ALGORITMA PENJADWALAN PRODUKSI PADA LINGKUNGAN MESIN *JOB SHOP* DENGAN MINIMALISASI RATAAN WAKTU TUNGGU OPERASI

Gamma/13502058

Abstraksi. Pada makalah ini dijelaskan mengenai pengembangan algoritma untuk mencari solusi dari masalah penjadwalan *job shop*. Algoritma tersebut merupakan pengembangan dari algoritma yang sudah ada, yaitu algoritma *minimum slack*. Pengembangan algoritma ini dilakukan dengan menambahkan proses *forward checking* pada waktu menjadwalkan operasi. Solusi penjadwalan yang dihasilkan oleh algoritma yang telah dioptimisasi memiliki total waktu pengerjaan yang lebih kecil dibandingkan solusi yang dihasilkan oleh algoritma *minimum slack*. Akan tetapi, algoritma yang telah dioptimisasi memiliki kompleksitas waktu $O(N^2bm^2)$ yang lebih besar daripada kompleksitas algoritma *minimum slack*, yaitu $O(N^2b)$.

1. Masalah Penjadwalan *Job Shop*

Masalah penjadwalan *job shop* merupakan masalah penjadwalan yang memiliki karakteristik sebagai berikut [GAR00]:

1. Penjadwalan *job shop* memiliki sejumlah *job* yang harus diselesaikan, direpresentasikan sebagai $J = \{J_1,J_2,...,J_n\}$.
2. Penjadwalan *job shop* memiliki sejumlah *resource* yang digunakan untuk menyelesaikan setiap operasi, direpresentasikan sebagai $R = \{R_1,R_2,...,R_m\}$. Pada beberapa referensi, *resource* biasa disebut *machine* dilambangkan dengan *M* [AYD02].
3. Setiap *job* memiliki sejumlah operasi yang harus diselesaikan pada tenggat waktu mulai dari *ready time*(r_t) sampai *due time*(d_t). Suatu *job* J_i memiliki sejumlah operasi yang direpresentasikan dengan $O^i = \{O^i_1,O^i_2,...O^i_n\}$.
4. Setiap operasi memiliki waktu proses yang berbeda-beda. Himpunan operasi O^i memiliki sejumlah waktu proses yang direpresentasikan dengan $\{\tau^i_1,\tau^i_2,...\tau^i_n\}$.
5. Setiap operasi pada satu *job* memiliki *precedence*. Pada suatu *job* J_i , O^i_1 harus dikerjakan lebih dulu daripada O^i_2 , direpresentasikan dengan $(O^i_1, O^i_2) \in O_i$.

Eksekusi suatu operasi O^i_k membutuhkan satu atau lebih *resource* R^i_k yang merupakan himpunan bagian dari *R*. Eksekusi tersebut dilakukan selama interval τ^i_k . Waktu mulai (*start time*) dari operasi tersebut, st^i_k , merupakan awal digunakannya R^i_k . Pada selang waktu *resource* R^i_k digunakan oleh O^i_k, yaitu dari st^i_k sampai dengan $(st^i_k + \tau^i_k)$, operasi lain tidak dapat menggunakan *resource* tersebut. Secara matematis, suatu solusi masalah penjadwalan pada lingkungan mesin *job shop* harus memenuhi persamaan berikut [XU01]:

$$\forall v \in O^i : \quad st(v) \geq 0$$

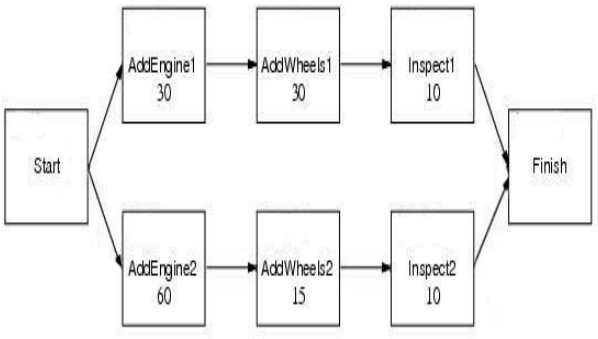
$$\forall v,w \in O^i, (v,w) \in O_i : \quad st(v) + \tau(v) \leq st(w)$$

$$\forall v,w \in O,\ v \neq w,\ R(v) = R(w) : \ st(v) + \tau(v) \leq st(w) \vee st(w) + \tau(w) \leq st(v)$$

2. Algoritma Penjadwalan Sederhana

Masalah penjadwalan memiliki berbagai variasi *constraint* yang harus dipenuhi dalam pencarian solusi. Berbagai algoritma penjadwalan sudah dikemukakan yang ditujukan untuk menyelesaikan masalah penjadwalan tertentu. Algoritma penjadwalan yang paling sederhana merupakan algoritma dasar yang menentukan kapan suatu operasi dikerjakan tanpa memperhatikan *resource* yang diperlukan untuk mengerjakan operasi tersebut.

