoh :

$$t = \left\lfloor \frac{10-1}{1} \right\rfloor + 1 = 10$$

1. for i←1 to 10 →
   x ← x+1 → **1**
   for j←1 to i
      y ← x+y
      x ← x+1
   endfor
   } *
endfor

Berapa waktu tempuhnya ?

*

$$t_i = \left\lfloor \frac{i-1}{1} \right\rfloor + 1 = i$$

$T_{for}(i) = t_i\,(T_s+1) = i\,(2+1) = 3i$

Hidden *1*

$T(i) = 1+3i+\mathbf{1} = 2+3i$

$$T = \sum_{i=1}^{10} T(i)$$

$$= \sum_{i=1}^{10} (2+3i)$$

$$= \sum_{i=1}^{10} 2 \; + \; 3\sum_{i=1}^{10} i$$

$= 20 + 3 * ½ * 10 * (10+1)$

$= 185$

2. for i←1 to 10 → t = 10
   x ← x+1 → **1**
   for j←i to 10
      y ← x+y
      x ← x+1
   } *

endfor

endfor

Berapa waktu tempuhnya ?

*

$$t_i = \left\lfloor \frac{10-i}{1} \right\rfloor + 1 = 10 - i + 1$$

$T_{for}(i) = t_i (T_s+1) = (10\text{-}i+1) * (2+1)$

$= (10\text{-}i+1)*3 = (11\text{-}i)*3$

Hidden *1*

T(i) =**1**+(11-i)*3+*1*

$= 35\text{-}3i$

$$T = \sum_{i=1}^{10} T(i)$$

$$= \sum_{i=1}^{10} (35 - 3i)$$

$$= \sum_{i=1}^{10} 35 - 3\sum_{i=1}^{10} i$$

$= 350 + 3 * ½ * 10 * (10+1)$

$= 185$

3. for i←1 to 10

x ← x+1

for j←1 to i

y ← x+y

endfor

endfor

Berapa waktu tempuhnya ? (T=130)

4. for i←1 to n

x ← x+1

for j←i to n

y ← x+y

endfor

endfor

Berapa waktu tempuhnya ? ($T(n)=n^2+3n$)

5. for i←1 to n^2
   x ← x+1
   for j←i+1 to 2n
      y ← x+y
      x ← x+1
   endfor
endfor

Berapa waktu tempuhnya ?

($T(n)= - 2n^4+8n^3+n^2+1$)

6. for i←1 to 2n
   x ← x+1
   for j←1 to i+1
      y ← x+y
      x ← x+1
   endfor
endfor

Berapa waktu tempuhnya ? ($T(n)=6n^2+15n+1$)

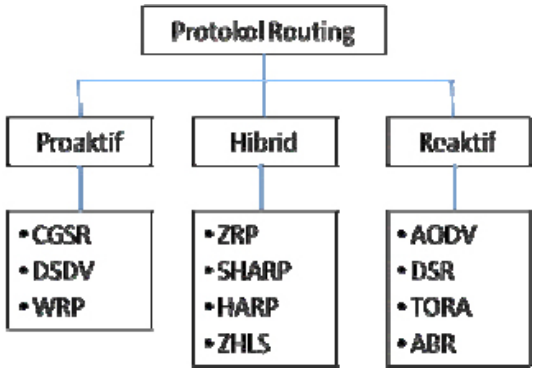
3.3 Aplikasi Algoritma Optimal Routing Hofnet

Pada bagian ini dijelaskan bagaimana menentukan Radius Optimal menggunakan Algoritma Routing Hopnet, yang menjadi topik menarik dan dibahas akhir-akhir ini. Salah satu contoh algortima hibrid pada MANET (*Mobile Ad-Hoc Network*) adalah HOPNET (*A hybrid ant colony optimization routing algorithm for MANET*). Routing dalam MANET merupakan suatu tantangan yang menarik karena MANET memiliki fitur yang dinamis, dia dibatasi oleh bandwidth dan power/energi. Jika suatu *node* sering berpindah, maka topologi jaringan akan sering berubah, sehingga jalur routing yang baik mungkin akan tidak tersedia untuk beberapa saat. Hal ini menyebabkan tiap *node* terus sering melakukan *update* terhadap tabel routingnya, akibatnya terlalu banyak kontrol paket yang membuat jaringan kebanjiran dan konsumsi sumberdaya jaringan yang terlalu

