Minimisasi waktu penyelesaian tugas berdepedensi dengan pekerja homogen terbatas menggunakan algoritma greedy

Candra Ramsi - 13514090[1]
Program Studi Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
[1]*candra_ramsi@arc.itb.ac.id*

***Abstract*—Manajemen tugas, pengaturan jadwal, dan penentuan deadline rilis produk merupakan hal yang penting dan sulit dilakukan manusia. Komputer juga memiliki masalah yang mirip dalam pengaturan jadwal penyelesaian tugas. Namun, sudah banyak bahasan dan implementasi algoritma pengaturan jadwal. Struktur tugas yang biasanya muncul dalam pengerjaan tugas adalah tugas yang memiliki depedensi satu dengan yang lainnya. Oleh karena itu, Masalah penetuan deadline rilis produk akan dipecahkan menggunakan algoritma greedy pada makalah ini. Terdapat tiga solusi greedy yang dibahas pada makalah ini dengan teknik pemilihan tugas masing-masing. Solusi greedy memanfaatkan teknik-teknik yang sudah ada dalam penjadwalan tugas pada komputer.**

Keywords—greedy; task distribution; task depedency; scheduling

I. LATAR BELAKANG

Manajemen tugas dan pengaturan jadwal merupakan hal yang sering dilakukan dalam lingkungan pekerjaan. Pengaturan jadwal terutama merupakan hal yang cukup sulit dilakukan. Beberapa perangkat lunak sudah dapat digunakan untuk menyelesaikan sebagian dari masalah ini seperti *Trello*, *Asana*, dan *Nostromo*.

Gambar. 1.Trello, Asana, Nostromo

Perangkat lunak manajemen tugas bertujuan membantu tim dalam memanajemen tugasnya. Perangkat-perangkat lunak tersebut tidak ditunjukan untuk menentukan waktu rilis produk. Waktu rilis produk biasanya telah ditentukan sebelumnya dan pekerjaan mengikuti agar waktu rilis tersebut tercapai.

Perangkat-perangkat lunak tersebut juga tidak memberikan cara yang mudah untuk menandakan depedensi tugas seperti tugas A membutuhkan tugas B dan C untuk selesai sebelum dapat dikerjakan. Dipedensi tugas ini banyak terjadi pada berbagai macam pekerjaan. Misalnya, dalam proses membuat kue, proses pembuatan adonan harus dilakukan sebelum proses pemanganggan kue.

Masalah ini tidak terbatas pada manajemen manusia tetapi juga manajemen tugas pada komputer. Dalam sistem komputer, Distribusi tugas juga menjadi hal yang penting. Sumber daya komputer yang tidak digunakan secara maksimal merupakan kerugian bagi penggunanya.

Berbagai teknik untuk melakukan pemrosesan tugas pada komputer sudah banyak diriset dan diimplementasikan pada *Operating System*. Diantaranya adalah *First Come First Serve (FCFS), Shortest Job First Scheduling, Priority Scheduling, Round Robin Scheduling, Multilevel Queue Scheduling*, dan *Multilevel Feedback-Queue Scheduling*[1]. Teknik-teknik ini selalu bertujuan untuk memaksimalkan penggunaan sumber daya komputer yang tersedia. Namun, berbagai teknik ini tidak mempedulikan depedensi pengerjaan tugas dan hanya merupakan variasi dari antrian tugas.

II. TUJUAN

Paper ini diharapkan memberikan gambaran mengenai cara terbaik menghitung waktu yang dibutuhkan pekerja untuk mengerjakan seluruh pekerjaannya. Perhitungan tersebut dapat digunakan untuk mengestimasi waktu yang dibutuhkan komputer menyelesaikan suatu persoalan, dan menentukan waktu rilis produk. Data dapat digunakan untuk melakukan analisis lebih lanjut terhadap projek tersebut.

