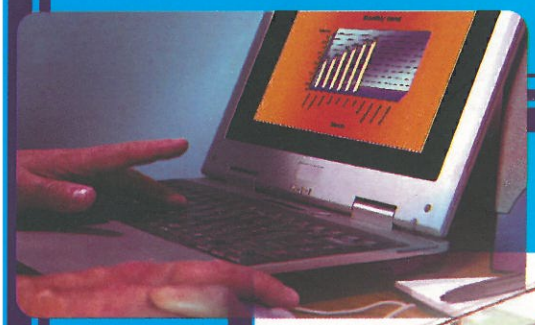
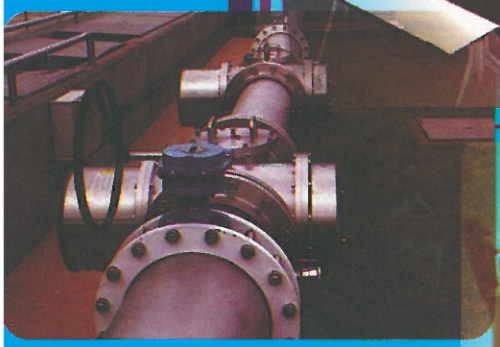
ISSN : 2088 - 8015

INTEGRA

Jurnal Teknik dan Manajemen Industri





INT | Volume 2 | Nomor 1 | Hlm.1-106 | Bandung | 15 JUNI 2012 | ISSN : 2088 - 8015

ISSN : 2088 - 8015

INTEGRA

Jurnal Teknik dan Manajemen Industri

Volume 2, Nomor 1

ISSN : 2088 - 8015

INTEGRA

Jurnal Teknik dan Manajemen Industri

Volume 2, Nomor 1 Juni 2012

Pengembangan Teknik Pencarian Optimal Menggunakan Algoritma *Generate and Test* dengan Diagram *Precedence* (GTPRE)

Development of Optimal Search Using Generate and Test Algorithm with Precedence Diagram (GTPRE)

Victor Suhandi
Universitas Kristen Maranatha
E-mail: victorsuhandi@yahoo.com

Abstrak

Pengembangan teknik pencarian cerdas dengan bantuan komputer sudah sangat banyak. Beragam karakteristik ditunjukkan dari metode yang dikembangkan, ada yang menjamin hasil optimal namun sangat lama dalam proses pencarian dan sebaliknya ada yang lebih menekankan pada perolehan hasil yang cepat namun tidak menjamin hasil yang optimal. Dalam penelitian ini dikembangkan algoritma GTPRE untuk memecahkan permasalahan tertentu yang cocok agar memperoleh hasil yang cepat dan optimal. Bermula dari algoritma Generate and Test yang membangkitkan seluruh kombinasi yang ada, kemudian ditambahkan kendala berupa diagram precedence yang dapat meredam ledakan kombinasi, sehingga hasil yang diperoleh menjadi lebih cepat dan tetap menjaga jaminan keoptimalan. Reduksi kombinasi dari kasus penjadwalan job yang disajikan sangat besar yaitu dari 5040 kombinasi untuk 7 job menjadi 36 kombinasi saja. Permasalahan yang cocok untuk dipecahkan menggunakan algoritma GTPRE ini adalah permasalahan dengan diagram precedence yang memiliki baris yang sedikit dan simpul yang banyak.

***Kata kunci**: heuristik, generate and test, diagram precedence*

Abstract

Development of intelligent search techniques with the help of a computer is already growth abundantly. Many characteristics of the methods which had developed are shown, some methods are guarantee optimal results but very long in the search process, another methods have greater emphasis on acquiring quick results but do not guarantee optimal results. In this study GTPRE algorithm developed to solve specific problems in order to obtain suitable and optimal results quickly. Starting from the Generate and Test algorithm that generates all combinations exist, then added precedence constraints in the form of diagrams, which can reduce the explosion of combinations, so that the process become faster and the results still maintain the optimality. Using a job scheduling case, the reduction of the sequence combination of jobs are significant, reduce from 5040 to 36 combinations for 7 jobs. GTPRE is suitable for problems that have a few lines and many joint in precedence diagram.

***Keywords**: heuristic, generate and test, precedence diagram*

1. Pendahuluan

Pada saat ini banyak ahli yang sudah mengembangkan teknik pencarian cerdas. Banyak teknik pencarian cerdas menggunakan bantuan komputer yang hasilnya mendekati optimal atau bahkan optimal. Terkadang teknik pencarian cerdas tidak menjamin tercapainya nilai optimal. Hal ini dikembangkan dengan alasan menyederhanakan waktu pencarian agar waktu pencarian tidak menjadi sangat lama. Teknik pencarian cerdas ini dituangkan ke dalam langkah-langkah sehingga disebut sebagai algoritma pencarian cerdas. Contoh algoritma yang telah dikembangkan adalah algoritma genetika, algoritma pencarian tabu, algoritma koloni semut, dan lain sebagainya.

Pada sisi lain teknik pencarian yang menjamin hasil yang optimal juga sudah dikembangkan. Salah satu teknik yang menjamin hasil optimal adalah *generate and test*, dimana algoritma ini membangkitkan seluruh kombinasi yang ada dan diuji satu persatu, kemudian dipilih alternatif solusi yang terbaik. Banyak kasus yang menghasilkan ledakan kombinasi sehingga waktu untuk membangkitkan seluruh kombinasi menjadi tidak layak lagi, sehingga metode ini dinilai hanya mampu menyelesaikan masalah yang sangat sederhana.

