Tugas Makalah
IF-2153 (Matematika Diskrit)
Prodi Informatika STEI-ITB

Penjadwalan *Job-Shop* dengan Algoritma Simulated Anealing

Halim Munawar -NIM: 13505106
Program Studi Teknik Informatika, STEI-Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail: if15106@students.if.itb.ac.id

Abstrak

Tujuan dari pembuatan makalah ini adalah untuk mengkaji dan mempelajari sejauh mana pengaplikasian teori graf dalam penyelesaian masalah penjadwalan *job-shop* (JSSP). Penulis memilih algoritma Simulated Anealing untuk memecahkan JSSP, dengan alasan lebih powerful dibandingkan dengan algoritma lain dan lebih populer, sehingga lebih mudah dalam pencarian sumber literature.

Karena tujuan dari penulisan makalah ini dan keterbatasan penulis, pada makalah ini hanya akan dibahas penerapan algoritma Simulated Anealing langkah per langkah, tanpa disertai pembuktian keefektifannya. Walaupun begitu penulis akan melampirkan hasil pengukuran kualitas solusi algoritma SA yang diperoleh dari makalah penulis lain.

1. Pendahuluan

Masalah penjadwalan *job-shop* (JSSP) merupakan salahsatu masalah optimasi kombinatorial non deterministic dengan waktu polynomial *(NP-complete)* yang paling sulit. Waktu komputassi untuk mencari solusi optimal meningkat secara eksponensial seiring dengan membesarnya nilai parameter masalah jumlah mesin dan jumlah job.

Berbagai macam cara telah dikembangkan untuk memecahkan masalah ini, diantaranya dengan menggunakan Algoritma Genetic dan Algoritma Simulated Anealing (SA). Namun beberapa penelitian menyimpulkan bahwa Algoritma SA dapat menghasilkan solusi yang optimal atau mendekati optimal dengan waktu yang cukup singkat. Dengan mempertimbangkan kenyataan tersebut penulis memilih algoritma SA sebagai bahan pengkajian makalah ini.

Yang menjadi fokus utama makalah ini, bukanlah pembuktian kekuatan algoritma SA tersebut, melainkan lebih kepada penijauan sejauh mana teori graf diaplikasikan dalam penyelesaian JSSP, yang dalam makalah ini

solusi JSSP tersebut dihasilkan oleh algoritma SA. Oleh sebab itu makalah ini akan lebih menekankan pada langkah-langkah pembangunan algoritma SA tersebut.

2. Masalah Penjadwalan *Job-shop*

Secara umum pemasalah job-shop didefinisikan sbagai berikut:

- Terdapat sejumlah job yang dalam hal ini penulis akan menganalogikannya dengan sejumlah bahan baku, yang harus diproses, setiap bahan baku ini dilambangkan dengan B. Berarti B_i melambangkan bahan baku ke-i. Apabila ada **n** bahan baku, maka himpunan bahan baku dapat ditulis **$\{B_1, B_2, B_3, \ldots, B_n\}$.**

- Kemudian setiap job diatas akan diolah pada sejumlah mesin. Urutan proses pengolahan bahan baku tersebut tidak harus sama. Setiap mesin dilambangkan dengan **M.** Apabila ada

m mesin maka himpunan-himpunan mesin tersebut dapat ditulis **{M_1, M_2, M_3, ….,M_m}.**

- Operasi pengolahan bahan baku **B_i** pada mesin **M_j** dilambangkan dengan **O_{ij}** ,operasi ini memiliki bobot waktu yang dilambangkan dengan **t_{ij}** .

Masalahnya adalah menentukan pengurutan operasi-operasi tersebut berdasarkan kriteria tertentu sehingga dihasilkan jadwal yang optimal. Dalam hal ini yang menjadi kriteria optimalnya suatu jadwal adalah lamanya waktu sampai semua bahan baku selesai diproses (*completion time* atau *makespan*).

JSSP yang dikaji dalam makalah ini diasumsikan sebagai berikut:

- Pada waktu t=0 semua bahan baku sudah siap untuk di olah. Tipe masalah job-shop seperti ini disebut juga masalah *job-shop* statis.

- Setiap bahan baku harus diolah pada semua mesin, sehingga jumlah operasi pada setiap bahan baku adalah sama.
  Contoh:
  Apabila ada **n** macam bahan baku **{B_1, B_2, B_3, …,B_n}** dan **m** buah mesin **{M_1, M_2, M_3, ….,M_m}**, maka terdapat **m** macam operasi pada setiap bahan baku. **{ O_{i1}, O_{i2}, O_{i3}, …, O_{im}}** merupakan himpunan operasi pada bahan baku ke-**i**. Jadi jumlah semua operasi adalah **mxn**.

- Setiap bahan baku memiliki urutan pengerjaan (*technological sequence/routing*) tertentu tidak ada *routing* alternatif bagi setiap bahan baku. Setiap operasi hanya dapat dikerjakan pada satu mesin tertentu sesuai urutan pengerjaannya, tidak ada penghentian proses operasi yang sedang berlangsung.

- Pada suatu waktu tertentu setiap bahan baku hanya dapat diolah pada satu mesin tertentu. Begitu pula sebaliknya setiap mesin hanya dapat mengolah satu macam bahan baku tertentu. Tidak ada mesin palarel, atau tidak ada mesin yang dapat mengerjakan lebih dari satu operasi pada waktu yang bersamaan.