III. DASAR TEORI

A. *Greedy Algorithm*

Algortima Greedy merupakan algoritma yang sangat simpel. Algoritma Greedy hanya mempedulikan pilihan terbaik saat ini dan tidak mempedulikan efek pilihan ini pada pilihan selanjutnya. Banyak masalah yang tidak dapat diselesaikan menggunakan algoritma greedy.[2]

Algoritma Greedy memiliki lima bagian [3]

1. Himpunan Kandidat (C)
2. Himpunan Solusi (S)
3. Fungsi seleksi (SELEKSI)
4. Fungsi kelayakan (LAYAK)
5. Fungsi objektif (OBJ)

Struktur program greedy adalah sebagai berikut

- Himpunan solusi kosong
- Dalam tiap langkah
    - Kandidat Solusi-solusi, (C), akan dicari menggunakan fungsi seleksi (SELEKSI)
    - Jika solusi tersebut layak (LAYAK) maka solusi akan dimasukan kedalam himpunan solusi (S)
    - Jika solusi-solusi sudah mencapai fungsi objectif (OBJ) maka algoritma selesai dan solusi tersebut merupakan jawabannya. [3]

Greedy sering digunakan untuk beberapa problem yang sederhana seperti persoalan penukaran uang. Pada problem ini, jumlah koin yang dikembalikan ingin diminimasi. [3]

TABLE I. TABEL FUNGSI-FUNGSI GREEDY PADA PERSOALAN PENUKARAN UANG

Fungsi	***Deskripsi***
Himpunan Kandidat	1,5,10,20,50
Fungsi seleksi	pilih koin dengan nilai tertinggi dibawah uang yang harus dikembalikan
Fungsi kelayakan	memeriksa apakah solusi melebihi jumlah uang yang diminta.
Fungsi Objectif	jumlah koin yang digunakan minimum

Kembalian yang harus diberikan adalah **69**.

TABLE II. TABEL TAHAP-TAHAP PENYELESAIAN PERSOALAN PENUKARAN UANG DENGAN ALGORITMA GREEDY

Permulaan	S = {}
Tahap 1	C = {1,5,10,20,50} x = SELEKSI(C) x = {50} Apakah {50} layak ? Ya, S = {50} Apakah S Solusi ? Tidak
Tahap 2	C = {1,5,10,20,50} x = SELEKSI(C) x = {10} Apakah {50,10} layak ? Ya, S = {50,10} Apakah S Solusi ? Tidak
Tahap 3	C = {1,5,10,20,50} x = SELEKSI(C) x = {5} Apakah {50,10,5} layak ? Ya, S = {50,10,5}
	Apakah S Solusi ? Tidak
…	…
Tahap 7	C = {1,5,10,20,50} S = {50,10,5,1,1,1} Apakah {50,10,5,1,1,1,1} layak? Ya, S = {50,10,5,1,1,1,1} Apakah S Solusi ? Ya

Program berakhir dan solusi didapatkan yaitu **{50,10,5,1,1,1}**

B. Scheduling Algorithm

Algoritma penjadwalan yang umum dilakukan pada operating sistem. Algoritma-algoritma ini hanya menggunakan sebuah CPU dengan jumlah proses yang minimal sedikit. Penjelasan dibawah ini hanya untuk menampilkan berbagai proses penjadwalan pada *operating system*. Berikut cara-cara penjadwalan tersebut,

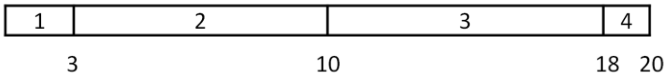
1) First-Come First-Serve

First-Come First-Serve, FCFS, seperti namanya merupakan algoritma penjadwalan dengan memroses tugas berdasarkan waktu tugas itu datang. FCFS memanfaatkan struktur data antrian, *queue*, dengan properti FIFO, *First In First Out*. Tugas baru akan dimasukan kebelakang antrian dan tugas paling depan akan diproses secara berurutan. [2]

Berikut contoh FCFS,

TABLE III. TABEL CONTOH ANTRIAN PROSES

Proses	***Waktu***
1	3
2	7
3	8
4	2



3 10 18 20

Gambar. 2.Illustrasi pemrosesan tugas dengan FCFS [4]