boros. Oleh karena itu, untuk proses *discovery* dan pemeliharaan routing pada lingkungan MANET merupakan hal yang sulit. Algoritma HOPNET merupakan kombinasi algoritma Hibrid yang berusaha mengatasi kelemahan – kelemahan diatas dengan mengombinasikan protokol routing jenis reaktif dan proaktif. Salah satu protokol yang digunakan didalam algoritma HOPNET ini adalah zona routing protokol (ZRP). Algoritma ZRP membagi *node* dalam zona – zona. ZRP menggunakan proses proaktif untuk update informasi routing ke *node* lokal dan menggunakan proses reaktif untuk update informasi routing antar zona (Wang, 2008). Parameter yang sangat menentukan optimal tidaknya pembagian zona dalam HOPNET adalah radius. Dalam algoritma yang sudah dikembangkan sebelumnya penentuan radius zona dilakukan secara manual. Hal ini menyebabkan kurang fleksibel untuk jaringan MANET yang cenderung dinamis / berubah topologinya. Untuk itu diusulkan penggunaan teknik *min-searching* untuk mengoptimasi penentuan radius zona pada protokol HOPNET. Teknik *min-searching* telah digunakan oleh Donggeon Noh dalam megembangkan protokol SPIZ, yang menekankan pada keefektifan layanan proses *discovery* pada MANET.

3.3.1 Algoritma routing

Algoritma routing untuk MANET dapat diklasifikasikan kedalam tiga kategori yaitu : proaktif, reaktif dan hibrid .



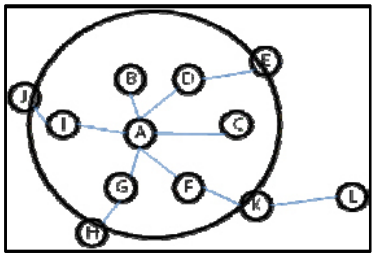
Gambar 4. Protokol routing pada MANET

Gambar 4 menunjukkan jenis protokol routing pada MANET. Pada protokol routing proaktif, tiap node secara periodik melakukan broadcast tabel routingnya

pada tetangga dan mengijinkan semua node untuk melihat konsistensi jaringan. Keuntungan dari protokol ini adalah waktu respon yang pendek untuk menentukan jalur terbaik dari sumber ke tujuan yang disebabkan oleh adanya topologi jaringan yang up to date tiap node. Waktu respon yang pendek ini harus dibayar mahal oleh konsumsi bandwidth yang besar pada kontrol paket yang tidak produktif untuk maintain jaringan tiap node. Lebih jauh, banyak jalur routing yang ada tidak pernah digunakan, yang menjadi sampah dan mengotori sumberdaya jaringan. Beberapa protokol yang menerapkannya antara lain DSDV, Fisheye dan WRP .

