2. LANDASAN TEORI

2.1 Penjadwalan

Penjadwalan bertugas untuk mengalokasikan sejumlah sumber daya yang ada untuk mengerjakan tugas seiring berjalannya waktu sebagai proses pengambilan keputusan untuk mengoptimalkan satu atau beberapa tujuan (Pinedo, 2011). Hal-hal yang perlu diperhatikan dalam melakukan penjadwalan adalah material yang digunakan, proses pengerjaan, waktu pengerjaan dari tiap proses, jumlah *job* yang harus dijadwalkan serta informasi sumber daya yang digunakan seperti jumlah mesin dan jumlah operator. Jika penjadwalan yang dilakukan tidak berjalan dengan baik maka akan sangat berbahaya bagi perusahaan dan akan ada konsekuensi-konsekuensi yang muncul akibat penjadwalan yang tidak baik tersebut.

Beberapa konsekuensi yang sering terjadi adalah *bottleneck* pada saat proses produksi berlangsung, pemanfaatan sumber daya yang rendah (% *utilization*), dan lain sebagainya. Hal-hal tersebut semuanya akan berujung kepada tidak tercapainya *deadline* yang dijadwalkan yang akan merugikan *customer* dan perusahaan itu sendiri. Tujuan dari penjadwalan adalah meminimumkan kemungkinan terjadinya konsekuensi-konsekuensi tersebut. Berikut adalah beberapa notasi yang umumnya digunakan dalam algoritma penjadwalan:

- p_j : *processing time,* yaitu waktu yang dibutuhkan untuk mengerjakan *job* j.
- C_j : *completion time,* yaitu waktu dimana *job* j selesai dikerjakan.
- r_j : *release time,* yaitu waktu dimana *job* j dapat mulai dikerjakan
- d_j : *due date,* yaitu waktu dimana *job* j harus sudah selesai dikerjakan. Bila C_j melebihi waktu d_j, maka *job* tersebut dikatakan terlambat (*tardy*).
- F_j : *flow time,* yaitu lamanya *job* j berada di lantai produksi. *Flow time* dimulai dari saat *job* j siap dijadwalkan hingga *job* j telah selesai dikerjakan.
- L_j : *lateness,* yaitu penyimpangan waktu penyelesaian dari *job* j terhadap *duedate job* j. *Lateness* dapat dihitung dengan persamaan $L_j = C_j - d_j$

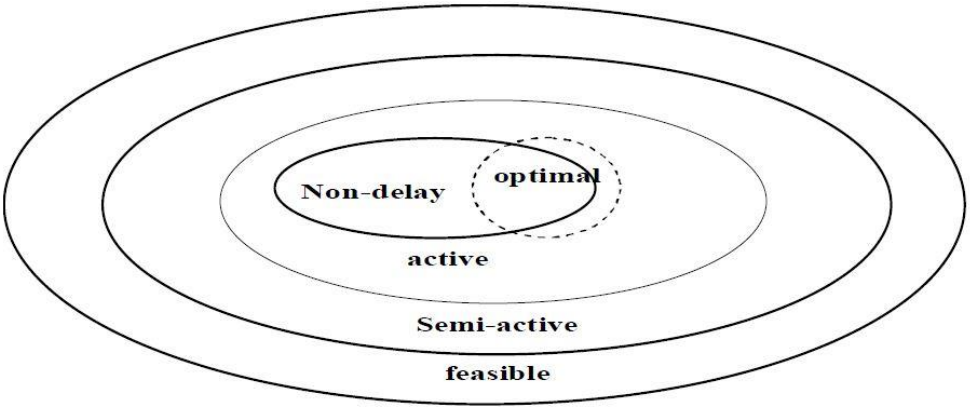
Jika Lj < 0, maka keadaan tersebut disebut dengan *earliness.*

Jika Lj > 0, maka keadaan tersebut disebut dengan *tardiness.*

- ***SL$_j$*** : *slack,* yaitu waktu kelonggaran yang tersedia untuk *job* j. *Slack* dapat dihitung dengan persamaan $SL_j = d_j - t_j$
- ***Ms*** : *makespan,* yaitu lama waktu dimulainya suatu *job* hingga semua *job* pada suatu periode selesai dikerjakan.