Dalam hal ini peneliti tertarik pada terjaminnya hasil yang optimal namun dalam waktu yang cukup layak. Teknik pencarian menggunakan *generate and test* yang dikombinasikan dengan diagram *precedence* (GTPRE) dapat berpotensi menjadi salah satu teknik ini. Pada algoritma ini, jumlah ledakan kombinasi akan dapat diredam dengan diagram *precedence.*

Peredaman ledakan kombinasi dari metode *generate and test* yang dikombinasikan dengan diagram *precedence* terjadi saat pembangkitan cabang-cabang untuk level berikutnya harus mengikuti diagram *precedence*, sehingga tidak untuk seluruh elemen yang ada. Apabila terdapat 10 elemen, untuk metode *Generate and test*, setiap cabang akan membangkitkan 10 cabang pada level berikutnya. Sedangkan pada algoritma GTPRE, hanya sejumlah elemen yang sesuai pada diagram *precedence* yang akan mengalami pembangkitan cabang-cabang untuk level beikutnya.

Penggunaan algoritma GTPRE sangat cocok untuk permasalahan yang memiliki diagram *precedence* seperti penyeimbangan lini, penjadwalan pekerjaan, penentuan rute yang dibatasi *precedence*, dan lain sebagainya.

Beberapa hal yang akan dilakukan dalam penelitian ini yakni:
- membandingkan keefisienan antara metode *generate and test* dengan Metode GTPRE.
- menentukan permasalahan yang cocok untuk dipecahkan menggunakan GTPRE dengan melihat keragaman bentuk dari diagram *precedence.*

2. Studi Literatur

Teknik pencarian cerdas merupakan salah satu aspek dari kecerdasan buatan dengan menggunakan *problem-solving agents* yang menentukan apa yang harus dilakukan dengan membuat urutan tindakan yang menuntun ke keadaan yang diharapkan. Agen ini akan berusaha untuk memaksimalkan hasil yang dapat dicapai dengan melihat tujuannya. Perumusan tujuan merupakan langkah awal untuk pemecahan masalah. Perumusan tujuan harus berdasarkan situasi saat ini dan ukuran kinerjanya. Selain itu tujuan yang jelas akan memberikan batasan dari apa yang hendak dicapai.

Setelah perumusan tujuan, kemudian dilakukan perumusan masalah. Perumusan masalah merupakan sebuah proses untuk menentukan tindakan yang akan diambil dan keadaan apa yang harus dipertimbangkan untuk mencapai tujuan. Algoritma pencarian akan menjadikan permasalahan ini sebagai input, dan solusi yang dihasilkan berupa urutan tindakan. Proses untuk mencari urutan ini yang disebut pencarian (Russell, 2003).

Terdapat empat komponen dari perumusan masalah:
1. Keadaan awal.
2. Deskripsi tindakan-tindakan yang mungkin, pada umumnya sesuai fungsi suksesor.
3. Pengujian *goal.*
4. Fungsi *path cost* yang menggambarkan ukuran kinerja pencarian.

Solusi permasalahan dinyatakan dengan sebuah jalur dari keadaan awal hingga keadaan tujuan. Kualitas solusi diukur dari fungsi *path cost.* Solusi optimal merupakan solusi yang memiliki *path cost* terkecil dari semua solusi yang tersedia.

Ukuran kinerja yang menjadi patokan antara lain (Russell, 2003):

- Kelengkapan: apakah algoritma menjamin tercapainya tujuan?
- Optimalitas: apakah algoritma menjamin tercapainya hasil yang optimal?
- Kompleksitas waktu: berapa lama waktu yang dibutuhkan untuk mencapai solusi?
- Kompleksitas ruang: berapa banyak memori yang dibutuhkan untuk melakukan pencarian?

Pada dasarnya terdapat dua teknik pencarian yaitu teknik pencarian buta (*blind search*) dan pencarian heuristik. Teknik pencarian buta yang sangat umum adalah *Breadth first search* (BFS) dan *Depth first search* (DFS). BFS memprioritaskan pencarian dalam level yang sama dahulu baru bergerak ke level yang berikutnya. Sebaliknya DFS memprioritaskan kedalaman dahulu dengan bergerak menuju level berikutnya baru kemudian bergerak dalam level yang sama. Sedangkan teknik pencarian heuristik yang sudah popular yaitu *generate and test*, *hill climbing*, algoritma genetika, dan lain-lain (Kusumadewi, 2005).

Algoritma *generate and test* merupakan penggabungan antara DFS dengan pelacakan mundur (*backtracking*), yaitu bergerak ke belakang menuju pada suatu keadaan awal. Nilai pengujian berupa 'ya' atau 'tidak'. Algoritmanya adalah sebagai berikut:

1. Bangkitkan suatu kemungkinan solusi (membangkitkan suatu titik tertentu atau lintasan tertentu dari keadaan awal).
2. Uji apakah *node* tersebut merupakan solusi, dengan membandingkannya dengan keadaan *goal*.
3. Apabila solusi ditemukan maka keluar. Jika tidak maka ulangi kembali langkah 1 (Kusumadewi, 2005).

Penerapan algoritma ini biasanya untuk permasalahan yang sulit dalam membuat model optimalnya. Permasalahan juga terbagi lagi menjadi permasalahan yang memiliki kendala *precedence* dan ada juga yang tidak. Penjadwalan pada mesin biasanya memiliki *precedence* tetapi lain halnya pada permasalahan perjalanan *salesman* (Kusiak, 2000). Sehingga permasalahan yang cocok untuk algoritma *generate and test* dengan *precedence* tentunya adalah yang memiliki diagram *precedence*.