Sebagai contoh, masalah penjadwalan sederhana direpresentasikan dalam graf seperti pada gambar 1. Pada gambar tersebut terdapat dua *job* perakitan mobil. Setiap *job* terdiri dari tiga operasi yaitu, *AddEngine*, *AddWheels*, dan *Inspect*. Setiap operasi tersebut memiliki durasi yang berbeda-beda. Durasi setiap operasi merupakan angka yang tercantum pada setiap simpul graf yang merepresentasikan operasi tersebut



Gambar 1

Anak panah pada graf tersebut melambangkan *precedence* setiap operasi. Operasi yang berada di kiri anak panah harus dikerjakan lebih dahulu dari operasi yang berada di kanan anak panah. Pada gambar 1 terdapat operasi *Start* dan *Finish* yang sebenarnya merupakan operasi semu (*dummy*) yang ditambahkan untuk menyatakan bahwa kedua *job* tersebut berada pada satu masalah penjadwalan. Durasi kedua operasi semu ini sama dengan nol.

Tujuan utama dari proses penjadwalan adalah menentukan waktu suatu operasi mulai dikerjakan. Hal ini dilakukan pada seluruh operasi sampai seluruh operasi sudah dijadwalkan dan memenuhi setiap batasan masalah yang dirumuskan.

Untuk menghasilkan solusi penjadwalan dengan total waktu pengerjaan yang minimum, seluruh operasi pada *critical path* harus dijadwalkan tanpa *delay* diantara operasi-operasi tersebut. Operasi yang bukan berada pada *critical path* boleh mengalami *delay* pada penjadwalannya. Walaupun boleh mengalami *delay*, setiap operasi harus dijadwalkan pada selang waktu di antara *earliest start time* (ES) dan *latest start time* (LS) dari operasi tersebut. Jika operasi tidak dijadwalkan pada selang waktu tersebut, maka solusi yang dihasilkan bukan merupakan solusi dengan total waktu pengerjaan yang minimum.

2.1. *Earliest Start Time* (ES) dan *Latest Start Time* (LS)

Earliest start time (ES) suatu operasi merupakan waktu paling awal dari operasi tersebut untuk mulai dikerjakan. Suatu operasi tidak dapat dikerjakan lebih awal dari ES-nya. *Latest start time* (LS) suatu operasi merupakan waktu paling akhir dari operasi tersebut untuk mulai dikerjakan. Jika suatu operasi mulai dikerjakan pada waktu yang lebih akhir dari LS-nya, maka seluruh proses akan mengalami perlambatan. Dengan kata lain, solusi yang dihasilkan tidak memiliki total waktu pengerjaan yang minimum.

Untuk menentukan ES setiap operasi digunakan algoritma sebagai berikut:

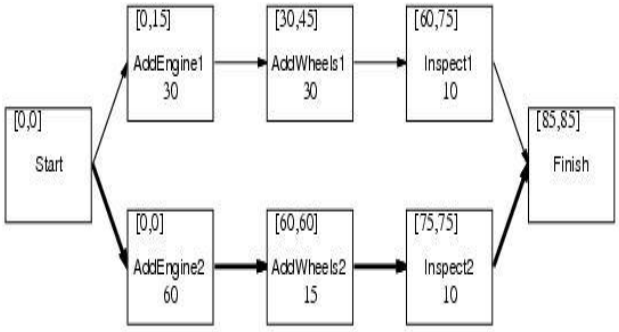
1. ES untuk operasi *Start* diinisialisasi dengan nol, ES(*Start*) = 0.
2. ES untuk operasi *B* ditentukan dengan nilai maksimum dari ES untuk operasi *A* dijumlahkan dengan durasi operasi *A*, dimana *A* merupakan *precedence* dari operasi *B*. Secara matematis dinotasikan dengan:
   ES(*B*) = *max*(ES(*A*) + *Duration*(*A*)) , *A* < *B*.

Untuk menentukan LS setiap operasi digunakan algoritma sebagai berikut:

1. LS untuk operasi *Finish* diinisialisasi sama dengan ES untuk operasi *Finish*, LS(*Finish*) = ES(*Finish*).
2. LS untuk operasi *A* ditentukan dengan nilai minimum dari LS untuk operasi *B* dimana *B* merupakan suksesor dari operasi *A*, dikurangi dengan durasi operasi *A*. Secara matematis dinotasikan dengan:
   LS(*A*) = *min*(*LS*(*B*)) - *Duration*(*A*) , A < B.