Jadwal terbagi dalam 3 jenis, yaitu *non-delay, active* dan *semi-active.* Definisi dari masing-masing jenis jadwal menurut Pinedo (1995) adalah sebagai berikut:

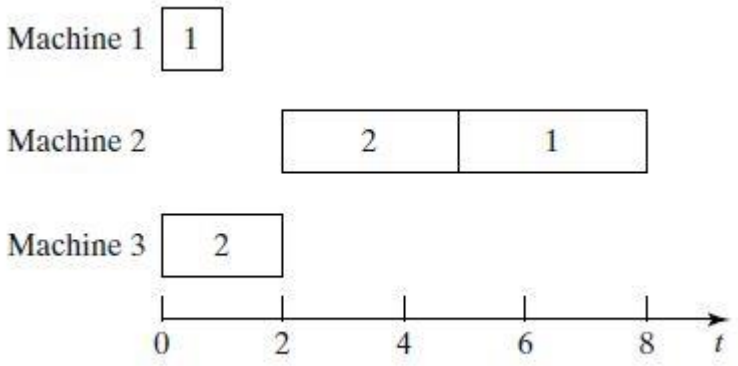
- *Non-delay* : suatu jadwal yang *feasible* dikatakan sebagai jadwal *non-delay* jika tidak ada mesin yang berstatus *idle* ketika terdapat *job* yang dapat dikerjakan.
- *Active* : suatu jadwal yang *feasible* dikatakan sebagai jadwal *active* jika tidak ada *job* yang dapat diselesaikan lebih awal tanpa merubah urutan pengerjaan *job* pada suatu mesin serta tanpa membuat job lain mengalami *delay* dari perubahan urutan pengerjaan job tersebut.
- *Semi-active*: suatu jadwal yang *feasible* dikatakan sebagai jadwal *semi-active* jika tidak ada job yang dapat diselesaikan lebih awal tanpa merubah urutan pengerjaan pada mesin manapun.



Gambar 2.1 Diagram venn tentang hubungan jadwal *non-delay, active* dan *semi-active.*

Sumber: Pinedo (2011)

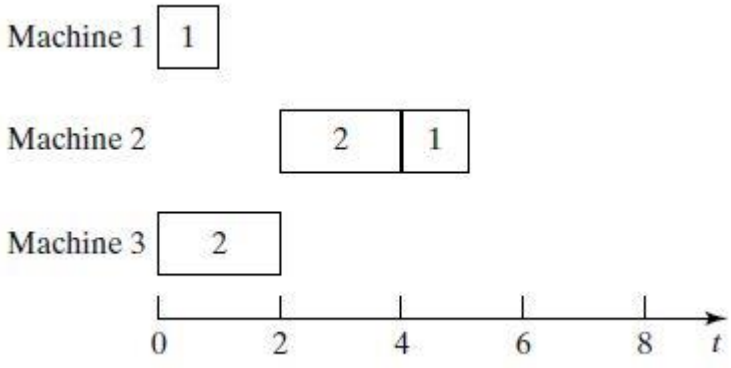
Gambar 2.1 menunjukkan hubungan antara ketiga jenis jadwal yang ada. Dari diagram venn tersebut dapat dilihat bahwa jadwal *non-delay* pasti merupakan jadwal yang active, tetapi tidak berlaku untuk sebaliknya.



Gambar 2.2 jadwal *active* tetapi bukan jadwal *non-delay*.

Sumber: Pinedo (2011)

Gambar 2.2 menunjukkan sebuah jadwal *active* namun bukan jadwal *non-delay*. Jadwal tersebut dikatakan sebagai bukan jadwal *non-delay* karena ketika mesin 2 berada dalam status *idle* sampai waktu t=2, sedangkan sebenarnya *job* 1 dapat dikerjakan pada mesin 2 pada waktu t=1. Jadwal tersebut juga dikatakan sebagai jadwal *active* karena jika pada mesin 2 urutan pengerjaan diubah menjadi 1 → 2, *makespan* yang dihasilkan akan menjadi lebih kecil, namun untuk pengerjaan *job* 2 harus ditunda hingga *job* 1 selesai dikerjakan pada mesin 2.



