Algoritma Tabu Search dan Penggunaannya dalam Penyelesaian Job Shop Scheduling Problem

M. Noversada[1], Fitri Meiriza[2], Dyah Ayuni Wijayanti[3]

Laboratorium Ilmu dan Rekayasa Komputasi
Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if13029@students.if.itb.ac.id[1], if14001@students.if.itb.ac.id[2], if14028@students.if.itb.ac.id[3]

Abstrak

Tabu search kini dikenal sebagai salah satu teknik untuk optimalisasi yang cukup efektif setelah dilakukan beberapa percobaan komputasional[1]. Algoritma tabu search mampu bersaing dengan teknik-teknik terkenal lainnya disebabkan fleksibilitasnya yang bisa menyaingi prosedur klasik lainnya. Karena sudah banyak terbukti bahwa tabu search adalah algoritma yang cukup efektif maka saat ini banyak sekali permasalahan dari berbagai bidang yang bisa menggunakan implementasi algoritma tabu search untuk mendapatkan solusi optimal. Salah satu contoh permasalahan yang juga dapat diselesaikan dengan algoritma tabu search adalah permasalahan job shop scheduling dimana permasalahan ini bertujuan untuk meminimasi waktu untuk menyelesaikan sederetan operasi dari job yang ada. Makalah ini akan membahas sedikit lebih mendalam mengenai algoritma tabu search dan disertai dengan contoh implementasinya dalam menyelesaikan permasalahan job shop scheduling.

***Kata kunci**:, algoritma tabu search, job-shop problem,scheduling*

1. Pendahuluan

Pada dasarnya, *job-shop problem* adalah salah satu contoh dari permasalah penjadwalan yang paling dasar (*basic scheduling problems*) di samping beberapa contoh lain seperti *open shop problem* atau *flow job problem*[2].

Dalam permasalahan *job-shop*, terdapat beberapa pekerjaan yang memiliki operasinya sendiri-sendiri yang hanya dapat dijalankan di sebuah mesin pada suatu waktu tertentu. Objektif dari permasalahan *job-shop* ini adalah untuk mendapatkan sebuah jadwal operasi pekerjaan yang memakan waktu yang paling singkat yang mungkin dilakukan. Banyak pendekatan yang dapat dilakukan untuk menyelesaikan masalah ini, dan salah satu caranya adalah menggunakan pendekatan algoritma *tabu search*. *Tabu search* ini sudah mendapatkan pengakuan luas sebagai salah satu pendekatan yang paling efektif dalam menghasilkan solusi yang berkualitas tinggi untuk masalah *job-shop*[4]. Namun ternyata, masih sedikit pihak yang mengetahui mengapa *tabu search* ini sangat efektif untuk penyelesaian masalah *job-shop*, dan dalam kondisi yang bagaimana sehingga kita bisa mendapatkan hasil yang paling optimal dari *tabu search* ini.

Makalah ini akan membahas mengenai kerangka dasar algoritma *Tabu Search* beserta sebuah contoh implementasi penggunaan algoritma *Tabu Search* dalam menyelesaikan permasalahan *job shop scheduling*.

2. Permasalahan Job Shop Scheduling

Permasalahan *job shop scheduling problem* (JSP) dapat dibentuk sebagai sebuah himpunan *J*, dimana *J* terdiri dari *n jobs* (pekerjaan) dan sebuah himpunan *M* yang terdiri dari *m* mesin. Setiap pekerjaan J_i memiliki n_i *subtasks* (disebut operasi), dan setiap operasi J_{ij} harus dijadwalkan pada mesin yang sudah ditetapkan sebelumnya, $\mu_{ij} \in M$ untuk lamanya waktu yang tetap, d_{ij}, tanpa interupsi. Tidak ada mesin yang boleh memproses lebih dari satu operasi pada sebuah waktu tertentu, dan setiap operasi $J_{ij} \in J_i$ harus selesai sebelum operasi selanjutnya pada *job* tersebut $(J_{i(j+1)})$ dimulai. Suksesor dari operasi x pada *job*nya dituliskan sebagai $SJ_{[x]}$, dan suksesor dari x pada mesinnya dituliskan sebagai $SM_{[x]}$. Demikian juga dengan predesesor yang dituliskan sebagai dan $PJ_{[x]}$ dan $PM_{[x]}$. Setiap operasi *x* memiliki waktu mulai yang dinotasikan sebagai r_2, dan waktu *tail* yang dinotasikan dengan t_2, dimana itu merupakan jalur terpanjang dari waktu *x* untuk bisa selesai sampai ke akhir.