- Waktu proses, routing dan criteria performansi besifat deterministic(sudah ditentukan sebelumnya dan tidak berubah). Waktu setup ,mesin dan waktu transportasibahan baku antar mesin diabaikan. Mesin selalu tersedia dan kondisi mesin pada setiap waktu dianggap sama. Sumber-sumber lain yang mendukung proses pengolahan selain mesin tidak diperhatikan.

JSSP dengan **m** mesin dan **n** buah bahan baku akan menghasilkan $(n!)^m$ buah solusi atau jadwal, namun tidak semua solusi tersebut valid. Suatu jadwal/solusi dikatakan valid apabila memnuhi criteria berikut:

- Urutan pengerjaan operasi pada setiap job cocok dengan *routing* yang telah ditetapkan.

- Tidak ada *overlap* waktu pengerjaan pada operasi-operasi yang berpadanan dengan mesin yang sama.

Masalah penjadwalan *job-shop* berhasil dipecahkan jika waktu awal pengerjaan setiap operasi telah diperoleh dan dua criteria di atas telah dipenuhi.

3. Pemodelan JSSP dengan Graf

Untuk mendeskripsikan masalah *job shop*, digunakan notasi :

V = {0, 1, ..., n} ialah himpunan operasi; operasi 0 dan n adalah operasi *dummy* untuk ‘mulai’ (*start*) dan ‘selesai’ (*finish*).
Misalnya untuk masalah 3-bahan baku 3-mesin,
operasi-operasi diberi nomor mulai dari 1 (operasi pertama pada bahan baku ke-1) sampai
dengan 9 (operasi terakhir pada bahan baku ke -3).
M = himpunan mesin
A = himpunan pasangan operasi yang berurutan dalam suatu bahan baku
Ek = himpunan pasangan operasi yang dikerjakan pada mesin k
pi = waktu pengerjaan dari operasi i
ti = titik waktu dimana operasi i mulai dikerjakan

Formulasi masalah penjadwalan *job shop* adalah sbb.:
minimasi tn
$ti \geq 0\ i \in V$ **(1)**
$tj - ti \geq pi\ (i,j) \in A$ **(2)**
$tj - ti \geq pi \vee ti - tj \geq pj\ (i,j) \in E\ k, k \in M$ **(3)**

Representasi graf berarah (*directed graph*) sangat cocok untuk memodelkan permaslahan penjadwalan *job-shop*. Graf ini dinotasikan dalam bentuk:

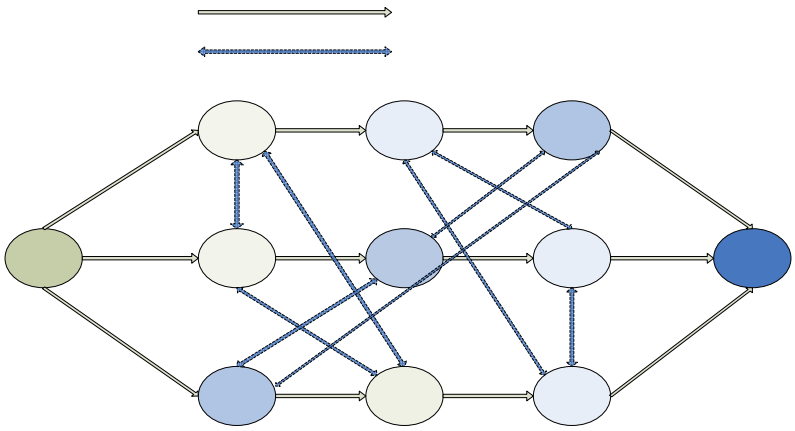
G = {V,A,E}

V = himpunan node yang merepresentasikan operasi-operasi.

A = himpunan busur *conjuctive* yang menghubungkan operasi-operasi pada bahan baku tertentu.

E = himpunan busur disjunctive yang menghubungkan operasi-operasi pada mesin yang sama.

Gambar 1 merupakan graf yang merepresentasikan *job-shop* 3-mesin 3-job atau bahan baku(11 operasi dengan 2 operasi *dummy*). Setiap bahan baku memiliki 3 operasi sesuai dengan jumlah mesinnya. Node-node dalam graf merupakan himpunan V. Tiap node dalam graf mewakili satu operasi. Setiap bahan baku memiliki 3 operasi yang berpadanan sesuai dengan jumlah mesinnya dengan *route/technological sequence* yang berbeda-beda. Node 0 dan node 10 adalah operasi-operasi fiktif(*dummy*) mewakili operasi awal dan akhir. Node yang memiliki warna yang sama dikerjakan pada mesin yang sama. Operasi-operasi yang dilakukan pada bahan baku yang sama, dihubungkan oleh busur *conjuctive* berwarna krem pada graf G. Dan merupakan element dari himpunan A. Busur ini merepresentasikan kondisi **(2)** pada formulasi masalah penjadwalan *jobshop*. Urutan pengerjaan operasi setiap bahan baku ditunjukkan arah panah. Urutan ini tidak dapat diubah. Bahan-baku ke-1 memiliki urutan operasi 1-2-3. Bahan-baku ke-2 memiliki urutan operasi 4-5-6. *Predesor* (operasi yang mendahului) langsung operasi 2 pada Bahan-baku ke-1 adalah operasi 1, dan Successor (operasi yang mengikuti) langsung operasi 8 pada Bahan-baku ke-3 adalah operasi 9.