Waktu rata-rata penundaan tugas adalah $\frac{0+3+10+18}{4} = 7.75$

2) Shortest-Job-First Schedule

Shortest-Job-First, SJF, seperti namanya merupakan algoritma penjadwalan dengan memroses tugas dengan waktu paling cepat diselesaikan terlebih dahulu. Cara ini merupakan cara dengan waktu menunggu rata-rata minimal. SJF menganjurkan agar tugas-tugas dipecah-pecah menjadi bagian yang lebih kecil. SJF juga dapat menyebabkan terjadinya *starvation*. Tugas yang membutuhkan waktu lama akan menunggu terus-menerus jika tugas dengan waktu relatif cepat datang. [2]

Berikut contoh SJF,

TABLE IV. TABEL CONTOH ANTRIAN PROSES

Proses	***Waktu***
1	5
2	2
3	10
4	8

Gambar. 3.Illustrasi pemrosesan tugas dengan SJF [4]

Waktu rata-rata penundaan tugas adalah $\frac{0+2+7+15}{4} = 6$

3) Priority Scheduling

Priority Scheduling adalah algoritma penjadwalan dengan memilih tugas dengan prioritas tertinggi. Prioritas tertinggi dapat merupakan angka terkecil maupun angka terbesar. Prioritas tugas biasanya merupakan bilangan bulat dengan range tertentu. [2]

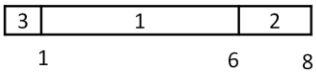
Prioritas tugas dapat ditentukan saat tugas diberikan atau dapat ditentukan secara internal sesuai dengan aturan masing-masing sistem. *Priority Scheduling* juga memiliki masalah yang sama dengan SJF yaitu *starvation.* Tugas dengan prioritas rendah dapat menunggu terus menerus jika tugas dengan prioritas tinggi terus berdatangngan.

Berikut Contoh *Priority Scheduling*,

TABLE V. TABEL CONTOH ANTRIAN PROSES

Proses	***Waktu***	***Prioritas***
1	5	1
2	3	2
3	1	0

*Sistem pada contoh ini memprioritaskan tugas dengan angka prioritas paling rendah



Gambar. 4.Illustrasi pemrosesan tugas dengan Priority Scheduling [4]

Rata-rata waktu menunggu tidak relevan dihitung pada pemrosesan tugas ini karena tidak bergantung terhadap algoritma namun tergantung pada prioritas.

4) Round Robin Scheduling

Round Robin Scheduling, RR, memiliki banyak kemiripan dengan FCFS. Namun, berbeda dengan FCFS pemrosesan tugas disegmentasi menjadi tugas-tugas kecil dengan waktu yang sama. RR menghindari terjadinya *starvation* karena seluruh proses dalam antrian diproses secara berkala. Waktu menunggu RR tidak optimal.

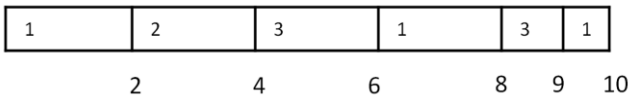
Pada sistem komputer nyata pemrosesan dengan cara *RR* menambahkan proses baru yaitu menganti konteks pekerjaan dari satu tugas ke tugas berikutnya. Hal ini menyebabkan pemilihan waktu pemrosesan sangat penting pada algoritma RR.

Berikut Contoh RR,

TABLE VI. TABEL CONTOH ANTRIAN PROSES

Proses	***Waktu***
1	5
2	2
3	3

*pada sistem ini waktu pemrosesan dibatasi 2 satuan waktu



Gambar. 5.Illustrasi pemrosesan tugas dengan Round Robin [4]

5) Multilevel Queue Scheduling

Multilevel Queue Scheduling, MQ, dilakukan ketika proses dapat dikategorisasi. Seperti namanya, Antrian tugas bersifat multilevel. Tugas akan diurutkan berdasarkan Antrian teratas hingga antrian-antrian dibawahnya. Misalnya, Tugas dikategorisasi berdasarkan prioritas lalu FCFS. Tugas dengan prioritas rendah tidak akan diproses sebelum semua proses dengan prioritas tinggi telah selesai. Pemrosesan tugas pada tiap prioritas menggunakan FCFS. Tugas tidak dapat berpindah antrian ketika sudah ditentukan.