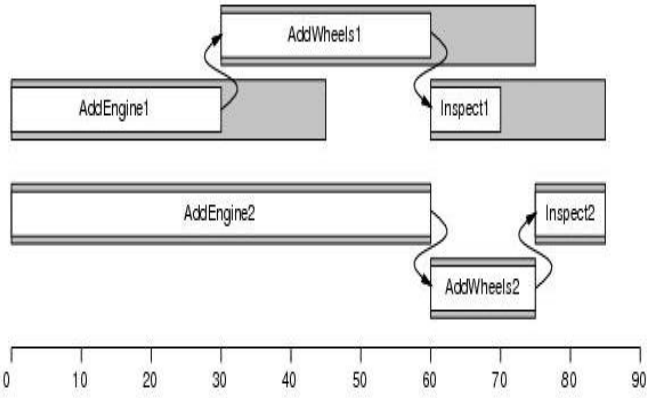
2.2. Penjadwalan Berdasarkan ES dan LS Setiap Operasi

Setelah ES dan LS seluruh operasi dihitung, maka representasi graf pada gambar 1 akan menjadi seperti pada gambar 2. Tampak bahwa setiap operasi diberi nilai ES dan LS pada sudut kiri atas dengan format [ES,LS].



Gambar 2

Solusi penjadwalan dari masalah penjadwalan pada gambar 2 dapat dilihat pada gambar 3. Terlihat bahwa setiap operasi dijadwalkan pada waktu ES dari operasi itu sendiri dan total waktu pengerjaan adalah 85 satuan waktu.



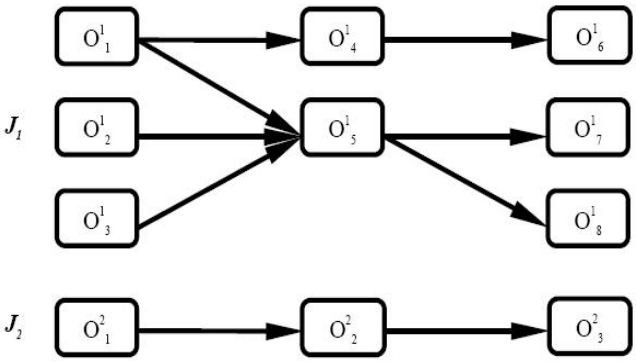
Gambar 3

Kompleksitas dari algoritma penjadwalan ini ditentukan dari jumlah perhitungan ES dan LS. Pada setiap operasi dilakukan perhitungan untuk ES dan LS. Setiap perhitungan diiterasi paling banyak sejumlah percabangan yang ada. Oleh karena itu, kompleksitas algoritma penjadwalan dengan perhitungan ES dan LS adalah O(*Nb*), dimana *N* adalah jumlah operasi dan *b* adalah maksimum percabangan dari satu operasi.

3. Konsep Penjadwalan *Job Shop*

Konsep penjadwalan *job shop* adalah menentukan waktu suatu operasi mulai dikerjakan dan mengalokasikan *resource* untuk mengerjakan operasi tersebut. Pada saat menjadwalkan suatu operasi selain menentukan kapan operasi tersebut mulai dikerjakan juga ditentukan *resource* mana yang dipakai oleh operasi tersebut. Oleh karena itu, pada saat menjadwalkan suatu operasi perlu diperhatikan dua *constraint* berikut:

1. *Precedence constraint*; penjadwalan untuk setiap operasi dari *job* yang sama harus berurutan sesuai dengan *precedence constraint job* tersebut. Sebagai contoh, masalah *job shop* yang ditunjukkan oleh gambar II-4 memiliki beberapa *precedence constraint*. Salah satunya adalah pada operasi O^1_1, O^1_2, O^1_3, dan O^1_5 yang berada pada *job* yang sama yaitu J_1. Oleh karena itu, operasi O^1_5 harus dijadwalkan setelah operasi O^1_1, O^1_2, dan O^1_3 selesai diproses. Namun, operasi O^1_5 dan O^2_2 berada pada *job* yang berbeda, maka urutan penjadwalan kedua operasi ini tidak menjadi masalah.



Gambar 4

2. *Resource constraint*; penjadwalan setiap operasi membutuhkan sebuah *resource* untuk mengerjakan operasi tersebut. Pada saat operasi ini mulai diproses *resource* tersebut harus sedang tidak dipakai operasi lain. *Resource* ini juga menjadi tidak dapat dipakai oleh operasi lain sampai operasi tersebut selesai.

Jika penjadwalan suatu operasi melanggar salah satu *constraint*, maka penjadwalan operasi tersebut harus dialihkan waktunya, dimana pengalihan waktu ini bisa membuat proses penyelesaian *job* dapat berlangsung lebih lama dan bahkan membuat seluruh proses mengalami keterlambatan. Dengan munculnya berbagai *constraint*, maka sesungguhnya algoritma penjadwalan *job shop*

merupakan bagian dari algoritma CSP (*Constraint Satisfaction Problem*).

3.1. Konflik

Suatu operasi yang akan dijadwalkan pada suatu waktu, penjadwalan tersebut harus memenuhi kedua *constraint* yang telah disebutkan diatas. Jika penjadwalan tersebut melanggar salah satu *constraint*, maka operasi tersebut harus dijadwalkan pada waktu yang lain. Ketika penjadwalan melanggar salah satu *constraint*, hal ini dikatakan penjadwalan tersebut mengalami konflik. Oleh karena itu, operasi itu akan dijadwalkan pada waktu yang lain. Jika penjadwalan pada waktu tersebut tetap menghasilkan konflik, maka harus dipilih kembali satu waktu lain yang tidak menimbulkan konflik, begitu seterusnya. Proses penjadwalan yang berulang-ulang dan terus menemui konflik ini membuat efisiensi dari algoritma penjadwalan itu semakin buruk.