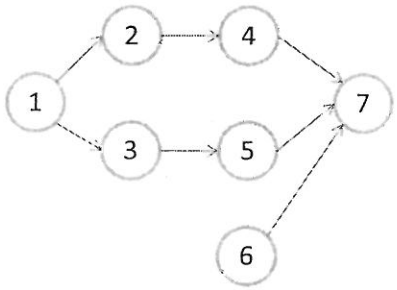
3. Algoritma Generate and Test

Algoritma *generate and test* yang dituliskan oleh Kusumadewi (2005) belum rinci, sehingga sulit untuk dibuat program komputernya. Algoritma *generate and test* secara lebih rinci adalah sebagai berikut:

1. Tentukan keadaan awal (*initial state*) yang menjadi node dan keadaan sekarang (*current state*) pada level 0.
2. Tentukan himpunan *nodes* yang dapat dipilih pada level berikutnya (*fringe:* himpunan *leaf nodes*). Apabila *fringe* berupa himpunan kosong maka tidak ada solusi untuk urutan *nodes* tersebut, lakukan *backtracking* (langkah 5).
3. Dengan mengacu pada algoritma DFS yaitu memilih *node* pertama dari *fringe*. *Node* ini menjadi *current state* pada level yang baru. *Node* yang dipilih dikeluarkan dari *fringe* pada level sebelumnya. (Contoh pada Gambar 2, *fringe* level 0 adalah (*node* 2, *node* 3, *node* 4, *node* 5, *node* 6, *node* 7), *node* 2 merupakan *node* pertama di dalam *fringe* tersebut.)
4. Apakah tujuan (*Goal*) tercapai, yaitu: apakah urutan *node* yang diperoleh dapat menjadi solusi? Jika 'ya' maka:
   4.1. hitung *path cost*,
   4.2. Apakah *path cost* < *best path cost* jika 'ya' maka *best path cost* = *path cost* dan *best path* = *path* (urutan nodes yang diperoleh) dan
   4.3. lakukan *backtracking* (langkah 5).
   Jika 'tidak' maka ulangi langkah 2.

5. Apakah level = 0? Jika 'ya' maka Tampilkan *best pathcost* dan *bestpath* kemudian berhenti. Jika 'tidak' maka lakukan pelacakan mundur dengan berpindah ke *node* pada level sebelumnya dan menjadi *current state.* Kemudian lakukan langkah 3.

4. Algoritma GTPRE



Gambar 1. Diagram *Precedence*

Pengembangan algoritma *generate and test* yang mempertimbangkan diagram *precedence* (Gambar 1) akan memodifikasi algoritma *Generate and test* dan juga penggunaan matriks untuk menggambarkan diagram *precedence*. Matriks *precedence* yang digunakan adalah sebagai berikut:

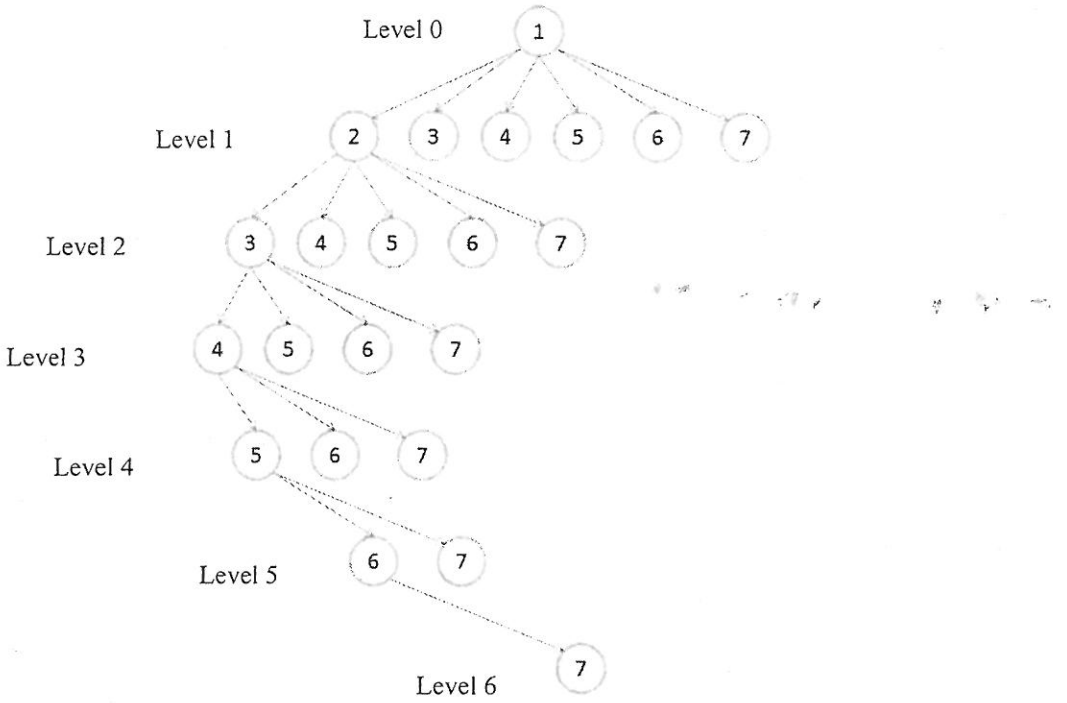
$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \\ 0 & 0 & 0 \\ 4 & 5 & 6 \end{bmatrix}$$

Baris matriks menyatakan suatu *state*, *state* tersebut didahului oleh *state* lain yang dinyatakan dalam matriks pada baris yang sama. Nilai 0 berarti bukan merupakan pendahulu. Sehingga *fringe* atau *state* yang berpotensi untuk segera dipilih adalah *state* 1 dan *state* 6 pada contoh matriks tersebut. Baris pertama menyatakan *state* 1 yang tidak memiliki pendahulu yang belum dilalui. Sedangkan untuk baris ke-7 menyatakan *state* 7 memiliki pendahulu yang belum dilalui yaitu: *state* 4, *state* 5, dan *state* 6.