Untuk menyelesaikan masalah ini, untuk setiap mesin, kita harus menemukan sebuah urutan dari operasi-operasi yang akan dijadwalkan

yang bisa mengoptimalkan fungsi objektif. Ada beberapa fungsi objektif yang sering digunakan pada permasalahan ini. Sejauh ini, fungsi objektif yang paling umum adalah meminimasi waktu total untuk menyelesaikan semua pekerjaan (*tasks*). Objektif ini penggunaannya cukup luas karena fungsi tersebut cukup mewakili banyak persoalan di dunia industri, dan sangat mudah untuk dikomputasi secara efisien.

3. Algoritma Tabu Search

Metode pencarian tabu berprinsip pada penggunaan memori sebagai elemen esensial dalam pencariannya, karena pencarian Tabu tidak hanya menyimpan nilai sebuah solusi terbaik seperti kebanyakan metode pencarian, namun juga menyimpan informasi selama pencarian melalui solusi terakhir yang dikunjungi. Sebuah informasi akan digunakan sebagai petunjuk untuk bergerak dari *i* ke solusi selanjutnya dalam *N(i)*. Penggunaan memori sebagai pembatas dalam pemilihan beberapa subset dari *N(i)* dengan mencegah pergerakan ke beberapa solusi tetangga.

Lebih tepatnya, kita akan menyadari bahwa struktur *N(i)* sebagai solusi *i* akan berupa variabel dari satu iterasi ke iterasi lainnya. Oleh karena itu metode pencarian tabu dapat disebut sebagai kelas dari prosedur-prosedur yang disebut *dynamic neighborhood search techniques*.

Secara formal,kita dapat menganggap masalah optimalisasi dalam cara berikut: Diberikan sebuah himpunan solusi *S* dan sebuah fungsi *f : S*, temukan solusi *i** dalam *S* sehingga *f(i*)* dapat diterima dengan beberapa kriteria. Secara umum kriteria untuk dapat diterima sebagai solusi *i** harus memenuhi *f(i*)=f(i)* untuk setiap *i* dalam *S*. Dalam situasi metode pencarian tabu akan menjadi sebuah algoritma minimimasi secara pasti yang menyediakan proses eksplorasi yang menjamin setelah sejumlah langkah-langkah terhingga *i** dapat dicapai.

Walaupun tidak ada jaminan bahwa *i** akan diperoleh, metode pencarian tabu dapat secara sederhana dipandang sebagai prosedur *heuristic* umum secara ekstrem. Karena metode pencarian tabu akan mencakup beberapa teknik *heuristic* dalam aturan-aturan operasi di dalamnya, maka akan lebih pantas untuk menggolongkan motode pencarian tabu sebagai *metaheuristic*. Perannya akan sering dijadikan sebagai petunjuk dan sebagai orientasi prosedur pencarian lainnya yang lebih lokal.

Untuk mendalami lagi prinsip kerja metode pencarian tabu, kita dapat memformulakan metoda menurun klasik (*classical descent method*) dalam beberapa langkah, yaitu:

1. Memilih sebuah solusi awal *i* dalam *S*
2. Membangkitkan subset *V** sebagai solusi dalam *N(i)*
3. Mencari *j* 'terbaik' dalam *V** dan menetapkan *i=j*
4. Jika *f(j)=f(i)* maka berhenti, namun jika tidak kembali ke langkah ke-2

Dalam metode menurun secara umum akan dapat langsung ditetapkan bahwa *V* = N(i)*. Tetapi hal ini seringkali membutuhkan waktu yang lama. Untuk itulah cara pemilihan *V** yang tepat seringkali dijadikan sebagai peningkatan yang penting dalam metode pencarian.

Pada kasus yang berlawanan, akan ditetapkan *|V*|=1*. Hal ini akan menurunkan fase pemilihan j 'terbaik'. Solusi *j* akan diterima jika *f(j)=f(i)*, sebaliknya hal ini akan diterima dengan kemungkinan tertentu yang bergantung pada nilai-nilai *f* pada *i* dan *j* serta pada sebuah parameter yang disebut temperatur.

Tidak ada temperatur dalam metode pencarian tabu, namun pemilihan *V** akan menjadi hal yang penting guna mendefinisikannya dalam setiap langkah di mana akan terjadi penggunaan memori secara sistematis untuk memanfaatkan informasi yang ada di luar fungsi *f* dan lingkungan *N(i)*.