Adapun dua sasaran utama pada algoritma penjadwalan *job shop* adalah:

1. *Feasibility*; menemukan solusi yang memenuhi setiap *constraint*. Jika sudah ditemukan solusi, dapat dilakukan optimisasi berdasarkan kriteria tertentu. Pada tugas akhir ini, optimisasi dilakukan dengan meminimalisir rataan waktu tunggu operasi untuk memperoleh total waktu pengerjaan (*completion time*) sekecil mungkin.
2. *Efficiency*; meminimalisasi jumlah terjadinya konflik pada setiap penjadwalan operasi. Untuk dapat mencegah terjadinya konflik dalam proses penjadwalan operasi dapat diantisipasi dengan mendeteksi kapan *resource* untuk mengerjakan operasi tersebut sedang tidak dipakai.

3.2. Teknik Dalam Penjadwalan

Pada setiap penjadwalan suatu operasi dapat dilakukan dua buah teknik, yaitu *retrospective* dan *prospective* [BEC94].

3.2.1. Teknik *Retrospective*

Retrospective berarti melakukan penjadwalan operasi pada suatu waktu dengan memeriksa operasi lainnya yang sudah dijadwalkan untuk menghindari konflik. Jika semua *constraint* tetap terjaga, maka penjadwalan operasi tersebut valid. Jika ada *constraint* yang tidak terpenuhi, maka harus dipilih waktu lain untuk operasi tersebut.

Sebagai contoh penerapan teknik ini, pada gambar 4 operasi O^1_5 harus dijadwalkan setelah operasi O^1_1, O^1_2, dan O^1_3 selesai diproses. Oleh karena itu, penjadwalan operasi O^1_5 memeriksa kapan operasi O^1_1, O^1_2, dan O^1_3 selesai diproses. Jika operasi O^1_1, O^1_2, dan O^1_3 selesai diproses pada waktu ke-n, maka operasi O^1_5 harus dijadwalkan pada waktu yang lebih besar atau sama dengan waktu ke-n. Jika operasi O^1_5 dijadwalkan pada waktu lebih awal dari waktu ke-n, maka sudah dipastikan akan terjadi konflik karena melanggar *precedence constraint*.

Dengan demikian, pemilihan waktu penjadwalan suatu operasi dilakukan dengan melihat operasi-operasi yang sudah dijadwalkan sebelumnya. Teknik *retrospective* bisa dikatakan sebagai teknik "melihat ke belakang" karena untuk menghindari konflik pada suatu proses penjadwalan, dilakukan pengamatan kembali terhadap operasi lain yang sudah dijadwalkan.

3.2.2. Teknik *Prospective*

Prospective berarti menyebarkan (*propagation*) akibat yang ditimbulkan dari penjadwalan suatu operasi kepada operasi lain yang belum dijadwalkan. Dengan memeriksa akibat yang ditimbulkan dari penjadwalan suatu operasi kepada operasi lain yang belum dijadwalkan, dapat meminimalisir kemungkinan terjadinya keterlambatan proses pengerjaan.

Sebagai contoh penerapan dari teknik ini, pada gambar 4 terdapat operasi O^1_5 dan O^2_2 yang berada pada *job* yang berbeda. Anggap kedua operasi ini menggunakan mesin yang sama yaitu M_i, dan hanya terdapat satu buah mesin M_i masalah penjadwalan ini. Oleh karena kedua operasi ini menggunakan mesin yang sama, maka perlu ditentukan operasi

mana yang harus lebih dulu dijadwalkan dan operasi mana yang harus dikerjakan sesudahnya.

Urutan pengerjaan kedua operasi tersebut sebenarnya tidak menjadi masalah karena kedua operasi ini tidak terikat oleh *precedence constraint*. Namun, dengan memperhatikan bahwa O^1_5 memiliki dua operasi suksesor yaitu O^1_7 dan O^1_8, sedangkan O^2_2 hanya memiliki satu suksesor yaitu O^2_3, maka O^1_5 lebih dulu dijadwalkan daripada O^2_2. Dengan memperlambat operasi O^2_2, diperkirakan perlambatan yang terjadi akan lebih kecil dibandingkan jika operasi O^1_5 yang diperlambat. Hal ini karena operasi O^1_5 memiliki dua buah operasi suksesor sedangkan O^2_2 hanya memiliki satu buah operasi suksesor.

Dengan demikian, pemilihan waktu penjadwalan suatu operasi dilakukan dengan memperkirakan akibat yang ditimbulkan dari penjadwalan suatu operasi kepada operasi lain yang belum dijadwalkan. Teknik *prospective* bisa dikatakan sebagai teknik "melihat ke depan" atau *forward checking* karena untuk menghindari konflik pada suatu proses penjadwalan, dilakukan pemeriksaan akibat yang mempengaruhi operasi lain yang belum dijadwalkan. Teknik ini hanyalah melakukan perkiraan, sehingga belum tentu dapat menghindari konflik yang akan datang, hanya memperkecil kemungkinan saja.