Apabila *state* 1 dipilih maka nilai di dalam matriks yang bernilai 1 menjadi 0 dan nilai pada baris 1 menjadi 99 yang berarti *state* 1 sudah dipilih menjadi *node* pada suatu *path* seperti berikut: Hal ini berarti *state* 1 tidak dapat dipilih kembali, dan selanjutnya yang dapat dipilih adalah *state* 2, *state* 3, dan *state* 6.

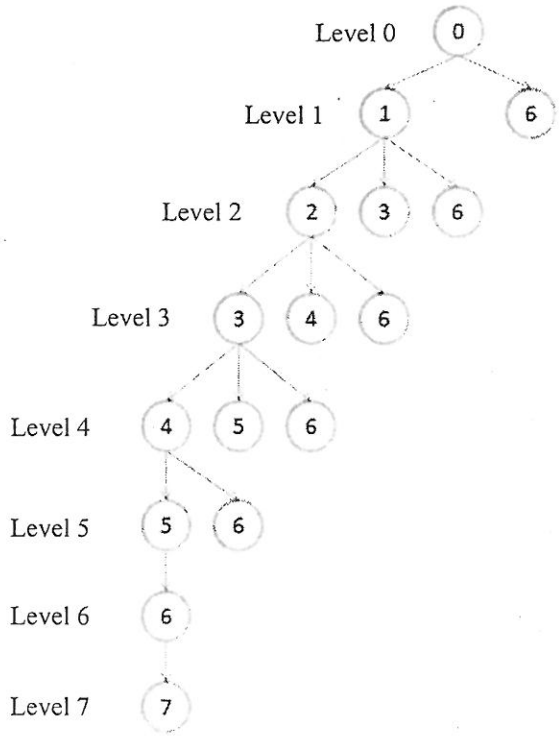
$$\begin{bmatrix} 99 & 99 & 99 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \\ 0 & 0 & 0 \\ 4 & 5 & 6 \end{bmatrix}$$

Fringe pada level ini menjadi {*state* 2, *state* 3, *dan state* 6}. *Backtracking* akan dilakukan apabila *fringe* merupakan himpunan kosong.



Gambar 2. Pembangkitan *fringe* dan pemilihan *state* hingga level 6 tanpa dibatasi oleh matriks *precedence*

Gambar 2 menunjukkan *nodes* yang dibangkitkan untuk setiap level yang berbeda. Pada level 0 *node* 1 merupakan satu-satunya *node* yang ada, sehingga akan terpilih untuk dikembangkan untuk level berikutnya. Setelah *node* 1 dipilih maka himpunan *node* yang mungkin dapat dipilih (*fringe*) adalah *nodes* pada level 1 (2, 3, 4, 5, 6, 7). Kemudian salah satu *node* pada level 1 akan dipilih dan bangkitkan *fringe* pada level tersebut, demikian seterusnya hingga mencapai level tertinggi, untuk contoh ini yaitu level 6. Pada Gambar 2, potensi ledakan kombinasi untuk algoritma *generate and test* dapat terlihat pada *nodes* yang belum mencapai level 6. Kombinasi yang dihasilkan akan mencapai 7 faktorial yaitu sebesar 5040 kombinasi.



Gambar 3. Pembangkitan *fringe* dan pemilihan *state* hingga level 7 dengan dibatasi oleh matriks *precedence*

Pada Gambar 3, apabila *node* 1 pada level 1 dipilih maka *fringe*-nya adalah (2, 3, 6). *Fringe* ini hanya terdiri dari tiga anggota hanya separuh dari pembangkitan tanpa *precedence*. Pada Gambar 3, potensi ledakan kombinasi untuk algoritma GTPRE dapat terlihat pada *nodes* yang belum mencapai level 7 jauh lebih sedikit dibandingkan Gambar 2.

Algoritma GTPRE adalah sebagai berikut:

1. Tentukan keadaan awal (*initial state*) yang menjadi *node* dan keadaan sekarang (*current state*) pada level 0 dengan memperhatikan matrik *precedence*. (Mencari *state* yang tidak memiliki pandahulu)

   Apabila terdapat lebih dari satu *state* yang dapat dipilih saat awal maka digunakan *node* pembantu sebagai *initial state*. Sebaiknya *node* pembantu ini selalu digunakan untuk dapat menyatakan matriks *precedence* level 0 menjadi sama dengan diagram *precedence* pada permasalahan awal. Apabila tidak maka matriks *precedence* level 0 sudah memperhitungkan sebuah *state* yang dipilih pada level 0. (Langkah ini berbeda dengan algoritma *generate and test*)

2. Tentukan himpunan nodes yang dapat dipilih pada level berikutnya (*fringe:* himpunan *leaf nodes*) dengan memperhatikan matriks *precedence* pada level sekarang. (Langkah ini sesuai dengan penjelasan Gambar 3.)

   State yang dijadikan sebagai *leaf node* harus memenuhi syarat tidak memiliki pendahulu yang belum dipilih sebagai *node* pada *path* yang sama. (Langkah ini berbeda dengan algoritma *generate and test.*)

   Apabila *fringe* berupa himpunan kosong maka tidak ada solusi untuk urutan *nodes* tersebut, lakukan *backtracking* (langkah 5).