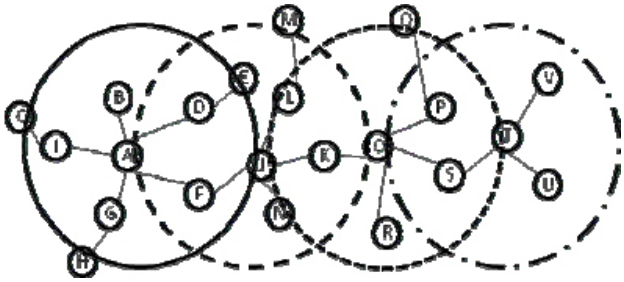
3.3.2 Algoritma HOPNET

Algoritma HOPNET: *A hybrid ant colony optimization routing algorithm for MANET* merupakan algoritma hibrid yang memanfaatkan algoritma ACO (Ant Colony Optimization) dan zone routing protocol (ZRP). Algoritma ACO didasarkan pada sifat sekumpulan *ant* tiruan dalam mencari jalur terpendek dari sumber ke tujuan. *Ant* tiruan ini bekerja sebagaimana *ant* sesungguhnya di alam yang mencari makanan dari sarang ke tujuan. *Ant* menyimpan substansi kimia yang disebut *Pheromone* yang mana *ant* yang lain dapat mengerti perjalanan mereka ke tujuan. *Ant* berinterkasi dengan lainnya dan lingkungannya menggunakan kosentrasi *Pheromone*. Seperti beberapa parfum, jika tidak dipakai ulang, maka baunya akan menguap. Dalam kerjanya ACO menggunakan dua *ant* yaitu *forward ant* dan *backward ant*. Algoritma ini juga sudah banyak dikembangkan untuk aplikasi peralatan bergerak . Algoritma HOPNET terdiri dari pencarian rute lokal proaktif dalam satu *node* tetangga dan komunikasi reaktif untuk *ant*ar tetangga. Jaringan dibagi kedalam zona-zona sebagi tetangga lokal. Ukuran dari zona ditentukan oleh radius atau hop. Oleh karena itu, sebuah routing zona terdiri dari *node* dan semua *node* dengan panjang radius yang telah ditentukan. Sebuah *node* dapat terdapat dalam zona pada berbagai ukuran. *Node* dapat digolongkan menjadi *node* interior dan *node* boundary (*peripheral*). *Node* boundary adalah *node* terjauh dari *node* pusat sedangkan *node* interior adalah sisanya.



Gambar 5. Contoh Jaringan HOPNET dengan radius 2

Pada gambar 5 dimisalkan radius dari zona *node* A adalah 2, maka *node* E,H,J,K adalah *node* boundary dan *node* B,C,D,F,G, dan I adalah *node* interior. Dalam menentukan suatu zona dan penentuan *node border*, *node* terlebih dahulu harus mengetahui *node* terdekatnya. Hal tersebut dapat diketahui melalui balasan dari pesan "*hello*" yang dikirim tiap *node*.



Gambar 6. Proses routing antar zona

Berdasar gambar 6, yang digunakan sebagai contoh *route discovery* antar zona dengan asumsi *node* asal adalah A , dengan *node* tujuan U. *Node* U tidak berada dalam satu zona dengan *node* A. *Node* A akan mengirim external *forward ant* menuju *node peripheral* (E,C,H,J) menggunakan rute yang terdapat pada tabel *IntraRT*. Ketika *ant* sampai pada E,C, dan H, *ant* akan dihancurkan karena *peripheral node* tidak mempunyai tetangga untuk melanjutkan paket keluar. Pada *node* J, dilakukan pengecekan pada tabel *IntraRT* apakah U berada dalam satu zona. Pada contoh ini U tidak terdapat dalam tabel. Oleh karena itu , J akan mengirim *ant* ke *node* periferal (M,O). J tidak akan mengirim *ant* pada *node peripheral* yang lain (A) karena *ant* datang dari F, dimana *ant* yang dikirim dari *node* tersebut akan dihancurkan, padahal untuk mencapai *node* periferal A harus melewati F. Ini adalah mekanisme pengaturan routing (duplikasi dan beban routing yang penuh). Mekanisme ini akan membantu *ant* berjalan langsung dari *node* asal. Mekanisme ini mencegah banjir *ants* pada jaringan. *Node* J

menambahkan alamat tersebut pada *field path* dan mengirim *ant* pada *node border* / periferal M (<J,L,M>) dan O (<J,K,O>). Dengan cara yang sama , ant pada M akan dihancurkan sedangkan *node* O tidak dapat menemukan U pada zona tersebut. Oleh karena itu *Node* O mengirim *ant* ke *node peripheral* (T,Q). *Node* T mengetahui bahwa U berada pada zona dalam mereka. T selanjutnya mengirim *forward ant* dengan menambahkan alamat menuju U melalui jalur yang diindikasikan menuju *node* U yang terdapat dalam tabel *IntraRT*. *Backward ant* melewati jalur kebalikan (U,T,O,J,A) menuju *node* asal A dari *node* tujuan U dengan tipe *field* pada struktur data diubah dari 1 menjadi 2. Algoritma HOPNET dapat digunakan mencari *multiple path* dari *node* asal ke *node* tujuan. Mekanisme routing yang sudah dijelaskan dalam HOPNET tersebut juga dilakukan jika zona routing mengalami perubahan.