3.3. Pengaruh Ketersediaan Mesin Terhadap Perhitungan ES dan LS Suatu Operasi

Pada masalah penjadwalan sederhana yang diuraikan sebelumnya, untuk menghitung ES dan LS suatu operasi digunakan algoritma seperti yang sudah diuraikan pada bagian 0. Namun, pada masalah penjadwalan *job shop*, untuk memperoleh nilai ES suatu operasi, tidak bisa hanya dengan menggunakan algoritma tersebut. Hal ini dikarenakan masalah *job shop* sudah melibatkan mesin-mesin yang dibutuhkan untuk mengerjakan operasi tersebut. Oleh karena itu, algoritma untuk memperoleh nilai ES suatu operasi menjadi:

1. ES untuk operasi *Start* diinisialisasi dengan nol, ES(*Start*) = 0.
2. Nilai ES awal untuk operasi *B* ditentukan dengan nilai maksimum dari ES untuk operasi *A* dijumlahkan dengan durasi operasi *A*, dimana *A* merupakan *precedence* dari operasi *B*. Secara matematis dinotasikan dengan:

$$ES_{awal}(B) = max(ES(A) + Duration(A)) \text{ , } A < B.$$

3. Nilai ES akhir diperoleh dengan memeriksa slot waktu mesin yang digunakan oleh operasi *B*. Jika dari waktu $ES_{awal}(B)$ sampai dengan $(ES_{awal}(B) + \tau(B))$ mesin tersebut bebas, maka nilai ES sama dengan $ES_{awal}(B)$. Jika tidak, akan dicari waktu terkecil yang lebih besar dari $ES_{awal}(B)$ dimana operasi *B* dapat dialokasikan pada waktu tersebut.

Dengan adanya *resource constraint*, nilai ES suatu operasi bisa berubah dari hasil perhitungan yang dilakukan berdasarkan algoritma. Hal ini dikarenakan pada waktu ES yang seharusnya, mesin untuk mengerjakan operasi tersebut sedang dipakai. Oleh karena itu, perlu dicari waktu ES yang lain dimana mesin tersebut bebas.

Perubahan nilai ES suatu operasi akan mempengaruhi nilai ES dan LS operasi lainnya. Hal ini dikarenakan perhitungan ES dan LS seluruh operasi dilakukan dengan iterasi setiap operasi. Oleh karena itu, pada setiap aktivitas yang melibatkan reservasi mesin, ES dan LS seluruh operasi perlu dihitung ulang. Aktivitas yang melibatkan reservasi mesin salah satunya adalah penjadwalan suatu operasi. Oleh karena itu, pada setiap algoritma penjadwalan *job shop*, setelah dilakukan penjadwalan suatu operasi selalu dilakukan perhitungan ulang ES dan LS seluruh operasi.

4. Algoritma *minimum slack*

Algoritma *minimum slack* merupakan algoritma penjadwalan *job shop* yang paling populer. Seperti namanya, algoritma ini menggunakan *slack time* suatu operasi sebagai *heuristic*. Algoritma ini mampu menyelesaikan berbagai masalah penjadwalan *job shop*, walaupun tidak selalu menghasilkan solusi dengan total waktu pengerjaan yang minimum.

4.1. *Slack Time*

Slack time dari suatu operasi merupakan selisih antara *latest start time* (LS) dengan *earliest start time* (ES) dari operasi tersebut. Secara matematis dapat dituliskan seperti pada persamaan berikut:

$$S(O^i_j) = LS(O^i_j) - ES(O^i_j)$$

Setiap operasi harus dijadwalkan pada selang waktu di antara ES dan LS dari operasi tersebut. Semakin kecil nilai *slack time* suatu operasi, semakin kecil rentang waktu antara ES dan LS operasi tersebut, sehingga semakin sedikit waktu yang dapat dialokasikan untuk penjadwalan operasi tersebut.

Pada algoritma *minimum slack*, *slack time* suatu operasi digunakan sebagai *heuristic* untuk menentukan urutan penjadwalan operasi. Semakin kecil nilai *slack time* suatu operasi, maka semakin tinggi prioritas operasi tersebut untuk dijadwalkan terlebih dahulu. Setelah operasi itu dijadwalkan, dipilih operasi lain dengan nilai untuk dijadwalkan. Operasi ini dipilih diantara operasi-operasi yang belum dijadwalkan dengan nilai *slack time* terkecil.

4.2. Skema Algoritma *Minimum Slack*

Langkah pertama yang dilakukan oleh algoritma *minimum slack* adalah dengan melakukan proses perhitungan ES dan LS untuk setiap operasi. ES dan LS ini akan digunakan untuk menentukan *slack time* setiap operasi.