3. Dengan mengacu pada algoritma DFS yaitu memilih *node* pertama dari *fringe*. *Node* ini menjadi *current state* pada level yang baru. (Contoh pada Gambar 3, *fringe* level 0 adalah (*node* 1, *node* 6), *node* 1 merupakan *node* pertama dan *node* 6 merupakan *node* kedua.)

   Node yang dipilih dikeluarkan dari *fringe* pada level sebelumnya.

   Perbaharui matriks *precedence* dengan mengubah *state* yang sudah terpilih sebagai *node* pada suatu *path*. (Langkah ini berbeda dari algoritma *generate and test.*)

4. Apakah tujuan (*goal*) tercapai, yaitu: apakah urutan *node* yang diperoleh dapat menjadi solusi?
   Jika 'ya' maka:
   4.1. hitung *path cost*,
   4.2. Apakah *path cost* < *best path cost* jika 'ya' maka *best path cost* = *path cost* dan *best path* = *path* (urutan *nodes* yang diperoleh) dan
   4.3. lakukan *backtracking* (langkah 5).
   Jika 'tidak' maka ulangi langkah 2.

5. Apakah level = 0? Jika 'ya' maka Tampilkan *best pathcost* dan *bestpath* kemudian berhenti. Jika 'tidak' maka lakukan pelacakan mundur dengan berpindah ke *node* pada level sebelumnya dan menjadi *current state*. Kemudian lakukan langkah 3.

5. Hasil dan Analisis

Contoh kasus pada Gambar 1 harus dilengkapi dengan parameter lain. (Gambar 1 merupakan diagram *precedence* dari kasus ini.) Apabila kasusnya merupakan kasus penjadwalan tugas atau *job*

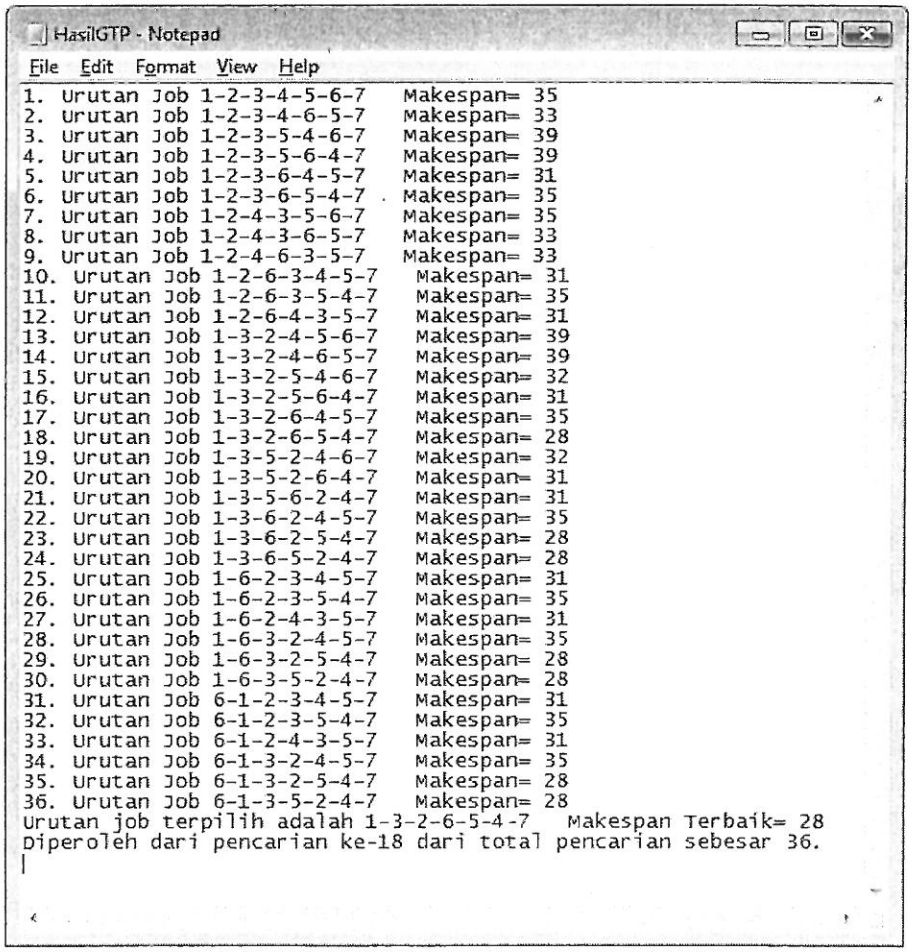
maka memerlukan data mesin atau peralatan atau departemen dimana *job* berlangsung dan berapa lama waktu yang dibutuhkan. Untuk contoh kasus ini digunakan data pada Tabel 1.

Tabel 1. Data penggunaan mesin dan lama waktu pengerjaan untuk setiap *job*.

Job	Mesin	Waktu (jam)
1	1	5
2	2	8
3	2	6
4	3	4
5	3	7
6	3	4
7	2	5

Hasil penggunaan algoritma GTPRE ditunjukkan pada Gambar 4 dan Gambar 5. Hasil dari algoritma ini menunjukkan seluruh kombinasi yang dapat dihasilkan dengan batasan diagram *precedence* yakni sebanyak 36 kombinasi urutan *job* yang dapat dilakukan. Nilai optimal yang diperoleh yaitu dengan *makespan* 28 jam (asumsi tujuannya adalah ingin meminimasi *makespan*) dan yang didapatkan pada nomor kombinasi ke-18.