Sebagai pengecualian pada kasus-kasus istimewa, penggunaan prosedur menurun (*descent procedure*) secara umum lebih rumit karena kita akan terperangkap pada sebuah minimum lokal yang mungkin masih jauh dari minimum global.

Maka untuk proses iterasi dalam eksplorasi apapun sebaiknya dalam beberapa hal juga menerima langkah-langkah yang tidak akan memberikan perkembangan dari *i* ke *j* dalam *V** (misal *f(j)>f(i)*). Metode pencarian tabu secara berbeda memilih *j* 'terbaik' dalam *V**.

Selama pergerakan yang tidak memberi perkembangan itu mungkin, resiko pengunjungan kembali sebuah solusi atau lebih umumnya disebut sebagai siklus mungkin untuk terjadi. Dalam hal inilah penggunaan memori sangat diperlukan untuk mencegah terjadinya pergerakan ke solusi yang telah dikunjungi. Jika memori seperti itu diperkenalkan, maka kita dapat menganggap struktur *N(i)* tergantung pada pengelilingan yang merupakan pengulangan *k*. Jadi kita dapat merujuk ke *N(i,k)* daripada *N(i)*.

Dengan perujukan ini kita dapat mencoba untuk melakukan peningkatan algoritma menurun dalam beberapa hal untuk lebih mendekatkan metode ini ke prosedur dalam metode pencarian tabu secara umum. Hal ini dapat didefinisikan dalam beberapa langkah (catatan *i** adalah solusi 'terbaik' yang ditemukan dan k adalah penghitung dalam pengulangan) :

1. Memilih solusi awal *I* dalam *S*. Tetapkan *i*=I* dan *k=0*.
2. Tetapkan nilai *k=k+1* dan membagkitkan subset *V** sebagai solusi dalam *N(i,k)*
3. Pilih *j* 'terbaik' dalam *V** dan tetapkan *i=j*.
4. Jika *f(i) < f(i*)* maka tetapkan *i*=i*.
5. Jika kondisi berhenti ditemukan, maka proses dihentikan, sedangkan jika belum kembali ke langkah 2.

Amati bahwa langkah-langkah pada metode penurunan klasik termasuk dalam formula ini (kondisi berhenti secara sederhana jika *f(i)=f(i*)* dan *i** akan selalu menjadi solusi akhir). Selain itu kita juga dapat mempertimbangkan penggunaan f yang telah dimodifikasi daripada *f* yang dalam beberapa keadaan di deskripsikan kemudian.

Dalam metode pencarian tabu, kondisi berhenti dapat berupa:

- *N(i,k+1) = tidak terdefinisi*
- *k* lebih besar daripada bilangan maksimum pada pengulangan
- banyaknya pengulangan selama peningkatan terakhir *i** lebih besar dari bilangan tertentu.
- Pembuktian dapat diberikan daripada solusi optimum yang telah didapatkan.

Selama aturan-aturan berhenti ini memungkinkan untuk memiliki beberapa pengaruh dalam prosedur pencarian dan pada hasil-hasilnya, penting untuk menyadari bahwa pendefinisian *N(i,k)* dalam tiap pengulangan *k* dan pemilihan *V** adalah hal yang krusial.

Definisi dari *N(i,k)* menyatakan secara tidak langsung bahwa beberapa solusi yang telah dikunjungi dihapus dari *N(i),* mereka dianggap sebagai solusi-solusi tabu yang harus dihindari dalam pengulangan selanjutnya. Sebagai contoh, pemeliharaan pada pengulangan *k* sebuah list *T(list tabu)* pada solusi yang telah dikunjungi terakhir *|T|* akan mencegah terjadinya siklus pada ukuran paling banyak sebesar *|T|*. Pada kasus ini, kita bisa mengambil *N(i,k)=N(i)-T*. Namun list *T* kemungkinan tidak dapat digunakan secara praktis. Oleh karena itu, kita akan mendeskripsikan proses eksplorasi pada S dalam masa pergerakan dari satu solusi ke solusi lainnya. Untuk setiap solusi *I* dalam *S*, kita dapat mendefinisikan *M(i)* sebagai himpunan gerak yang dapat digunakan oleh *i* untuk memperoleh solusi baru *j* (notasi: $j=i \nabla m$). Secara umum kita dapat menggunakan gerakan-gerakan yang dapat dibalik. Untuk setiap *m* terdapat gerakan m^{-1} sehingga $(i \nabla m) \nabla m^{-1} = i$.