Gambar 1 Contoh representasi graf untuk masalah *job-shop*

Operasi-operasi yang dikerjakan oleh satu mesin dihubungkan oleh busur-busur *disjunctive* (garis putus-putus dalam graf G). Dan merupakan element himpunan E. Busur-busur ini merepresentasikan kondisi (3) dalam formulasi masalah penjadwalan *job shop* di atas. Dalam graf tersebut, urutan pengerjaan operasi pada tiap mesin belum ditentukan. Operasi 1, 4, dan 8 dikerjakan oleh mesin ke-1; operasi 2, 7, dan 6 dikerjakan oleh mesin ke -2; operasi 3, 5, 9 dikerjakan oleh mesin ke-3.

Sekarang yang menjadi masalah utama penjadwalan *job-shop* adalah merubah seluruh busur *disjunctive* di atas menjadi busur *conjuctive*, sehingga proses 10 atau proses *dummy* akhir dapat dimulai dengan waktu seminimal mungkin.

4. Algoritma Simulated Anealing

Algoritma SA diperkenalkan oleh Metropolis *et al*. pada tahun 1953, dan aplikasinya dalam masalah optimasi dilakukan pertama kali oleh Kirkpatrick *et al*. tahun 1983. Algoritma ini beranalogi dengan proses *annealing* (pendinginan) yang diterapkan dalam pembuatan material *glassy* (terdiri dari butir kristal). Dari sisi ilmu fisika, tujuan sistem ini adalah untuk meminimalisasi energi potensial. Fluktuasi kinematika acak menghalangi sistem untuk mencapai energi potensial yang minimum global, sehingga sistem dapat terperangkap dalam sebuah keadaan minimum lokal. Dengan menurunkan temperatur sistem, diharapkan energi dapat dikurangi ke suatu level yang relatif rendah. Semakin lambat laju pendinginan ini, semakin rendah pula energi yang dapat dicapai oleh sistem pada akhirnya.

Guna mensimulasikan proses evolusi menuju kesetimbangan termal untuk suatu material zat padat dalam sebuah tungku pemanas pada setiap temperatur T, Metropolis membuat algoritma sbb.:

- Jika diketahui *state current* dari zat padat (energi E), maka sebuah mekanisme gangguan digunakan untuk membuat *state* berikutnya (energi E') dengan melakukan sedikit pergeseran terhadap suatu partikel yang dipilih secara acak.

- Jika ($\Delta E = E' - E) \leq 0$, maka proses dilanjutkan dengan *state* baru ini. Jika $\Delta E > 0$, *state* yang dibuat ini diterima dengan probabilitas tertentu, yaitu *exp* ($-\Delta E/kBT$), yang disebut kriteria Metropolis.

- Agar zat padat dapat mencapai kesetimbangan termal untuk tiap nilai temperatur, proses penurunan temperatur dilakukan dengan membuat sejumlah transisi untuk setiap nilai temperatur. Temperatur merupakan parameter kunci yang mengontrol proses *annealing* dan menentukan berapa tingkat keacakan dari *state* energi.

Algoritma Metropolis juga dapat digunakan terhadap urutan-urutan konfigurasi (solusi) yang dibuat untuk sebuah masalah optimasi kombinatorial. Konfigurasi dipandang sebagai *state* dari zat padat, sedangkan fungsi *cost* F dan parameter kontrol c sebagai energi E dan temperatur T. Algoritma *simulated annealing* dapat dipandang sebagai suatu urutan algoritma Metropolis yang dievaluasi pada serangkaian nilai-nilai parameter kontrol yang semakin mengecil.

Dalam konteks optimasi, temperature adalah variabel kontrol yang berkurang nilainya selama proses optimasi. Level energi sistem diwakili oleh nilai fungsi objektif. Skenario pendinginan dianalogikan dengan prosedur *search* yang menggantikan satu *state* dengan *state* lainnya untuk memperbaiki nilai fungsi objektif. Analogi ini cocok untuk masalah optimasi kombinatorial dimana jumlah *state* terbatas namun terlalu besar untuk ditelusuri dengan cara *enumerative search*.

Diberikan sebuah contoh masalah optimasi kombinatorial (S,F), dimana i adalah konfigurasi/solusi sekarang (*current*) dengan fungsi *cost* F(i) dan j adalah konfigurasi berikutnya dengan fungsi *cost* F(j). Konfigurasi j diperoleh melalui sebuah mekanisme *generate* yang mewakili mekanisme gangguan dalam algoritma Metropolis, dan j akan diterima menggantikan i dengan suatu kriteria penerimaan yang mewakili criteria Metropolis yang didefinisikan sbb.:

$$\text{Prob(menerima j)} = \min[1, \exp(-(F(i) - F(j))/c)]$$

dimana $c \in R+$ adalah parameter control dan
$i,j \in S$ adalah dua konfigurasi yang berbeda.

Topologi sistem harus dibuat sedemikian rupa sehingga setiap titik dapat dicapai dari setiap titik lainnya. Hal ini berarti terdapat sebuah *path* dari setiap minimum lokal menuju minimum global.