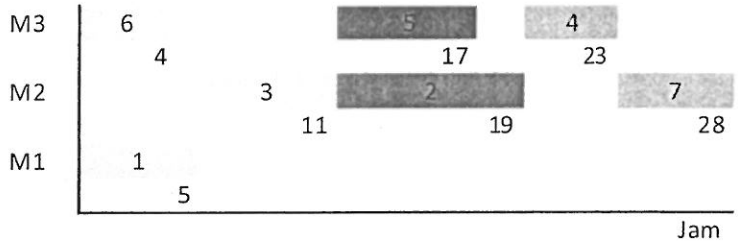
Apabila kita membandingkan algoritma GTPRE dengan algoritma *generate and test,* dimana algoritma GTPRE menghasilkan 36 kombinasi dan algoritma *generate and test* akan menghasilkan 5040 kombinasi. Kita akan dapat membandingkan betapa hematnya metode GTPRE ini. Hal ini menunjukkan metode GTPRE mampu meredam ledakan kombinasi dengan mempertahankan jaminan hasil yang optimal pada contoh kasus ini.



Gambar 4. Hasil yang diperoleh menggunakan Algoritma GTPRE.

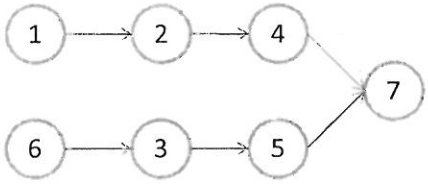
Pada Gambar 4, hasil yang diperoleh merupakan urutan *job* pertama yang menghasilkan *makespan* terendah, dalam kasus ini *makespan* terendah sebesar 28 satuan waktu. Urutan penjadwalannya adalah 1-3-2-6-5-4-7, yang berarti *job* 1 dijadwalkan pada urutan pertama, selanjutnya *job* 3 dan seterusnya hingga *job* 7. Urutan *job* ini diperoleh dari pencarian ke-18 dari total pencarian sebesar 36.

Gambar 5 menunjukkan grafik penempatan *job* pada mesin yang sesuai. Terdapat asumsi waktu siap ketiga mesin (M1, M2, dan M3) pada saat awal adalah 0. *Job* 1 dikerjakan pada mesin 1 dan dipetakan pertama kali sesuai urutan yang diperoleh. *Job* 3 dijadwalkan setelah *job* 1 tetapi pada mesin 2, waktu siap mesin 2 adalah 0 dan waktu siap *job* 3 adalah 5 karena sesuai diagram *precedence*. Berbeda keadaannya untuk *job* 6, *job* ini dijadwalkan pada mesin 3, dimana waktu siap mesin dan waktu siap *job* adalah 0, sehingga *job* 6 dapat langsung dipetakan pada mesin 3 dimulai dari 0.



Gambar 5. Grafik perhitungan *makespan* dengan menggunakan algoritma GTPRE

Pengaruh diagram *precedence* terhadap besarnya ruang keadaan yang ada dapat kita uji dengan melakukan modifikasi diagram *precedence*. Salah satu diagram *precedence* pembanding dapat dilihat pada Gambar 6.



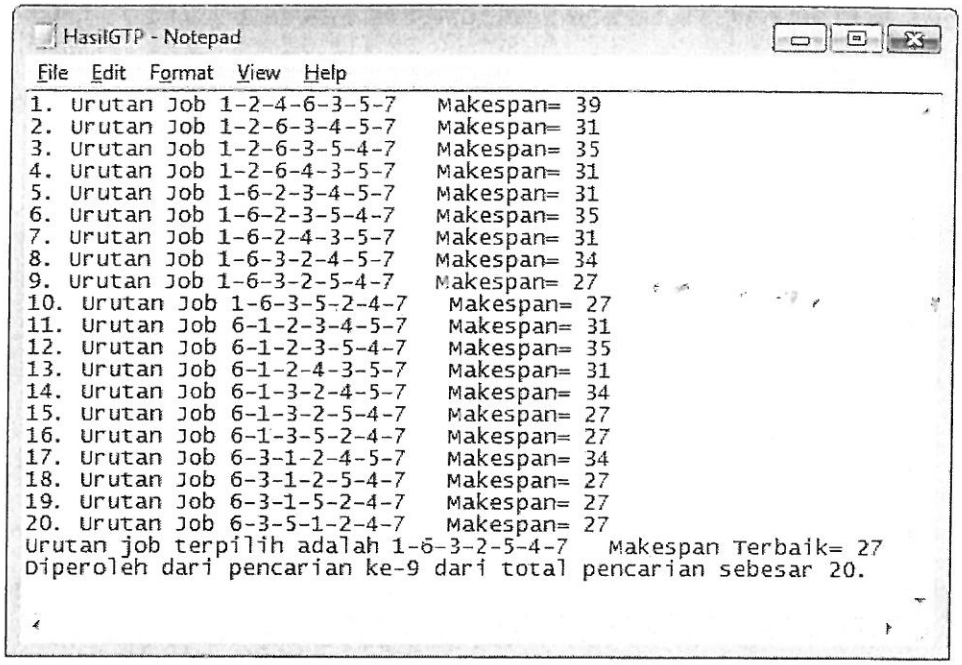
Gambar 6. Diagram *precedence* pembanding 1

Matriks *precedence* berubah menjadi sebagai berikut:

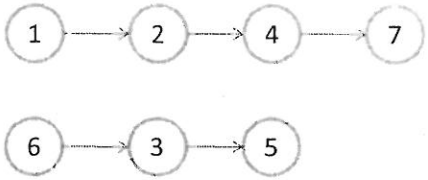
$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 6 & 0 \\ 2 & 0 \\ 3 & 0 \\ 0 & 0 \\ 4 & 5 \end{bmatrix}$$

Matriks ini berukuran 7 x 2 berubah dari sebelumnya dengan ukuran 7 x 3.