Jadi daripada mempertahankan list *T* dari solusi-solusi yang telah dikunjungi, kita dapat secara sederhana memelihara jalur gerak terakhir *|T|* atau gerak balik terakhir *|T|* yang diasosiasikan dfengan gerakan-gerakan yang sebenarnya telah dilakukan. Maka dengan jelas bahwa pembatasan yang ada adalah kehilangan informasi, dan hal itu tidak menjamin tidak terjadinya siklus dengan panjang paling banyak *|T|*.

Untuk kepentingan efisiensi, maka diperlukan penggunaan beberapa *list Tγ* dalam satu waktu maka beberapa unsur pokok *tγ* (dari *i* atau dari *m*) akan diberikan tabu status untuk mengindikasi bahwa unsur pokok ini sedang tidak diperbolehkan untuk terlibat dalam pergerakan. Secara umum pergerakan untuk tabu status adalah fungsi tabu status pada unsur-unsur pokoknya yang dapat diubah pada setiap pengulangan.

Suatu gerakan *m* (digunakan pada solusi *i*) akan menjadi gerak Tabu jika semua kondisi telah dipenuhi. Ilustrasi dari konsep ini dapat lebih dimengerti pada penggunaan metode pencarian tabu pada penjadwalan pekerjaan (*Job Scheduling*).

Kekurangan lain dari simplifikasi pada *list* tabu (penggantian solusi menjadi gerak) adalah fakta bahwa kita mungkin memberikan status tabu ke solusi-solusi yang belum dikunjungi. Maka kita pun dapat memaksakan perenggangan dari tabu status. Kita dapat menguasai tabu status saat beberapa tabu solusi akan terlihat menarik. Hal ini akan dilakukan dimaksudkan untuk tingkat kondisi aspirasi (*aspiration level conditions*)

Gerak tabu *m* digunakan pada solusi i yang mungkin tampak menarik karena itu diberikan sebagai contoh sebuah solusi yang lebih baik dari pada yang telah ditemukan. Kita akan dapat menerima *m* tanpa memperhatikan statusnya. Kita akan melakukan hal tersebut jika *m* memiliki tingkat aspirasi (*aspiration level*) *a(i,m)* yang lebih baik daripada permulaan *A(i,m)*.

Sekarang kita dapat mendefinisikan karakteristik dari prosedur pencarian tabu dalam langkah-langkah berikut, antara lain:

1. Memilih solusi awal *i* dalam *S*. Tetapkan *i*=I* dan *k=0*.
2. Tetapkan *k=k+1* dan bangkitkan sebuah subset *V** dari solusi dalam *N(I,k)* sehingga salah satu dari kondisi tabu tγ yang melanggar *(γ=1,2,...,t)* atau setidaknya satu kondisi aspirasi aγ yang memiliki *(γ=1,2,...,a)*.
3. Pilih *j* terbaik melalui *j=i∀m* dalam *V** dan tetapkan *i=j*.
4. Jika *f(i)<f(i*)* maka tetapkan *i*=i*.
5. Perbaharui kondisi Tabu dan aspirasi.
6. Jika kondisi berhenti ditemukan, maka proses dihentikan. Jika tidak, kembali ke langkah 2.

Hal-hal di atas dapat dikatakan sebagai bahan-bahan dasar dari metode pencarian tabu yang nantinya akan digunakan untuk memecahkan masalah penjadwalan pekerjaan (*job scheduling*) pada pembahasan selanjutnya.

4. Penerapan Algoritma Tabu Search dalam Job Shop Scheduling Problem

4.1. Penggunaan Tabu List

Tabu Search adalah sebuah *meta-heuristic* untuk kendali *local search* yang secara deterministik mencoba untuk menghindari solusi yang baru-baru saja dikunjungi[3]. Secara spesifik, algoritma ini mengelola sebuah *Tabu List* yang berisi perpindahan yang terlarang. *List* ini mengikuti aturan LIFO dan biasanya sangat pendek (panjangnya biasanya sebesar $O(\sqrt{N})$, dimana *N* adalah jumlah total dari operasi). Setiap saat ada langkah yang diambil, maka langkah itu akan ditempatkan dalam tabu list.

Salah satu komponen paling penting dalam algoritma *tabu search* adalah *Neighborhood function* (*NC*), dimana fungsi tersebut secara signifikan akan mempengaruhi *running time* dan kualitas dari solusi yang dihasilkan.

Dalam tabu list, elemen yang ditempatkan di dalam list adalah busar yang dibalik, dan langkah dianggap tabu jika ada komponen dalam busar adalah tabu.