3.4 Algoritma Simulated Annealing

Algoritma *Simulated Annealing* (SA) mensimulasikan proses *annealing* pada pembuatan materi yang terdiri dari butir kristal *(glassy)* atau logam. Tujuan dari proses ini adalah menghasilkan struktur kristal yang baik dengan menggunakan energi seminimal mungkin. Ide SA berasal dari suatu makalah yang dipublikasikan oleh Me-tropolis pada tahun 1953. Jika kita memanaskan suatu materi keras hingga mencair dan kemudian mendinginkannya, maka sifat struktur dari materi tersebut bergantung pada tingkat pendinginan. Jika materi cair didinginkan secara perlahan, maka akan dihasilkan kristal-kristal yang berkualitas baik. Sebaliknya, jika materi cair didinginkan secara cepat, kristal-kristal yang terbentuk tidak akan sempurna. Algoritma yang diusulkan Metropolis mensimulasikan materi sebagai suatu sistem dari partikel-partikel. Algoritma tersebut mensimulasikan proses pendinginan yang secara bertahap menurunkan suhu sistem hingga konvergen pada keadaan beku dan stabil.

Pada tahun 1983, Kirkpatrick dan koleganya menggunakan ide dari algoritma Metropolis dan mengaplikasikannya pada permasalahan optimasi. Idenya adalah bagaimana menggunakan *simulated annealing* untuk mencari solusi-solusi yang layak dan konvergen pada solusi optimal.

3.4.1 Aplikasi Algoritma Simulated Annealing

STUDIKASUS : *JOB SHOP SCHEDULING*

Misalkan terdapat ***n*** buah *job* atau pekerjaan $\{J_1, J_2,\ldots,J_n\}$ yang akan diproses pada m buah mesin $\{M1, M_2,...,M_m\}$. Waktu yang diperlukan untuk mengerjakan *job* J_i menggunakan mesin M_j. (operasi O_{ij}) adalah $t_{ij.}$ Masalahnya adalah bagaimana membuat jadwal *(schedule)* yang berupi urutan pengerjaan yang optimal berdasarkan kriteria tertentu, misalkw) total waktu penyelesaian semua *job (completion time* atau *makespan).*

Untuk mempermudah masalah, dibuatlah asumsi-asumsi berikut ini:

- Pada waktu $t = 0$, seluruh *job* sudah siap untuk dikerjakan. Tipe masalah *job shop* seperti ini disebut juga masalah job *shop* statis.
- Semua *job* harus melewati semua mesin sehingga jumlah operasi dalam setiap *job* adalah sama, yaitu sama dengan jumlah mesin.
- Operasi-operasi dalam suatu *job* mempunyai urutan pengerjaan *(routing)* tertentu dan tidak ada *routing* alternatif untuk tiap *job.* Tiap operasi hanya dapat dikerjakan oleh satu mesin tertentu sesuai dengan *routing-nya.*
- Tidak ada penghentian operasi yang sedang dikerjakan. Hal ini disebut sebagai *non-preemptive scheduling.*
- Pada suatu waktu tertentu setiap *job* hanya dapat dikerjakan pada satu mesin tertentu. Demikian pula, pada suatu waktu tertentu setiap mesin hanya dapat mengerjakan satu operasi dari satu *job* dan tidak terdapat mesin paralel.
- Waktu proses, *routing, due date,* dan kriteria performansi bersifat deterministik (sudah ditetapkan sebelumnya dan tidak berubah). Waktu *setup* mesin dan waktu transportasi *job* antarmesin diabaikan. Mesin selalu tersedia dan tidak pernah mengalami kerusakan. Sumber-sumber lain di luar mesin tidak diperhatikan.