Hasil yang diperoleh dapat dilihat pada Gambar 7.



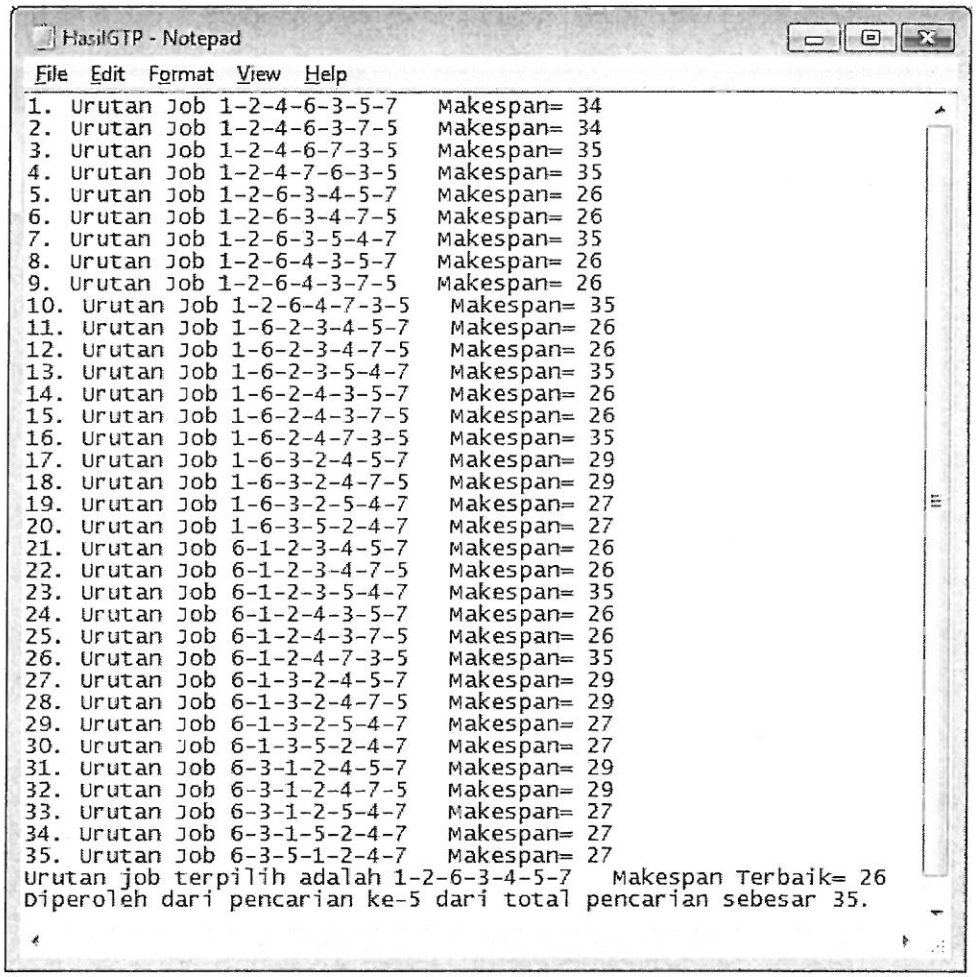
Gambar 7. Hasil algoritma GTPRE dengan menggunakan diagram *precedence* pembanding 1



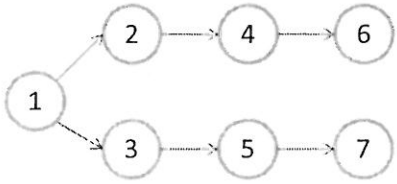
Gambar 8. Diagram *precedence* pembanding 2.

Matriks *precedence* berubah menjadi matriks yang berukuran 7 x 1 sebagai berikut:

$$\begin{bmatrix} 0 \\ 1 \\ 6 \\ 2 \\ 3 \\ 0 \\ 4 \end{bmatrix}$$



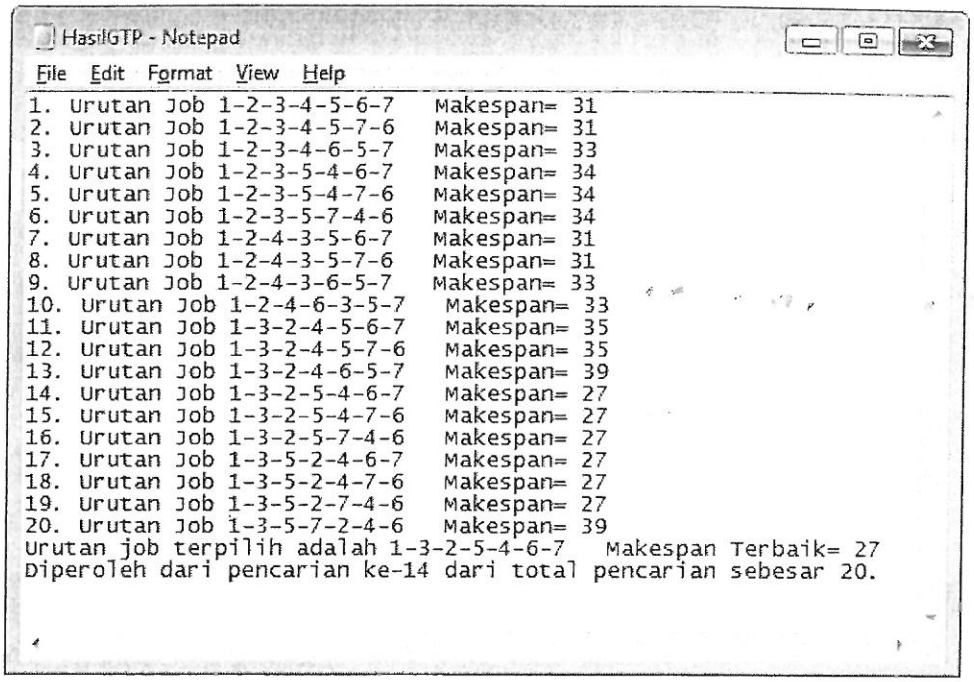
Gambar 9. Hasil algoritma GTPRE dengan menggunakan diagram *precedence* pembanding 2.