Langkah berikutnya adalah melakukan penjadwalan setiap operasi, sehingga dihasilkan solusi untuk masalah *job shop* tersebut. Untuk memilih operasi mana yang harus dijadwalkan terlebih dahulu, ditentukan kriteria untuk operasi tersebut, yaitu:

1. Seluruh ***precedence*** dari operasi tersebut harus sudah dijadwalkan. Hal ini untuk memenuhi *precedence constraint*.
2. Berdasarkan *heuristic* yang digunakan, yaitu memiliki ***slack time*** terkecil.

Secara sistematis, langkah-langkah dari algoritma *minimum slack* ini adalah:

1. Hitung ES dan LS untuk setiap operasi.
2. Ambil operasi-operasi yang sudah dijadwalkan *precedence*-nya. Hal ini dilakukan untuk menjaga *precedence constraint*.
3. Pilih satu diantara sejumlah operasi tersebut yang memiliki *slack time* terkecil, lalu operasi ini dijadwalkan.
4. Hitung ulang ES dan LS pada setiap operasi yang terpengaruh oleh penjadwalan sebelumnya.
5. Kembali ke langkah 2 sampai seluruh operasi telah terjadwalkan.

Prinsip yang digunakan dalam algoritma ini adalah penjadwalan diutamakan pada operasi yang memiliki *constraint* paling banyak. Dengan menganggap operasi dengan *slack time* terkecil merupakan operasi yang memiliki *constraint* terbesar, maka algoritma ini akan selalu mendahulukan operasi yang demikian. Algoritma *minimum slack* ini menggunakan prinsip *greedy*, karena itu banyak digunakan karena cukup tangguh dalam menyelesaikan berbagai masalah penjadwalan. Akan tetapi, algoritma ini tidak selalu menghasilkan solusi dengan total waktu pengerjaan (*completion time*) yang minimum pada beberapa kasus.

4.3. Kompleksitas Algoritma *Minimum Slack*

Pada algoritma *minimum slack* pada setiap penjadwalan operasi dilakukan perhitungan ulang ES dan LS setiap operasi. Pada bagian 0, telah dijelaskan bahwa kompleksitas algoritma dari perhitungan ES dan LS setiap operasi adalah $O(Nb)$, dimana N adalah jumlah operasi dan b adalah maksimum percabangan dari satu operasi.

Algoritma *minimum slack* melakukan perhitungan ES dan LS sebanyak jumlah operasi yang dijadwalkan. Oleh karena itu, kompleksitas algoritma *minimum slack* adalah:

$$O(Nb).N = O(Nb).O(N) = O(Nb.N) = O(N^2 b)$$

dengan N adalah jumlah operasi dan b adalah maksimum percabangan dari satu operasi.

5. Optimisasi Algoritma Dengan Minimalisasi Rataan Waktu Tunggu Operasi

Masalah utama yang dikaji dalam tugas akhir ini adalah bagaimana mengoptimisasi algoritma *minimum slack* untuk penjadwalan *job shop*, sehingga mampu menghasilkan solusi yang lebih optimal. Tolok ukur yang digunakan pada tugas akhir ini dalam menentukan optimasi suatu solusi adalah total waktu pengerjaan (*completion time*) dari solusi tersebut. Untuk itu diterapkan metode minimalisasi rataan waktu tunggu operasi agar dapat memberikan solusi yang lebih optimal dibanding algoritma *minimum slack*. Pada tugas akhir ini dikembangkan algoritma penjadwalan *job shop* dengan minimalisasi rataan waktu tunggu operasi.

Algoritma ini tetap menggunakan prinsip *heuristic* pada dasarnya yaitu, menjadwalkan operasi yang paling banyak memiliki *constraint* pada suatu waktu sedemikian sehingga penjadwalan tersebut sedikit menimbulkan konflik [GAR00]. Pemberian prioritas utama pada operasi yang memiliki *slack time* terkecil untuk dijadwalkan terlebih dahulu dirasa kurang tepat. Ini dikarenakan operasi tersebut bisa saja memiliki durasi yang cukup lama, sehingga membuat operasi lain harus menunggu lebih lama untuk bisa dikerjakan.

Tujuan utama dari mendahulukan penjadwalan operasi dengan *slack time* terkecil adalah supaya memperoleh solusi dengan *completion time* minimum. Akan tetapi, dengan bertambahnya waktu tunggu operasi lain akibat penjadwalan operasi tersebut, kemungkinan untuk menghasilkan solusi dengan *completion time* minimum semakin kecil. Oleh karena itu, pada algoritma ini operasi yang terlebih dahulu dijadwalkan adalah operasi yang paling sedikit memperbesar waktu tunggu operasi lainnya, bukan operasi dengan nilai *slack time* terkecil.

Operasi yang paling sedikit memperbesar waktu tunggu operasi lainnya, diutamakan proses penjadwalannya. Hal ini dilakukan karena semakin kecil waktu tunggu suatu operasi, maka semakin cepat operasi itu dapat dikerjakan. Jika seluruh operasi memiliki waktu tunggu sekecil mungkin, maka semakin cepat seluruh operasi dapat dikerjakan. Oleh karena itu, semakin kecil pula total waktu pengerjaan (*completion time*) seluruh *job*.