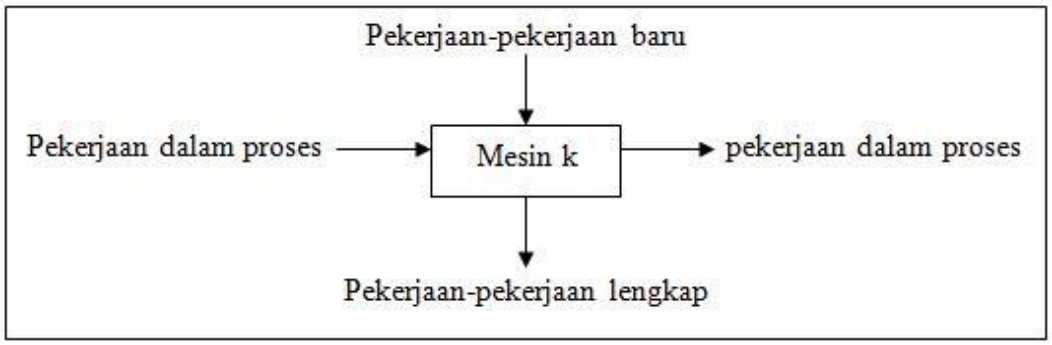
Gambar 2.3 Jadwal *semi-active* tetapi bukan jadwal *active*.

Sumber: Pinedo (2011)

Gambar 2.3 menunjukkan sebuah jadwal *semi-active* namun bukan merupakan jadwal *active.* Jadwal tersebut dikatakan sebagai jadwal *semi-active* karena waktu pengerjaan *job* 1 pada mesin 2 dapat dipercepat, sehingga *job* 1 pada mesin 2 dikerjakan pada waktu t=1 hingga t=2 tanpa harus menunda waktu mulai pengerjaan *job* 2 pada mesin 2.

2.2 *Job Shop*

Menurut Pinedo (2011), jika urutan pengerjaan suatu *job* telah pasti, tetapi urutannya tidak harus sama antara *job* yang satu dengan *job* yang lain, maka pola aliran tersebut dikenal sebagai *job shop.* pada proses produksi berpola *job shop*, proses pengerjaan setiap *job* mempunyai *routing* yang berbeda sehingga jumlah proses yang dikerjakan biasanya berbeda jumlah mesin yang dimiliki karena tidak semua *job* harus melewati seluruh mesin. Alur pengerjaan yang tidak searah ini mengakibatkan setiap *job* yang akan diproses pada suatu mesin merupakan *job* yang baru akan dimulai atau merupakan *job* yang telah dimulai pada proses sebelumnya dan merupakan barang *work in process* (WIP). Contoh pola alur pengerjaan dari *job shop* dapat dilihat pada Gambar 2.4.



Gambar 2.4 Pola Aliran *Job Shop*

Sumber : Baker (1995)

Penetapan penjadwalan yang tepat dapat memberikan proses produksi yang efektif dan efisien. Penjadwalan yang baik juga dapat mengatasi proses produksi yang mengalami keterlambatan. Pada penelitian ini, algoritma yang digunakan untuk melakukan penjadwalan adalah *Disjunctive Programming*

2.2.1 *Disjunctive Programming*

Salah satu algoritma yang dapat digunakan untuk meminimumkan *makespan* pada proses produksi berpola *job shop* adalah *disjunctive programming.* Langkah-langkah pengerjaan dengan algoritma ini adalah membuat formulasi dari mesin-mesin dan *job-job* yang akan dijadwalkan lalu formulasi tersebut akan dicari solusinya. Menurut Pinedo (2011) , berikut adalah formulasi yang digunakan dalam metode *disjunctive programming* :

Meminimumkan C_{max} :

$y_{kj} - y_{ij} \geq p_{ij}$ untuk semua (i,j) → (k,j) ϵ A.....(2.1)

$C_{max} - y_{ij} \geq p_{ij}$ untuk semua (i,j) ϵ N.....(2.2)

$y_{ij} - y_{il} \geq p_{il}$ *atau* $y_{il} - y_{ij} \geq p_{ij}$ untuk semua (i,l) dan (i,j), i=1,2,...,m....(2.3)