Algoritma SA bertujuan untuk meminimasi sebuah fungsi objektif atau fungsi energi. Pada tahap pertama, didefinisikan sebuah solusi awal. Lalu dari solusi awal ini dibuat sebuah solusi baru, yang kemudian dibandingkan nilai fungsi objektifnya dengan solusi awal. Jika solusi baru ini lebih baik, ia akan diterima. Keunikan metode SA adalah bahwa solusi yang lebih buruk kadang-kadang dapat diterima, sehingga sistem dapat terhindar dari perangkap minimum local

(namun solusi terbaik yang pernah dicapai selalu dicatat).

Algoritma SA secara umum adalah sbb.:

A. Pilih sebuah solusi awal x0 secara acak dan tetapkan nilai temperatur awal. Pada langkah ke-i, solusi yang *current* disebut xi. Parameter kontrol adalah ci dan fi = f(xi).

B. Ulangi langkah-langkah berikut :
   1) Buat sebuah *neighbour* xp dari solusi *current* xi dan hitung nilai fungsi objektifnya.
      State xp adalah sebuah kandidat potensial untuk *state* xi+1.
   2) Set xi+1 = xp dengan probabilitas min {1,exp((fi-fp)/c i)}.
      Jika tidak, set xi+1 = xi.
      Turunkan nilai temperatur berdasarkan faktor d tertentu : ci = ci + dci.
      Tambahkan 1 pada jumlah iterasi : i = i + 1.

Kondisi terminasi algoritma dapat berupa dicapainya jumlah iterasi tertentu dimana tidak ada *state* baru yang diterima, atau temperatur mencapai nilai tertentu yang telah ditetapkan.

Algoritma ini pasti akan mengubah *state* jika nilai fungsi objektif diperbaiki. Namun dengan probabilitas tertentu (yang akan berkurang sebagai fungsi dari jumlah iterasi), *state* dapat digantikan dengan yang lebih buruk (namun *state* terbaik tetap dicatat).

State awal dari sistem dapat dipilih secara acak atau dengan menggunakan metode heuristik tertentu. Nilai temperatur awal (T0) harus cukup besar supaya beberapa *state* awal yang dipilih dapat diterima, karena probabilitas penerimaan berkurang seiring dengan menurunnya temperature (T). Fungsi probabilitas penerimaan yang umum digunakan adalah:

$P = e^{-\Delta E/T}$; dimana
$\Delta E = E(Vnew) - E(Vold)$
T : temperatur; E : fungsi energi

sistem yang dihitung pada *state* Vnew dan Vold untuk ΔE yang lebih besar, yaitu bila *state* baru benar-benar tidak seperti yang diharapkan, probabilitas penerimaan tidak ada, dan bila ΔE negatif *state* yang baru selalu diterima.

5. Penerapan Algoritma SA Untuk Penjadwalan *Job-Shop*

Untuk mengaplikasikan algoritma SA ke dalam masalah optimasi kombinatorial, perlu didefinisikan tiga ha dengan tepat, yaitu:

- Konfigurasi (dalam masalah *job-shop* berarti konfigursi jadwal).
- Fungsi biaya(*cost function*).
- Struktur *neighbourhood.*

Model algoritma SA untuk penjadwalan *job-shop* ini dibagi ke dalam 4 tahapan, yaitu:

a) Tahapan pemilihan jadwal awal

Diketahui graf berarah G = {V,A,E} untuk masalah *job-shop* yang akan diselesaikan.

Algoritma Giffler dan Thompson digunakan untuk mendefinisikan jadwal awal. Algoritma ini membuat sebuah jadwal dengan memperhatikan semua operasi pada semua mesin. Kriteria yang digunakan adalah nilai ES(*earliest start time*) dan waktu pengerjaan dari tiap operasi.

Algoritma Giffler Thompson :

1) Set himpunan D yang elementnya merupakan operasi pertama pada setiap bahan-baku

2) Cari operasi pada himpunan D yang memiliki waktu penyelesaian paling singkat. Set **T** sama dengan waktu tersebut.

3) Untuk setiap mesin yang memiliki 1 atau lebih operasi yang diselesaikan dalam waktu T, cek untuk konflik antara operasi-operasi tersebut dan operasi yang diselesaikan pada waktu berikutnya.
   Himpunan konflik pada setiap mesin terdiri dari:
   - Semua operasi yang mempunyai waktu penyelesaian sama.

- Semua operasi yang overlap dengan operasi pada langkah 2.

4) Pada suatu mesin yang padanya terdapat >= 1 operasi pada himpunan konflik yang ditentukan pada langkah 3, pilih salah satu operasi. Apabila mungkin *Left-shift* operasi ini . Ganti waktu penyelesaian setiap operasi pada himpunan konflik dengan menambahkan waktu pengerjaannya dengan waktu penyelesaian operasi yang dipilih tadi.

5) Untuk setiap operasi baru yang diperoleh dari langkah ke-4, lihat operasi selanjutnya. Masukkan nilai penyelesaiannya ditambah dengan waktu pengerjaan baru dari operasi selanjutnya.

6) Apabila T bukan waktu paling besar dari entri-entri array waktu penyelesaian operasi. Pilih nilai pada array yang lebih besar selanjutnya dari T. Kembali kelangkah 2.