5.1. Uji Penjadwalan

Untuk mengetahui apakah penjadwalan suatu operasi akan memperlambat *completion time*, dapat diterapkan teknik *prospective* atau *forward checking*. *Forward checking* dilakukan dengan memeriksa apakah penjadwalan suatu operasi akan memperlambat LS operasi *Finish* karena LS operasi *Finish* merupakan minimum *completion time* yang dapat dicapai seluruh proses.

Pemeriksaan pengaruh penjadwalan suatu operasi terhadap LS operasi *Finish* untuk selanjutnya disebut sebagai *uji penjadwalan*. Pada saat melakukan uji penjadwalan suatu operasi, operasi ini sebenarnya tidak dijadwalkan, hanya saja diperiksa seberapa besar penjadwalan operasi tersebut akan memperlambat LS operasi *Finish*.

Uji penjadwalan dilakukan pada sekelompok operasi yang menggunakan mesin yang sama dan tiap-tiap *precedence*-nya sudah dijadwalkan sebelumnya. Sekelompok operasi ini sudah dapat dijadwalkan karena tiap-tiap *precedence*-nya sudah dijadwalkan. Namun, sekelompok operasi ini menggunakan mesin yang sama, sehingga penjadwalan satu operasi berpeluang besar untuk menimbulkan konflik pada penjadwalan operasi lainnya.

Untuk mengetahui perlambatan LS operasi *Finish* akibat dari penjadwalan operasi *O*, dapat digunakan algoritma uji penjadwalan. Algoritma ini mengembalikan nilai LS operasi *Finish* akibat dari penjadwalan operasi *O*. Langkah-langkah dari algoritma tersebut adalah:

1. Ambil seluruh operasi yang *precedence*-nya sudah dijadwalkan dan menggunakan mesin yang sama dengan operasi *O*. Masukkan sekelompok operasi ini ke dalam himpunan A.
2. Lakukan penjadwalan semu operasi *O*. Pada saat ini sebenarnya operasi *O* belum dijadwalkan. Pada penjadwalan semu ini, dilakukan reservasi mesin untuk

mengerjakan operasi *O*, tetapi operasi *O* itu sendiri belum dijadwalkan.

3. Hitung ulang ES dan LS setiap operasi. ES dan LS setiap operasi perlu dihitung ulang karena mengalami perubahan akibat reservasi mesin yang dilakukan sebelumnya.
4. Untuk setiap operasi pada himpunan A lakukan penjadwalan semu. Lakukan reservasi mesin dan hitung ulang ES dan LS setiap operasi pada setiap penjadwalan semu yang dilakukan.
5. Simpan LS operasi *Finish* pada suatu variabel setelah dihitung ulang.
6. Hapus reservasi mesin yang sebelumnya telah dilakukan akibat seluruh penjadwalan semu.
7. Hitung ulang ES dan LS setiap operasi. ES dan LS setiap operasi perlu dihitung ulang karena mengalami perubahan akibat reservasi mesin yang sebelumnya dibatalkan.
8. Nilai LS operasi *Finish* yang telah disimpan pada variabel dijadikan *return value*.

5.2. Kompleksitas Algoritma Uji Penjadwalan

Pada setiap uji penjadwalan dilakukan perhitungan ulang ES dan LS setiap operasi sebanyak dua kali. Pada upa bab 0, telah dijelaskan bahwa kompleksitas algoritma dari perhitungan ES dan LS setiap operasi adalah O(*Nb*), dimana *N* adalah jumlah operasi dan *b* adalah maksimum percabangan dari satu operasi.

Algoritma uji penjadwalan melakukan perhitungan ES dan LS sebanyak jumlah operasi yang menggunakan mesin yang sama. Jika pada suatu masalah penjadwalan jumlah operasi yang menggunakan mesin yang sama adalah *m*, maka kompleksitas algoritma uji penjadwalan ini adalah:

$$O(Nb).m = O(Nb).O(m) = O(Nb.m) = O(Nbm)$$

dengan *N* adalah jumlah operasi, *b* adalah maksimum percabangan dari satu operasi, dan *m* adalah jumlah operasi yang menggunakan mesin yang sama.

5.3. Skema Algoritma Penjadwalan *Job Shop* Dengan Minimalisasi Rataan Waktu Tunggu Operasi

Sama seperti pada algoritma algoritma *minimum slack* untuk penjadwalan *job shop*, pertama kali dilakukan proses perhitungan ES dan LS untuk setiap operasi. Walaupun tidak menggunakan *slack time*, ES dan LS ini dibutuhkan dalam melakukan *forward checking* dan menghitung LS operasi *Finish* pada suatu waktu.