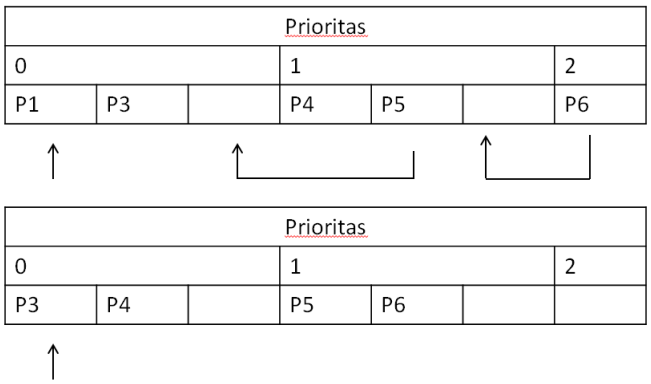
TABLE VII. ILLUSTRASI MULTILEVEL QUEUE SCHEDULING DENGAN PRIORITY SCHEDULING DAN FCFS

Prioritas						
0			1			2
P1	P3		P4	P5		P6

6) Multilevel Feedback-Queue Scheduling

Multilevel Feedback-Queue Scheduling, MFQ, sangat mirip dengan MQ. Perbedaan diantara keduanya hanya terletak pada tugas yang dapat dipindahkan dari satu antrian ke antrian lain pada MFQ. Pemindahan suatu tugas dari satu antrian ke antrian lainnya untuk menghindari terjadinya *starvation.* Ketika tugas sudah lama tidak diproses maka tugas tersebut sebaiknya ditingkatkan prioritasnya.

MFQ bersifat lebih fleksibel dari MQ karena dapat beradapatasi dengan situasi apapun. Tetapi, MFQ mempunyai kelembahan yaitu sulit untuk diimplementasikan.



Gambar. 6.Illustrasi Multilevel Feedback-Queue Scheduling [4]

IV. PERSOALAN

A. Deskripsi Persoalan

Terdapat sekumpulan tugas-tugas dengan waktu pengerjaan yang dibutuhkan masing-masing tugas. Beberapa tugas tersebut memiliki depedensi pengerjaan. Misalnya,

Tugas A membutuhkan hasil dari tugas B dan C untuk bisa dikerjakan. Satu tugas hanya dapat dikerjakan oleh tepat satu pekerja. Pekerja dalam persoalan ini bersifat homogen. Tugas membutuhkan waktu yang sama siapapun yang mengerjakannya. Pekerja yang sedang mengerjakan tugas tidak dapat meninggalkan tugasnya dan menerima tugas baru. Jumlah pekerja terbatas pada tepat n orang. Pekerja tidak harus bekerja setiap saat.

Hasil keluaran yang diharapkan adalah

- Waktu total pengerjaan tugas
- Waktu yang terbuang oleh pekerja yang tidak dapat mengerjakan tugas

Algoritma yang dinginkan adalah algoritma dengan waktu total pengerjaan tugas minimum dan waktu terbuang minimum. Tentu saja keduanya tidak minimum diposisi yang sama.

Waktu yang dibutuhkan untuk menyelesaikan perhitungan dianggap tidak penting karena algoritma greedy relatif cepat dan kompleksitas tidak berubah antara satu solusi dengan solusi lainnya.

B. *Testcase*

Testcase akan digenerasi secara otomatis dengan komputer. Testcase berupa 50 tugas dengan waktu pengerjaan yang digenerasi menggunakan distribusi uniform dari angka 5 hingga 100 satuan waktu.