$y_{ij} \geq 0$ untuk semua (i,j) ϵ N.....(2.4)

dimana:

p_{ij} = waktu proses *job* j pada mesin i

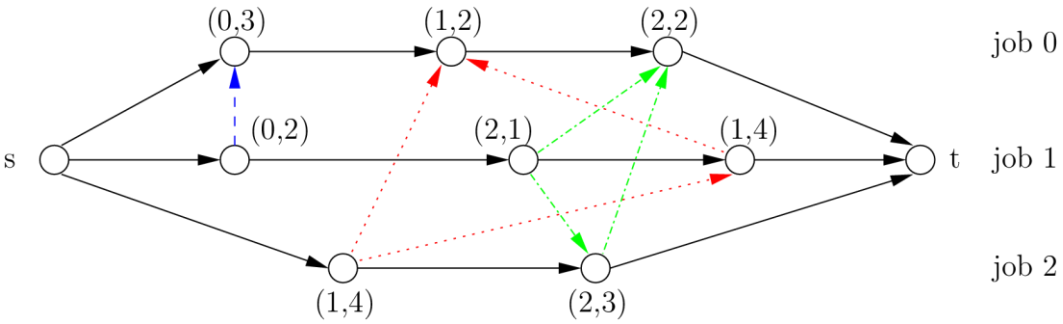
p_{il} = waktu proses *job* l pada mesin i

y_{ij} = waktu *job* j mulai dikerjakan pada mesin i

y_{il} = waktu *job* l mulai dikerjakan pada mesin i

y_{kj} = waktu *job* j mulai dikerjakan pada mesin k, dimana mesin k adalah mesin selanjutnya yang mengerjakan *job* j setelah mesin i

Pada formulasi tersebut, himpunan N merupakan himpunan dari semua operasi (i,j) yang berhubungan dengan *node* pada *directed graph*, sedangkan himpunan A merupakan himpunan dari batasan-batasan pendahulu (i,j) → (k,j) dimana *job* j harus diproses pada mesin i sebelum *job* tersebut diproses pada mesin k, dengan kata lain operasi (i,j) merupakan pendahulu dari operasi (k,j). Contoh dari *directed graph* dapat dilihat pada Gambar 2.5.



Gambar 2.5 *Directed Graph* untuk *Job Shop* 3 mesin dengan 3 *job.*

Langkah-langkah yang dilakukan dalam melakukan penjadwalan dengan Algoritma *Disjunctive Programming* adalah sebagai berikut:

Step 1 : Menetapkan Ω sebagai set mesin dan operasi pertama dari setiap *job.*

rij = 0 untuk semua (i,j) ϵ Ω.

Step 2 ; Hitung t (Ω) = $\min_{(i,j)\,\epsilon\,\Omega} \{r_{ij} + p_{ij}\}$

lalu menetapkan i* sebagai mesin yang memiliki t paling minimum.

Step 3 : Menetapkan Ω' sebagai set operasi (i*,j) pada mesin i* dengan ketentuan $r_{i*j} < t\ (\Omega)$.

Uji satu persatu setiap set sebagai operasi pertama, dan operasi lainnya pada Ω' sebagai operasi selanjutnya.

Setelah 1 set dari Ω' terpilih untuk dikerjakan terlebih dahulu, ganti set tersebut dari Ω dengan operasi selanjutnya pada *job* tersebut.

2.2.2 *Rescheduling Procedure*

Pendekatan *Rescheduling Procedure* digunakan untuk meminimasi *makespan* dalam penjadwalan *job shop.* Permasalahan yang sering terjadi adalah jadwal yang dibuat kurang optimal karena kurangnya perhatian pada urutan *job* yang diproses pada setiap mesin yang ada. Dengan cara merubah urutan dari *job* yang ada, maka akan dicapai urutan jadwal yang lebih baik melalui empat prosedur berikut:

Prosedur I

Tujuan dari prosedur ini adalah menyelesaikan *job* terakhir dengan waktu lebih cepat. Dapat dilakukan melalui beberapa tahapan berikut:

Step 1 : Menetapkan S sebagai *job* yang ditargetkan untuk diubah urutannya. Set S ← $\{J_1, J_2, \ldots, J_n\}$

Step 2 : Set j ← index *job* terakhir diselesaikan pada S dan memindahkan J_j dari S.

Step 3 : Membuat waktu mulai pengerjaan *job* menjadi seterlambat mungkin tanpa merubah urutan pengerjaan *job* tersebut pada semua mesin dan menaikkan waktu pengerjaan maksimum. Ada n waktu dan m operasi yang akan dicek waktu mulai pengerjaannya.