7) Apabila T entri paling besar pada array. Stop looping. Waktu penyelesaian dari jadwal yang dihasilkan pasti T.

b) Tahap evaluasi fungsi biaya dari jadwal.

Setelah diperoleh sebuah graf untuk jadwal awal dihitung nilai ES(early start time) dan LS(latest start time setiap) dari setiap operasi dalam graf dengan menggunakan *critical path method* (CPM). Makespan jadwal adalah nilai ES atau LS dari operasi dummy terakhir. Nilai ini disebut juga biaya dari jadwal.

Gambar2.

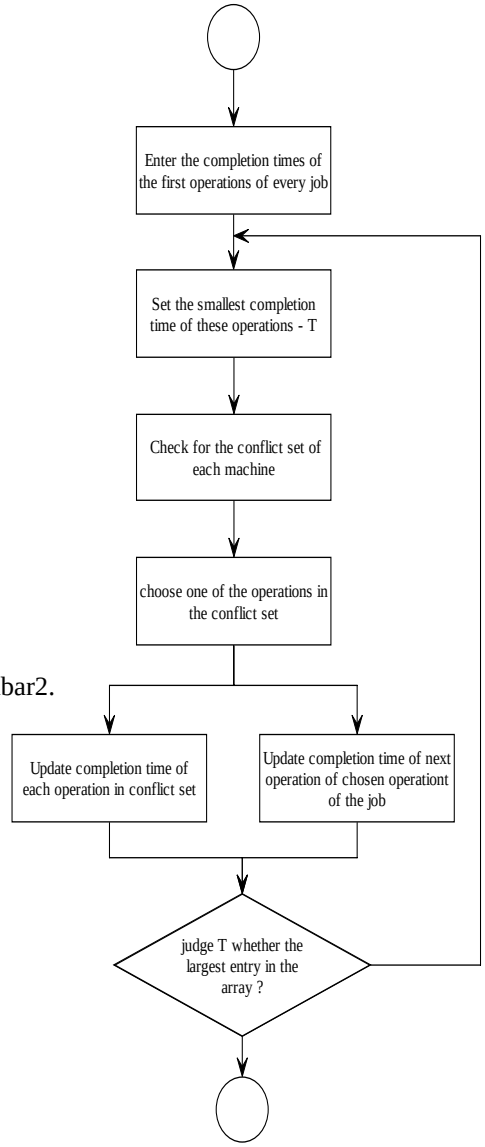


Diagram Alir GT Algoritma

c) Tahap komputasi lintasan kritis.

Setelah menghitung makespan jadwal, diidentifikasikan lintasan kritis dalam graf, yaitu himpunan busur-busur dari node yang memenuhi kriteria berikut:

- Nilai ES dan LS dari setiap *node* yang dihubungkan oleh busur-busur harus sama atau dengan kata lain operasi yang merupakan tetanggannya dikerjakan sesaat setelah operasi tersebut selesai dikerjakan.

- Untuk busur $u \rightarrow v$, hasil penjumlahan *start time* dan wakru pengerjaan dari operasi *u* harus sama dengan *start time* operasi *v*.

Sebuah busur dalam lintasan kritis dibalik arahnya untuk membuat sebuah *neighbour* yang baru.

d) Tahap pembuatan *neighbour* baru

Neighbourhood dari sebuah jadwal adalah himpunan jadawal yang dapat diperoleh dengan menerapkan fungsi transisi terhadap jadwal tersebut. Pertanyaannya sekarang adalah fungsi transisi seperti apa yang menjamin tidak terbentuknya graf yang *cyclic* dan tidak terjadinya penaikan *makespan*.

Berikut pendefinisian fungsi transisi tersebut:

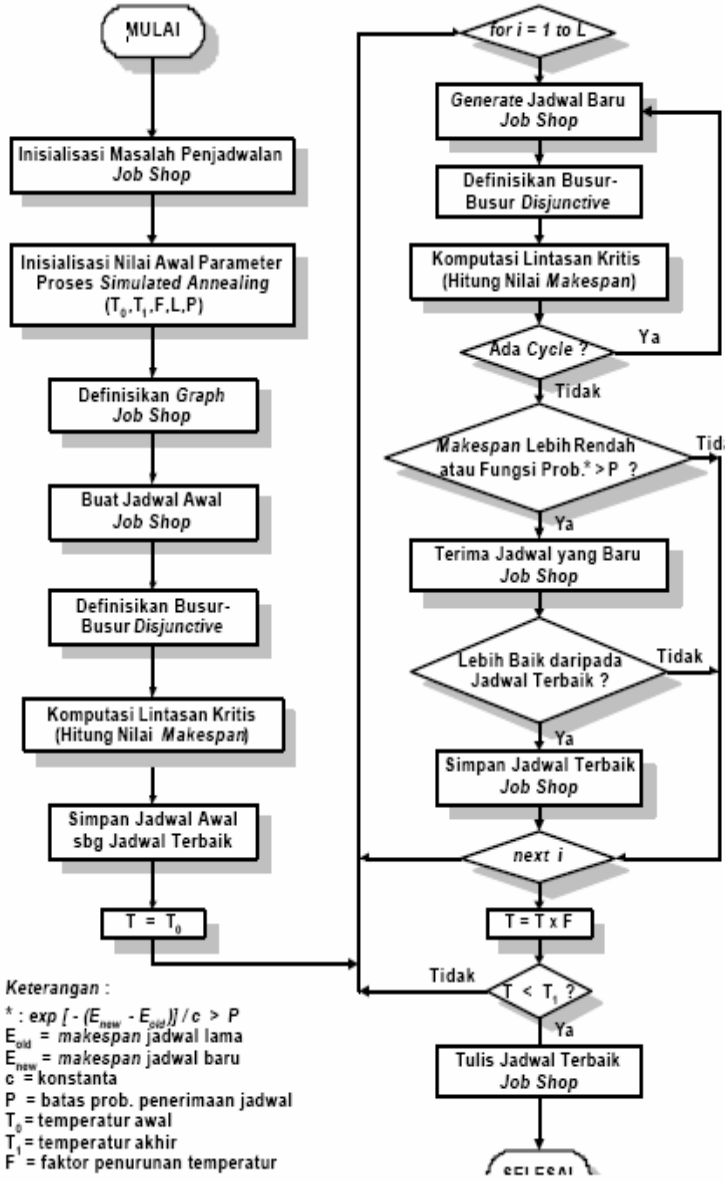
Pilih node *v* dan *w* sedemikian rupa sehingga:

- *v* dan *w* adalah dua operasi berurutan sembarang yang dikerjakan pada mesin k.
- busur $(v,w) \vee E_k$ adalah sebuah sebuah busur kritis, atau *(v,w)* berasa pada lintasan kritis dari graf.

Sebuah *neighbour*, yaiut anggota dari *neighbourhood* suatu jadwal, dibuat dengan membalikkan urutan pengerjaan operasi *v* dan *w* pada mesin k. Struktur *neighbourhood* ini didasarkan pada dua kenyataan bahwa:

- Pembalikan sebuah busur kritits dalam graf D_i tidak akan menghasilkan sebuah graf D_j yang *cyclic.*
- Jika pembalikkan sebuah busur non-kritis dalam D_i menghasilkan sebuah graf *acyclic* D_j , maka lintasan kritis q dalam D_j tidak mungkin lebih pendek dari lintasan kritis p dalam D_i karana D_j masih memuat lintasan p

Dengan cara ini, dapat dihindari beberapa jadwal yang tidak menghasikan penurunan *makespan* dan semua jadwal yang mengakibatkan terjadinya cyclic graf. Struktur *neighbourhood* ini mungkin model untuk meninjau graf-graf yang mewakili solusi yang feasible. 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*), dimana *u* adalah operasi sebelum *v* pada mesin k, dan *x* adalah operasi setelah *w* pada mesin yang sama.



Gambar3. Diagram alir Algoritma SA

6. Pembuktian Validitas Jadwal *job-shop* dari Algoritma SA

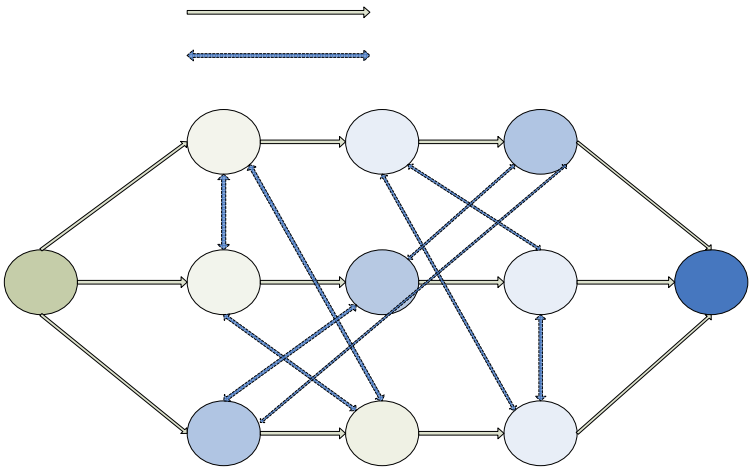
Seperti telah disebutkan di atas, dalam sebuah jadwal *job shop* yang valid urutan pengerjaan operasi-operasi dalam tiap job memenuhi *routing* yang telah ditetapkan, dan tidak ada *overlap* waktu pengerjaan dari operasi-operasi yang dikerjakan pada mesin yang sama.

Untuk membuktikan bahwa jadwal yang dihasilkan algoritma SA adalah valid, berikut disajikan sebuah contoh kasus *job shop* 3-mesin 3-bahan baku yang telah diselesaikan dengan algoritma SA.

Contoh kasus penjadwalan *job shop* 3-mesin 3-bahan baku (jumlah operasi per bahan baku = 3)
Catatan : Setiap operasi dalam job diberi nomor urut, mulai dari operasi pertama pada job ke -1 (diberi nomor urut 1) hingga operasi terakhir pada job ke -5 (diberi nomor urut 25).