Pseudo code Algoritma *Tabu Search* untuk *Job Shop Scheduling Problem* (*JSSP*)

```
function
TabuSearch(JSSP)→solution
{inisiasi awal}
sol ← initSolution(JSSP)
bestCost ← cost(sol)
bestSolution ← sol
tabuList ← ∅
while KeepSearching() do
 N_valid ← s{s∈N(sol)|Move[sol,s]∉
        tabuList}
 if N_valid ≠ ∅ then
   sol'←x{x∈N(sol)|∀y∈N_valid(sol)
   cost(x) ≤ cost(y)}
 endif
 updateTabuList(sol')
 if cost(Move[sol,sol']) <
 bestCost then
   bestSolution ← sol'
   bestCost ← cost(sol')
 endif
   sol ← sol'
endwhile
return bestSolution
```

4.2. Mendapatkan Solusi Yang Lebih Baik

Tujuan akhir dari semua algoritma optimalisasi adalah mendapatkan dengan cepat solusi optimal yang terdekat. *Tabu search* sudah terbukti cukup efektif untuk menyelesaikan persoalan tersebut, namun beberapa pelaku riset telah menyatakan bahwa ada beberapa kondisi dimana algoritma tersebut bisa bekerja dengan lebih baik. Beberapa langkah yang bisa membuat algoritma *tabu search* lebih optimal antara lain :

1. *Aspiration Criterion*. Ini adalah sebuah fungsi yang mengontrol, kapan saja dia bisa mengabaikan sebuah *tabu-state* dari sebuah langkah. *Aspiration criterion,* dalam gambaran general, akan menerima sebuah *tabu move* jika *cost* dari solusi tersebut akan bisa lebih baik daripada *cost* dari solusi terbaik yang sudah pernah dikunjungi.
2. *Resizing the tabu list*. Ini adalah teknik untuk memodifikasi panjang dari *tabu list*. Pada kebanyakan kasus, *tabu list* akan diperpendek jika solusi yang lebih baik ditemukan, dan sebaliknya, akan diperpanjang jika solusi yang lebih buruk ditemukan. Asumsi utama dibalik teknik ini adalah ketika sebuah solusi yang memenuhi telah ditemukan, kemungkinan masih ada solusi lain dalam beberapa langkah selanjutnya, yang juga memenuhi. Menambah jumlah dari pergerakan *neighboring* yang valid akan membuat algoritma bisa menemukan solusi-solusi yang

lebih baik dengan kemungkinan yang lebih besar.

3. *Restoring the Best Known Solution*. Salah satu cara untuk menghindari pemborosan waktu untuk memeriksa solusi yang kurang optimal adalah dengan mengeset ulang **curent solution** sebagai *the best known solution* (solusi yang terbaik yang dari seluruh solusi yang sudah diperiksa) secara perodik. Waktu tunggu sebelum mengeset ulang harus diperhitungkan secara hati-hati, tidak bisa terlalu cepat ataupun terlalu lama. Dalam praktiknya, dengan delay time yang sesuai (misalnya 800-1000 iterasi), me-reset current solution akan bisa meningkatkan kualitas solusi yang sudah ditemukan.

5. Kesimpulan

Setelah meneliti metode pencarian tabu dapat disimpulkan bahwa walaupun metode ini belum diketahui secara pasti bekerja optimal dalam keadaan apa dan mengapa metode ini bekerja dengan baik, namun dalam melakukan pencarian untuk memecahkan beberapa masalah terutama job scheduling problem metode ini terbukti cukup efektif dibandingkan dengan motede-metode lainnya.

Penelitian ini telah menunjukkan bahwa ada kemungkinan untuk memakai algoritma tabu search dan melakukan beberapa penyesuaian untuk bisa menghasilkan solusi yang layak pada sebuah kelas yang lebih lebar dari sebuah permasalahan.

6. Referensi

1. Alain Hertz, Eric Taillard, and Dominique de Werra, *A Tutorial On Tabu Search,* http://www.cs.colostate.edu/~whitley/CS640/hertz92tutorial.pdf, diakses tanggal 18 Mei 2006
2. Albert Ludwigs, *Scheduling*, Universitat Freiburg Press,Germany, 2006
3. Keith Schmidt, *Using Tabu Search to Solve the Job Shop Scheduling Problem with Sequence Dependent Setup Times*, http://cs.brown.edu , diakses tanggal 18 Mei 2006
4. Jean Paul Watson, L.Darrel Whitley, and Adele E.Howe, *A Dynamic Model of Tabu Search for the Job-Shop Scheduling Problem,* Colorado State University, 2004
5. http://wikipedia.org