Untuk *job shop* dengan n *job* dan *m* mesin, jumlah jadwal yang mungkin adalah sangat besar, yaitu (n!)m. Untuk $n = 5$ dan $m = 6$, jumlah kemungkinan , jadwal sudah mencapai 2.985.984.000.000 (hampir 3 trilyun). Tetapi, tidak semua kemungkinan jadwal tersebut valid. Sebuah jadwal dikatakan valid jika urutan

pengerjaan operasi-operasi dalam suatu *job* memenuhi routing yang telah ditetapkan, dan tidak ada tumpang tindih *(overlap) waktu* pengerjaan dari operasi-operasi yang dikerjakan pada mesin yang sama. Masalah *job shop* berhasil diselesaikan jika waktu awal pengerjaan dari setiap operasi telah diperoleh dan ke dua syarat di atas dipenuhi.

Untuk mendeskripsikan masalah *job shop,* di sini kita akan menggunakan notasi berikut:

V = {0, 1,..., *n]* ialah himpunan operasi; operasi 0 dan *n* adalah operasi *dummy* untuk mulai dan selesai. Misalnya untuk masalah 4-job dan 3-mesin, operasi-operasi diberi nomor mulai dari 1 (operasi pertama pada *job* ke-1) sampai dengan 12 (operasi terakhir pada *job* ke -3). Dengan penambahan operasi *dummy*, maka himpunan operasinya menjadi V = {0,1,13).

M = himpunan mesin

A = himpunan pasangan operasi yang berurutan dalam suatu *job*

E_k = himpunan pasangan operasi yang dikerjakan pada mesin *k*

p_i = waktu pengerjaan operasi *i*

t_i = titik waktu di mana operasi mulai dikerjakan

Masalah *job shop scheduling* dapat diformulasikan sebagai: **minimasi t_n**

$t_i > 0$ i EV dimana (E = Elemen)

$t_j - t_i > pi$ (i,j) E A

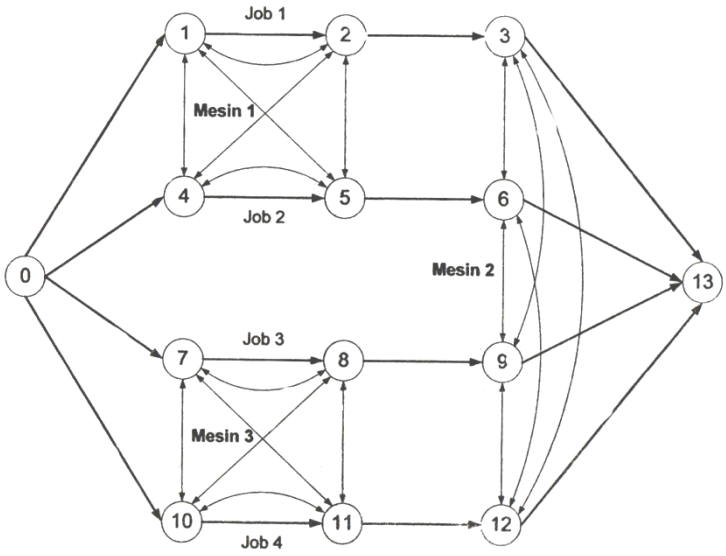
$t_j - t_i > pi$ V $t_i - t_j \geq p_j$ *(i,j)* E *Ek, k E* M

Untuk mempermudah memahami masalah *job shop scheduling,* kita bisa memodelkannya menggunakan representasi *disjunctive graph. Graph* ini dinotasikan dalam bentuk G = (V, A, E} di mana :

V = himpunan *vertex* (simpul) yang menyatakan operasi-operasi. V = O U {0} U {N + 1}, di mana O adalah himpunan seluruh operasi pada masalah job shop; N = I O I adalah jumlah seluruh operasi, sedangkan {O} dan {N+1} adalah vertex khusus yang mewakili vertex awal dan vertex akhir.

A = himpunan busur *conjunctive* (berarah) yang menghubungkan operasi-operasi dalam satu job yang sama. A = {(i,j): operasi i adalah predesesor langsung dari operasi dalam *job* J*i* (= J_i)} U {((), j); j E O} U {(i N +1) *;* i EO}; dan

E = himpunan busur *disjunctive* (bolak balik) yang menghubungkan operasi-operasi yang dikerjakan pada mesin yang sama. E = *Mi = Mj* atau operasi *i* dan *j* dikerjakan pada mesin yang sama, i, j E O}. Pada tiap *vertex* i EO, diberikan sebuah bobot d_i *; vertex* 0 dan N+l memiliki bobot 0. Waktu awal dari *vertex* 0 dan waktu akhir dari *vertex* N+1 merepresentasikan waktu awal dan waktu penyelesaian dari masalah *job shop* tersebut. Jadi, penyelesaian masalah *job shop* sama dengan penentuan orientasi (arah) dari busur-busur disjungtif dalam *graph* G sedemikian rupa sehingga *graph* G tidak mengandung *cycle* dan lintasan terpanjang dari *vertex* 0 menuju *vertex* N+l diminimalkan.