Operasi yang akan dijadwalkan lebih dahulu harus memiliki *precedence* yang sudah dijadwalkan sebelumnya. Hal ini sama dengan algoritma *minimum slack* untuk menjaga *precedence constraint*. Perbedaan yang mencolok terdapat pada pemilihan operasi yang akan dijadwalkan dengan memperhatikan pengaruh dari penjadwalan operasi tersebut.

Secara sistematis, langkah-langkah dari algoritma ini adalah:

1. Hitung ES dan LS untuk setiap operasi.
2. Memilih operasi-operasi mana saja yang sudah dijadwalkan *precedence*-nya.
3. Dari sejumlah operasi yang sudah dijadwalkan *precedence*-nya, akan dipilih (irisan) sejumlah operasi yang memiliki *resource constraint* paling banyak. Ini berarti sejumlah operasi yang menggunakan mesin yang sama akan dijadwalkan terlebih dahulu. Sejumlah operasi ini dimasukkan ke dalam himpunan A. Jika operasi yang demikian tidak ada, atau dengan kata lain seluruh operasi menggunakan mesin yang berbeda, maka operasi yang memiliki LS paling kecil akan dijadwalkan lebih dahulu. Hal ini dilakukan dengan anggapan bahwa operasi dengan nilai LS kecil memiliki kemungkinan kecil untuk memperlambat operasi-operasi lain yang akan dijadwalkan berikutnya.
4. Jika pada himpunan A terdapat dari lebih dari satu operasi, maka lakukan *forward checking* pada tiap-tiap operasi yang berada di himpunan A dengan cara melakukan uji penjadwalan untuk setiap operasi kepada ES-nya. Pada setiap uji penjadwalan, perhatikan pengaruhnya pada perlambatan

LS operasi *Finish*. Operasi yang pada uji penjadwalannya paling sedikit memperlambat LS operasi *Finish*, dipilih untuk dijadwalkan seperti pada pengujian yang telah dilakukan. Jika pada himpunan A hanya terdapat satu operasi, maka operasi tersebut dapat langsung dijadwalkan tanpa dilakukan uji penjadwalan.

5. Hitung ulang ES dan LS pada setiap operasi yang terpengaruh oleh penjadwalan sebelumnya. Keluarkan operasi yang telah dijadwalkan dari himpunan A.
6. Kembali kepada langkah 4 sampai seluruh operasi pada himpunan A sudah dijadwalkan, atau A = $\varnothing$.
7. Kembali kepada langkah 2 sampai seluruh operasi telah terjadwalkan.

5.4. Kompleksitas Algoritma Yang Telah Dioptimisasi

Seperti pada algoritma *minimum slack*, algoritma ini melakukan perhitungan ulang ES dan LS pada setiap penjadwalan operasi. Namun, perhitungan ulang ES dan LS juga dilakukan pada setiap uji penjadwalan. Algoritma ini melakukan uji penjadwalan pada sejumlah operasi yang *precedence*-nya sudah dijadwalkan dan menggunakan mesin yang sama.

Jika pada suatu masalah penjadwalan terdapat *m* buah operasi yang menggunakan mesin yang sama, maka kompleksitas dari uji penjadwalan yang dilakukan sebanyak jumlah operasi ini adalah:

$$O(Nbm).m = O(Nbm).O(m) = O(Nbm^2)$$

dengan *N* adalah jumlah operasi, *b* adalah maksimum percabangan dari satu operasi, dan *m* adalah jumlah operasi yang menggunakan mesin yang sama.

Jumlah perhitungan ES dan LS pada algortima ini sama dengan jumlah perhitungan ES dan LS pada algoritma *minimum slack* ditambah dengan yang dilakukan pada uji penjadwalan. Oleh karena itu, kompleksitas algoritma ini adalah penjumlahan dari kedua kompleksitas algoritma tersebut:

$$O(N^2b) + O(Nbm^2) = O(N^2bm^2)$$

dengan *N* adalah jumlah operasi, *b* adalah maksimum percabangan dari satu operasi, dan *m* adalah jumlah operasi yang menggunakan mesin yang sama.

6. Daftar Pustaka

[RUS95] Russel, Stuart dan Norvig, Peter. (1995). *Artificial Intelligence A Modern Approach*. Prentice Hall.

[GAR00] Garrido,A. (2000). *Heuristic Methods for Solving Job-Shop Scheduling Problems*.

[HOC99] Hochbaum, Dorit S. (1999). *The Scheduling Problem*. http://riot.ieor.berkeley.edu/riot/Applications/Scheduling/index.html Tanggal 24 November 2005.

[FOX90] Fox, M.S. (1990). *Why is Scheduling difficult? A CSP Perspective*.

[AYD02] Aydin, M. Emin. (2002). *Modular Simulated Annealing Algorithm for Job Shop Scheduling running on Distributed Resource Machine (DRM)*.

[XU01] Xu, Qianjun. (2001). *Introduction to Job Shop Scheduling Problem*.

[BEC94] Beck, J.C. (1994). *A Schema for Constraint Relaxation with Instantiations for Partial Constraint Satisfaction and Schedule Optimization,* Master's Thesis, University of Toronto.