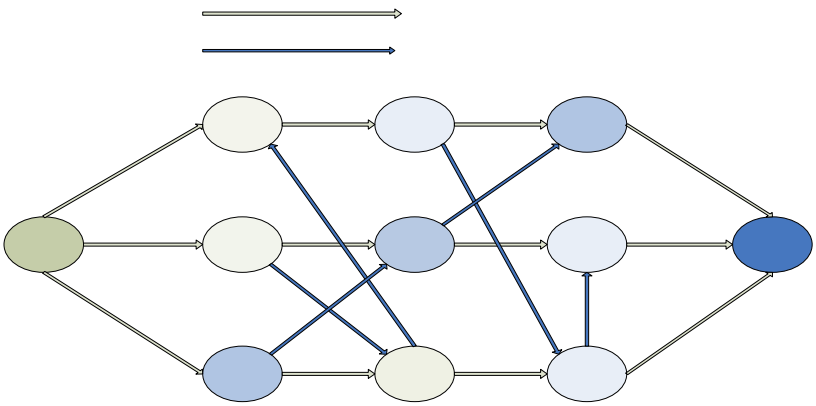
Bahan baku ke-	No Operasi(No mesin yg mengerjakan,bobot waktu operasi)		
1	1(1,67)	2(3,79)	3(2,67)
2	4(1,60)	5(2,68)	6(3,87)
3	7(2,57)	8(1,47)	9(3,92)

Kasus diatas dapat direpresentasikan dengan graf G:



Graf G

Pertama kita menentukan jadwal/state awal dengan menggunakan algoritma GT, dan akan dihasilkan jadwal awal sebagai berikut(jadwal ditulis berdasarkan mesin):
Format penulisan operasi: No operasi(ES,bobot waktu operasi,LS)
M ke-1 : 4(0,60,60) → 8(60,74,134) → 1(134,67,201)
M ke-2 : 7(0,57,57) → 5(60,68,128) → 3(280,67,347)
M ke-3 : 2(201,87,280) → 9(280,92,372) → 6(372,87,459)
Dengan makespan : 459
Direpresentasikan dengan graf D_1 :

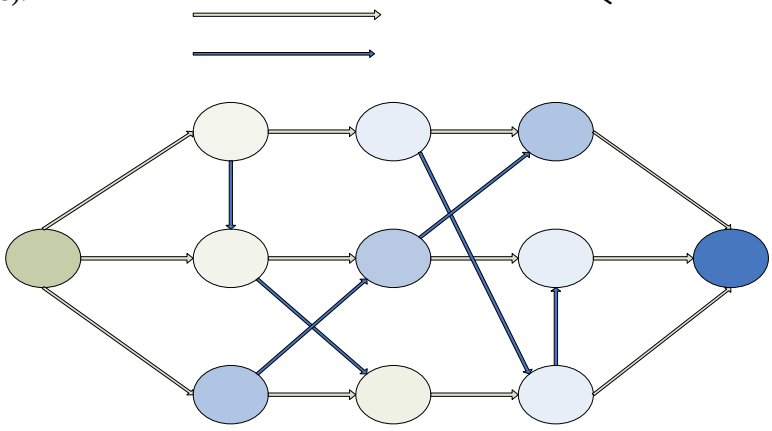


Graf D_1

Kemudian identifikasi lintasn kritis, berdasarkan spesifikasi lintasan kritis maka Graf D_1 memiliki lintasan kritis {(4,8),(8,1),(1,2),(2,3)} yaitu himpunan busur yang menghubungkan operasi-operasi pada mesin 1.

Buat *neighbour* baru dengan memilih dua *node*(opearsi) yang dikerjakan pada mesin yang sama, busur yang menghubungkan 2 *node* tersebut harus element lintasan kritis. Kemudian balik busur kritis tersebut. Contoh busur (4,1) dibalik menjadi (1,4) konsekuensinya busur (1,8) menjadi busur (4,8).

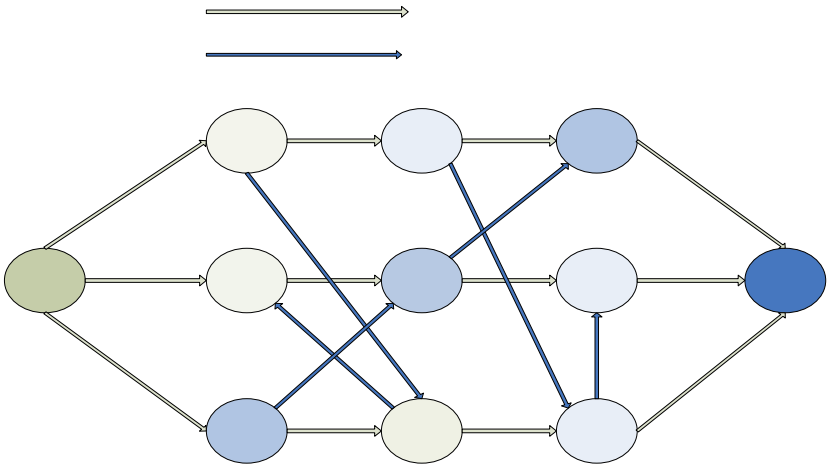
Neighbour baru tersebut (ditulis berdasarkan mesin):
M ke-1 : 1(0,67,67) → 4(67,60,127) → 8(127,74,201)
M ke-2 : 7(0,57,57) → 5(127,67,194) → 3(194,68,262)
M ke-3 : 2(67,79,146) → 9(201,92,293) → 6(293,92,**380**)
Dengan makespan : 380
Direpresentasikan dengan graf D_2



graf D_2

Neighbour ini cukup baik dibandingkan dengan jadwal awal tadi, tapi bukanlah yang terbaik.