Gambar 7 *Contoh masalah job shop scheduling, dengan 4 job dan 3 mesin, yang dimodelkan menggunakan representasi disjunctive graph.*

Gambar 7 mengilustrasikan suatu masalah *jobshop scheduling, dengan 4 job dan 3 mesin, yang dimodelkan menggunakan representasi disjunctive graph.* Dengan asumsi bahwa seniua *job* harus melewati semua mesin, maka terdapat 12 operasi ditambah awal dan akhir, sehingga himpunan operasi (atau *vertex* dalam *graph)* adalah V = {0, 1, 13}. Pada gambar tersebut, misalkan *job* 1 terdiri dari tiga operasi 1, 2, dan i gedangkan *job 2* terdiri dari operasi 4, 5, dan 6. Demikian seterusnya untuk *job* **3** dan 4. Pada gambar juga terlihat bahwa operasi 1, 2, 4, dan 5 dikerjakan oleh mesin 1, operasi 3, 6, *9,* dan 12 dikerjakan oleh mesin 2, dan sisanya dikerjakan oleh mesin 3.

Selanjutnya kita bahas bagaimana menerapkan algoritma **SA** unntuk masalah *job shop scheduling* ini. Terdapat tiga tahap yang perlu didefenisikan, yaitu:

(a) Pembuatan jadwal awal

Solusi (jadwal) awal untuk algoritma SA dapat dibuat secara acak atau dengan menggunakan metode heuristik tertentu. Pembuatan solusi awal secara acak berpeluang menghasilkan *graph* jadwal yang mengandung *cycle* (tidak valid), terutama pada kasus yang berukuran besar. Sehingga proses pembuatan solusi awal perlu diulang berkali-kali. Oleh karena itu, untuk pembuatan solusi awal biasanya digunakan metode heuristik, seperti algoritma Schrage.

(b) Evaluasi fungsi biaya dan penentuan lintasan kritis

Setelah diperoleh sebuah *graph* untuk jadwal awal, dihitung nilai *earliest start time* (ES) dan *latest start time* (LS) dari setiap operasi dalam *graph* dengan menggunakan *critical path method* (CPM). *Makespan* jadwal adalah nilai ES atau LS dari operasi terakhir (operasi *dummy* N+l). Pada proses komputasi ini juga sekaligus diidentifikasi apakah terdapat *cycle* dalam *graph.* Jika dalam *graph* terdapat *cycle,* maka jadwal tersebut ditolak dan dibuat jadwal baru. Setelah menghitung *makespan* jadwal, selanjutnya diidentifikasi lintasan kritis dalam *graph,* yaitu himpunan busur-busur dari *vertex* pertama menuju *vertex* terakhir yang memenuhi syarat berikut:

• Nilai ES dan LS dari setiap *vertex* yang dihubungkan oleh busur-busur tersebut harus sama.

• Untuk busur *(u,v),* hasil penjumlahan *start time* dan waktu pengerjaan operasi *u* harus sama dengan *start time* operasi *v.*

(c) Pembuatan tetangga baru

Ketetanggaan (Neighbourhood) dari suatu jadwal adalah himpunan jadwal baru yang dapat diperoleh dengan menerapkan fungsi transisi terhadap jadwal tersebut. Fungsi transisi dalam kasus penjadwalan *job shop* adalah memilih *vertex v* dan *w* sedemikian rupa sehingga:

• *v* dan *w* adalah dua operasi berurutan sembarang yang dikerjakan pada mesin *k.*

• busur (v, w) E E_i adalah sebuah busur kritis, atau *(v,w)* berada pada lintasan kritis dari *graph.*