Step 4 : Set k ← 1.

Step 5 : Untuk menyelesaikan operasi k dari J_j lebih cepat, dilakukan perubahan urutan operasi pada mesin. Jika *feasible* (*makespan*nya tidak lebih lama dari yang ditentukan) pakai urutan yang baru, dan jika tidak maka ditolak.

Step 6 : Set k ← k+1. Jika k > m, maka akan dilakukan *Step* 7, jika tidak maka kembali ke *Step* 5.

Step 7 : Memindahkan setiap operasi yang lebih cepat tanpa operasi lain mengalami *delay*.

Step 8 : Menghitung *makespan* dari jadwal baru C_{max}. Jika C > C_{max} lalu set C ← C_{max} dan menyimpan jadwal baru dimana C sebagai *makespan* terbaik.

Step 9 : Jika S ≠ ø maka dilakukan *Step* 2, jika tidak maka prosedur I dihentikan.

Prosedur II

Tujuan dari prosedur ini untuki menyelesaikan *job* terakhir lebih cepat dan memulai *job* pertama lebih cepat agar mendapatkan jadwal yang lebih baik. Prosedur ini sama dengan prosedur I, kecuali untuk kelima tahap berikut:

Step 2 : Set j ← index *job* pertama yang dimulai dalam S dan memindahkan J_j dari S.

Step 3 : Membuat setiap waktu awal operasi sedapat mungkin lebih cepat tanpa merubah urutan pengerjaan *job* tersebut pada semua mesin.

Step 4 : Set k ← m.

Step 5 : Untuk menyelesaikan operasi k dari J_j lebih cepat, dilakukan perubahan urutan operasi pada mesin yang dilalui J_j. Jika *feasible* (*makespan*nya tidak lebih lama dari yang ditentukan) pakai urutan yang baru, dan jika tidak maka ditolak.

Step 6 : Set k ← k-1. Jika k=0, maka akan dilakukan *Step* 7, jika tidak maka kembali ke *Step* 5.

Prosedur III

Ada beberapa permasalahan pada *job shop* dimana beberapa *job* mempunyai prioritas lebih tinggi untuk jadwal yang optimal. Prosedur ini sama seperti prosedur I, kecuali untuk *step* 2, yaitu:

Step 2 : Set j ← index dari *job* pertama yang diselesaikan dalam S dan memindahkan J_j dari S.

Prosedur IV

Prosedur ini sama dengan prosedur I, kecuali untuk *step* 2, yaitu:

Step 2 : Set j ← index *job* terakhir yang dimulai dalam S dan memindahkan J_j dari S

Pendekatan Prosedur *Rescheduling Procedure*

Dari keempat prosedur diatas, digabungkan menjadi satu dengan mengadaptasi pendekatan *Rescheduling Procedure.* Melalui beberapa tahapan berikut:

Step 1 : Input parameter : L dan i_{max}. Dimana L adalah jumlah iterasi dan i_{max} adalah iterasimaksimum. Set $L_t \leftarrow 0$ dimana L_t adalah *iteration counter.*

Step 2 : Membuat jadwal awal dengan *dispatching rule* dan set $C \leftarrow C_{max}$.

Step 3 : Jika $Lt \leq L$, maka dilakukan *step* 4, jika tidak maka prosedur dihentikan.

Step 4 : Set $l \leftarrow 1$

Step 5 : Set $i \leftarrow 0$

Step 6 : Jika $i \leq n$ dan $i \leq i_{max}$, maka dilakukan *step* 7, jika tidak maka dilakukan *step* 9.

Step 7 : Jika l = 1 maka menggunakan prosedur I dan II
Jika l = 2 maka menggunakan prosedur II dan I
Jika l = 3 maka menggunakan prosedur III dan IV
Jika l = 4 maka menggunakan prosedur IV dan III

Step 8 : Jika C berubah, maka dilakukan *step* 7, jika tidak maka set $i \leftarrow i+1$ dan dilakukan *step* 6.

Step 9 : Set $l \leftarrow l+1$. Jika $l \leq 4$ maka dilakukan *step* 5, jika tidak $Lt \leftarrow Lt+1$ dan dilakukan *step* 3.