Setelah melakukan 4 kali percobaan penyusunan jadwal yang feasible diperoleh jadwal terbaik, kita sebut dengan *neigbour* ke-n, yang jadwalnya adalah sebagai berikut:
M ke-1 : 1(0,67,67) → 8(67,74,141) → 4(141,60,201)
M ke-2 : 7(0,57,57) → 5(201,68,269) → 3(269,67,336)
M ke-3 : 2(67,79,146) → 9(146,92,238) → 6(269,87,**356**)
Dengan makespan : 356
Direpresentasikan dengan graf D_n



graf D_n

Terbukti dengan menggunakan algoritma SA kita dapat menghasilkan jadwal yang valid yaitu urutan pengerjaan operasi-operasi dalam tiap job memenuhi *routing* yang telah ditetapkan, dan tidak ada *overlap* waktu pengerjaan dari operasi-operasi yang dikerjakan pada mesin yang sama.

7. Kesimpulan

Algoritma *Simulated Anealing* merupakan salah satu metode untuk memecahkan masalah penjadwalan *job-shop*. . Keunikan metode SA adalah bahwa solusi yang lebih buruk kadang-kadang dapat diterima, sehingga sistem dapat terhindar dari perangkap minimum local (namun solusi terbaik yang pernah dicapai selalu dicatat).

Pengaplikasian graf sangat jelas terlihat dalam menyelesaikan masalah penjadwalan *job-shop* terutama dalam pemodelan JSSP dan jadwal yang dihasilkan. Graf yang digunakan disini adalah graf berarah yang terboboti.

Terdapat 2 macam busur pada representasi graf JSSP, yaitu busur *conjuctive* , yang merepresentasikan technological sequence dari tiap *job*, busur *disjunctive*, yang merepresentasikan hubungan antar node yang dilakukan pada mesin yang sama. Tujuan dari JSSP ini adalah merubah busur *disjunctive* tersebut menjadi busur conjuctive sehingga dihasilkan waktu penyelesaian semua operasi yang minimum.

8. Daftar Pustaka

i) http://www.kecl.ntt.co.jp/as/members/yamada/SA+SB.pdf. Tanggal akses: 7 Desember 2006 pukul: 15:00

ii) http://home.unpar.ac.id/~integral/Volume%207/Integral%207%20No%202/Penjadwalan%20Job%20Shop%20Statik.PDF. Tanggal akses: 23 November 2006 pukul: 16:30

iii) http://user.encs.concordia.ca. Tanggal akses: 29 Desember 2006 pukul: 19:00

LAMPIRAN

Pengukuran kualitas solusi algoritma SA ini dilakukan dengan membandingkan *makespan* jadwal produksi yang dihasilkannya terhadap *makespan* jadwal yang dihasilkan oleh sebuah perangkat lunak *scheduling* bernama *Quant System*, yang menggunakan teknik *priority dispatching* (tergolong dalam metode heuristik) dalam melakukan penjadwalan produksi. Menu program *Quant System* yang dipilih dalam penyelesaian masalah *job shop* ini adalah *option* no. 2, yaitu menerapkan semua aturan *priority dispatching* yang disediakan oleh program untuk penyusunan jadwal *job shop*.

Masalah	No	*Makespan* Jadwal	
		SA	*QS*
4-mesin 4-job	1	420	420
	2	477	477
	3	470	470
5-mesin 5-job	1	631	641
	2	626	630
	3	560	560
6-mesin 6-job	1	717	717
	2	703	711
	3	835	848
5-mesin 10-job	1	771	866
	2	783	842
	3	814	878
5-mesin 20-job	1	1411	1411
	2	1282	1336
	3	1512	1521
10-mesin 10-job	1	1147	1179
	2	1067	1121
	3	1072	1121
10-mesin 15-job	1	1149	1294
	2	1184	1189
	3	1414	1530
10-mesin 20-job	1	1382	1429
	2	1406	1454
	3	1500	1554
15-mesin 15-job	1	1301	1414
	2	1298	1350
	3	1300	1387
15-mesin 20-job	1	1609	1659
	2	1448	1557
	3	1597	1618

Tabel 2. Perbandingan *Makespan* Jadwal antara Algoritma SA dengan *Quant System* (QS)

Tabel 2 memuat data nilai *makespan* jadwal yang dihasilkan oleh algoritma SA dan *Quant System* untuk 30 kasus *job shop* berbeda dengan berbagai ukuran jumlah mesin dan jumlah job. Dari tabel ini tampak bahwa dari 30 kasus yang diselesaikan, kualitas solusi yang dihasilkan oleh program *Quant System* tidak pernah mengungguli algoritma SA dan hanya mampu menyamainya pada 6 kasus. Selebihnya algoritma SA dapat menghasilkan jadwal dengan *makespan* yang lebih baik. Perbandingan waktu komputasi tidak dilakukan dalam penelitian ini karena program Quant System tidak menyediakan fasilitas untuk penghitungan waktu komputasi. Namun dari pengalaman empirik dalam melakukan eksekusi program, algoritma SA membutuhkan waktu komputasi yang lebih besar dibandingkan *Quant System*, terutama untuk masalah-masalah *job shop* berskala besar.