Fungsi transisi akan memilih satu solusi feasibel secara acak dari himpunan *neighborhood,* yang berisi semua kemungkinan solusi yang dapat diperoleh dengan penukaran urutan terhadap operasi-operasi semesin yang posisinya bersebelahan pada lintasan kritis. Sebuah tetangga dari suatu jadwal dibuat dengan cara mempertukarkan urutan pengerjaan operasi *v* dan *w* pada mesin *k* atau membalikkan arah busur *(v,w).* Struktur *neighbourhood* ini didasarkan pada dua fakta bahwa:

• Pembalikan sebuah busur kritis dalam *graph* D_i tidak akan menghasilkan *graph* D_j yang *cyclic.*

• Jika pembalikan sebuah busur non kritis dalam D_i menghasilkan *graph* D_j yang *acyclic,* maka lintasan kritis *a* dalam D_j tidak mungkin lebih pendek daripada lintasan kritis *p* dalam D_i karena D_j masih memuat lintasan *p.*

Dengan cara ini, dapat dihindari pemilihan jadwal yang mengakibatkan terjadinya *cyclic graph* dan/atau tidak menghasilkan penurunan *makespan.* Struktur *neighbourhood* ini memungkinkan model SA hanya memperhatikan *graph-graph* yang valid. Jadi, transisi ini menyebabkan pembalikan busur yang menghubungkan *v* dan *w,* dari *(v,w)* menjadi *(w,v),* dan penggantian busur *(u,v)* dan *(w,x)* dengan busur *(u,w)* dan *(v,x),* di mana *u* adalah operasi sebelum *v* pada mesin *k,* dan *x* adalah operasi setelah *w* pada mesin *k.*

Algoritma SA yang lengkap untuk masalah *Job shop scheduling* diilustrasikan sebagai berikut :

1. Inisialisasi masalah Job Shop Scheduling (JSS)
2. Inisialisasi nilai parameter SA:
   - TO : temperatur awal
   - Tl : temperatur akhir
   - d : faktor penurunan temperatur
   - L : jumlah iterasi
3. Definisikan graph untuk JSS
4. Buat jadwal awal
5. Definisikan busur-busur disjunctive

6. Komputasi lintasan kritis (hitung makespan)

7. current state = jadwal awal

8. BEST-SO-FAR = current state

8. T = TO

9. i = 1

10. while i <= L or T > Tl do

repeat

a. Buat jadwal baru

b. Definisikan busur-busur disjunctive

c. Komputasi lintasan kritis (hitung makespan)

until jadwal baru tidak mengandung cycle

new state = jadwal baru

Evaluasi new state.

if new state adalah goal then

Kembalikan state ini sebagai solusi dan keluar

dari program

ELSE

Hitung △E = f(new state) - f(current state)

if AE < 0 then

current state = new state

ELSE

r = random ( ) {bangkitkan bilangan acak)