Gambar 10. Diagram *precedence* pembanding 3

Matriks *precedence* berubah menjadi sebagai berikut:

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$



Gambar 11. Hasil algoritma GTPRE dengan menggunakan diagram *precedence* pembanding 3

Dari pembandingan diagram *precedence* dapat dilihat ketika ada *job* yang merupakan simpul percabangan maka kombinasi yang dihasilkan menjadi lebih sedikit. Hal ini menggambarkan simpul akan mengurangi kebebasan *job* tersebut sehingga akan mengurangi kombinasi. Pada Gambar 8 dan Gambar 9 dimana diagram *precedence* tidak memiliki simpul menghasilkan jumlah kombinasi sebanyak 35. Sebaliknya untuk Gambar 6 dan Gambar 7 dimana terdapat simpul diakhir *precedence* menghasilkan kombinasi sebanyak 20. Hal yang sama untuk Gambar 10 dan Gambar 11 dimana simpul berada di depan menghasilkan kombinasi sebanyak 20 buah.

Pembandingan lain juga dapat dilakukan antara diagram *precedence* pada Gambar 1 dengan diagram *precedence* pada gambar lainnya. Gambar 1 memiliki jumlah baris yang lebih banyak yakni 3 baris sedangkan diagram yang lain hanya 2 baris. Hasil yang didapatkan pada Gambar 2 menunjukkan jumlah kombinasi sebanyak 36 lebih banyak dari yang lain. Hal ini menggambarkan penambahan baris dalam diagram *precedence* akan mengakibatkan peningkatan jumlah kombinasi karena penambahan baris akan menambah ukuran *fringe* (himpunan alternatif tindakan yang mungkin pada level pencarian berikutnya) yang terjadi. Dampak penambahan satu baris lebih besar dari dampak penambahan satu simpul.

Dari uraian diatas terlihat faktor pemicu dan peredam ledakan kombinasi. Secara sederhana kita dapat simpulkan bahwa penambahan baris dalam diagram *precedence* merupakan pemicu ledakan kombinasi dan penambahan simpul pada diagram *precedence* merupakan peredam ledakan kombinasi. Dalam hal ini kita dapat menggunakan algoritma GTPRE dengan melihat diagram *precedence* dari permasalahan yang akan dipecahkan, sehingga hasil optimal dapat diperoleh dan waktu pencarian yang tidak lama (layak).

Algoritma *generate and test* melakukan pencarian dengan DFS yang dikombinasikan dengan pelacakan mundur (*backtracking*). Begitu pula dalam algoritma GTPRE, pencarian dilakukan dengan DFS sesuai diagram *precedence* kemudian diteruskan dengan pelacakan mundur, sehingga alternatif di luar *precedence* sama sekali tidak dibangkitkan. Contohnya pada Gambar 1, *job* 7 tidak boleh mendahului *job* 6, pada algoritma GTPRE tidak akan pernah membangkitkan *job* 7 yang mendahului *job* 6 sebagai solusi. Hal yang menjamin perilaku ini adalah pembangkitan *fringe* yang selalu mengacu pada diagram *precedence*. Setiap *node* pada *fringe* merupakan *states* yang tidak melanggar diagram *precedence*. Berbeda dengan membangkitkan solusi tersebut dahulu baru

kemudian menguji apakah solusi tersebut valid (tidak melanggar diagram *precedence*). Apabila alternatif telah dibangkitkan berarti sudah terjadi pemborosan langkah pencarian.

6. Simpulan dan Saran

Algoritma GTPRE lebih efisien dari algoritma *Generate and Test* untuk permasalahan tertentu yang memiliki diagram *precedence*, contohnya dalam penelitian ini yaitu penjadwalan *job*.

Algoritma GTPRE semakin efisien apabila semakin sedikit jumlah baris yang ada pada diagram *precedence* dan semakin banyak simpul yang ada pada diagram *precedence*. Jumlah baris memiliki dampak yang lebih besar dibandingkan simpul pada diagram *precedence*.

Penelitian selanjutnya yaitu membandingkan algoritma GTPRE dengan algoritma pencarian cerdas lain yang tidak menjamin keoptimalan hasil. Perbandingan waktu keduanya untuk memecahkan suatu kasus menjadi pertimbangan yang utama. Apabila algoritma GTPRE dapat memberikan waktu yang tidak jauh berbeda dari teknik pencarian cerdas lain, maka algoritma ini sangat baik untuk digunakan karena menjamin hasil yang optimal.

7. Daftar Pustaka

Kusiak, A., (2000), *Computational Intelligence in Design and Manufacturing*, John Wiley & Son, Inc., Canada.

Kusumadewi, S., Purnomo, H., (2005), *Penyelesaian Masalah Optimasi dengan Teknik-teknik Heuristik*, Graha Ilmu, Yogyakarta.

Russell, S., Norvig, P., (2003), *Artificial Intelligence – A Modern Approach*, Prentice Hall, New Jersey.