if p(△E) < 0 then

current state = new state

end

end

end

i f current state < BEST-SO-FAR then

BEST-SO-FAR = current state

end

i - i + 1 T - d * T

end

11. Kembalikan BEST-SO-FAR sebagai solusi

IV KESIMPULAN

1. Algoritma Optimasi (*Optimization Algorithms)* merupakan algoritma untuk menemukan nilai x sedemikian sehingga menghasilkan f(x) yang bernilai sekecil atau sebesar mungkin untuk suatu fungsi f yang diberikan, yang mungkin disertai dengan beberapa batasan pada x. Dimana x bisa berupa skalar atau vektor dari nilai-nilai kontinu maupun diskrit. Algoritma ini adalah generalisasi algoritma pencarian (*search algorithmn).*
2. Suatu algoritma dikatakan optimal pada *worstcase*-nya jika tidak ada algoritme lain yang sekelas yang melakukan lebih sedikit operasi dasar atau Algoritma optimal adalah yang terbaik, dimana dalam *worstcase*-nya melakukan perbandingan 2 entri dalam list yang paling sedikit.
3. Cara memeriksa optimalitas algoritma adalah dengan menentukan batas bawah dari jumlah operasi yang diperlukan untuk menyelesaikan masalah, jika sembarang algoritma yang melakukan operasi sejumlah batas bawah tersebut berarti optimal.
4. Efisiensi suatu algoritma ditentukan oleh 2 hal, yaitu :
    a. Kecepatan (Waktu), faktor-faktor yang mempengaruhinya adalah banyak langkah, tipe data dan operator-operator.
    b. Space memori (Alokasi Memori). Space memori dipengaruhi oleh struktur data dinais, procedure / function call dan recursif.
5. Salah satu aplikasi penggunaan algoritma yang optimal adalah bagaimana menentukan radius optimal pada algoritma routing hopnet dalam jaringan *Mobile Ad-Hoc.* Teknik ini menemukan nilai total trafik minimal yang dihitung dari trafik dalam zona dan antar zona. Penghitungan ini dilakukan pada masing-masing nilai radius yang diberikan secara *increment* pada waktu tertentu.
6. Permasalahan pada *job shop schedulling* dengan 4 job dan 3 mesin, yang dimodelkan menggunakan representasi disjunctive graph, dapat

diselesaikan dengan baik menggunakan Algoritma Probabilistik Simulated Annealing.

7. Setiap algoritma memiliki kekurangan dan kelebihan masing-masing. Tidak ada satupun algoritma yang berlaku umum dan bisa digunakan untuk menyelesaikan semua jenis masalah, termasuk dalam berbagai permasalahan optimasi. Pencarian dan pemilihan Algoritma optimasi yang sesuai menjadi faktor kunci untuk menyelesaikan setiap masalah yang dihadapi.

V. REFERENSI

Cormen TH, Leiserson CE, Rivest RL, Stein C. 2001. *Introduction to Algorithms, Second Edition*, Massachusetts Institute of Technology, MIT Press.

DEPDIKNAS. 2007. *Kamus Besar Bahasa Indonesia*. Balai Pustaka. Jakarta

http://en.wikipedia.org/wiki/Asymptotically_optimal_algorithm , diakses pada tanggal 20 September 2010

http://id.w3support.net/index.php?db=so&id=343667 , diakses pada tanggal 19 September 2010

http://ijcai.org/papers09/Papers/IJCAI09-316.pdf , diakses pada tanggal 20 September 2010, pukul 12.26 wib

http://ocw.ui.ac.id/materials/.../07-SO0910-Algoritma_Pergantian_Page.pdf , diakses pada tanggal 17 September 2010, pukul 09.53 wib

http://web.mit.edu/k_lai/www/6.046/r6-handout.pdf , diakses pada tanggal 20 September 2010, pukul 12.28 wib

http://www.cs.jhu.edu/~yairamir/cs418/os6/sld018.htm , diakses pada tanggal 20 September 2010

http://www.eepis-its.edu/id/proceeding/196/Penyelesaian-Persoalan-Multidimensional- Knapsack-(pmk)-Dengan-Algoritma-Genetik, diakses pada tanggal 17 September 2010

http://www.informatika.org/~rinaldi/Stmik/Algoritma%20Greedy.ppt , diakses pada tanggal 17 September 2010

Munir Rinaldi. 1999, *Algoritma dan Pemrograman dalam bahasa Pascal dan C*, Informatika, Bandung.

Nurdiati S. 2010. *Materi Kuliah Analisis Desain Algoritma*, Departemen Ilmu Komputer IPB, Bogor

Suyanto. 2010. *Algoritma Optimasi Deterministik atau Probabistik*, Graha Ilmu, Yogyakarta.

Andi Hasad menempuh pendidikan di program studi Teknik Elektro (S1) UNHAS, Makassar, kemudian melanjutkan di Ilmu Komputer (S2) IPB, Bogor. Penulis pernah menimba ilmu dan pengalaman di berbagai perusahaan / industri di Jakarta dan Bekasi. Saat ini menekuni profesi sebagai dosen tetap di Fakultas Teknik UNISMA Bekasi dan aktif dalam pengembangan ilmu di bidang *robotics*, *electronic instrumentation* , *intelligent control*, *knowledge management system*, *network* dan *cryptography*. Info lengkap penulis dapat diakses di http://andihasad.wordpress.com/