Berikut kode C++ untuk mengenerasi waktu pekerjaan

```
std::uniform_int_distribution<int> distribution(5,100);
map<JOB_ID, int> task_time;

for( int id = 1; id <= 50; ++ id ){
    task_time[id] = distribution( generator );
}
```

Depedensi tugas akan digenerasi dengan cara berikut

1. A = { Seluruh ID tugas }
2. L = {}
3. Memilih satu tugas dari A simpan dalam X
4. Menghapus X dari A
5. K = random(angka 0 hingga max_depedencies)
6. Memilih K tugas dari A
7. Memamsukan tugas-tugas tersebut dalam L dengan format X membutuhkan T dengan T adalah tugas-tugas tersebut.
8. Mengulangi langkah 3 hingga 7 hingga seluruh A habis

Berikut Kode C++ untuk mengenrasi depedensi tersebut

```
default_random_engine generator;
uniform_int_distribution<int> task_depedencies(0,max_depedencies);

vector<JOB_ID> A;
// step 1: A = { Seluruh ID tugas }
for( auto it : task_time )
    A.push_back( it->first );

// step 2: L = {}
vector< pair<JOB_ID,JOB_ID> > L;
do {
    // step 3: Memilih satu tugas dari A simpan dalam X
    uniform_int_distribution<int> pick_id(0, ids.size()-1);
    int idx = pick_id( generator );
    int X = ids[idx];
    // step 4: Menghapus X dari A
    ids.erase( ids.begin() + idx );
    // step 5: K = random( angka 0 hingga max_depedencies )
    int K = task_depedencies( generator );
    // step 6: Memilih K tugas dari A
    for( int k = 1; k <= K; ++ k ){
        if( ids.size() == 0 ) break;
        std::uniform_int_distribution<int> pick_id2(0, ids.size()-1);
        // step 7
        int T = ids[pick_id2( generator )];
        L.push_back( make_pair(X,T) );
    }
// step 8
} while( ids.size() != 0 );
```

C. *Format Output*

Output program berupa waktu yang dibutuhkan untuk menyelesaikan seluruh tugasnya dan waktu yang terbuang oleh pekerja yang tidak bekerja karena tidak ada pekerjaan yang bisa dilakukan kedua output ini akan diplot pada grafik delta time dan delta loss.

Output program akan digenerasi untuk 1 hingga n pekerja dan 0 hingga n depedensi tugas, dimana n adalah jumlah tugas. Percobaan akan diulang 10 kali dan dicari rata-ratanya.

Berikut Pseudocode generasi Output

```
generasi 50 tugas dengan waktu pengerjaan 5 hingga 100

for( worker = 1 to 50 )
    for( max_depedencies = 0 to 50 )
        for( x = 1 to 10 )
            generasi depedensi tugas dengan max_depedencies
            lakukan kalkulasi dengan algoritma 1
                dengan n worker
            lakukan kalkulasi dengan algoritma 2
                dengan n worker

        hitung rata-rata 10 kali percobaan kedua algoritma

        bandingkan hasil percobaan dengan
            menghitung selisihnya
```

Untuk melihat lebih jelas lagi maka digambarkan pula selisih dari hasil plot ketiga solusi.

V. Proposal Solusi

A. *Solusi 1*

Solusi pertama adalah memilih tugas dengan menggunakan algortima *First-Come First-Serve*, FCFS. Tugas-tugas yang layak akan diproses sesuai waktu kedatangannya.

B. *Solusi 2*

Solusi kedua adalah memilih tugas dengan menggunakan algoritma scheduling *Shortest-Job-First*, SJF. Dari seranai tugas, ambil tugas yang dapat dikerjakan dengan waktu pengerjaan minimal.

C. *Solusi 3*

Solusi ketiga adalah memilih tugas dengan *Multilevel Queue*. Pada level pertama antrian akan didasarkan pada prioritas. Pada level kedua antrian akan didasarkan waktu pengerjaan minimal. Prioritas tugas akan didasarkan pada jumlah tugas yang bergantung pada tugas tersebut. Semakin banyak tugas yang bergantung pada tugas tersebut semakin tinggi prioritas tugas.

VI. IMPLEMENTASI

Implementasi algoritma Greedy menggunakan empat bagian ini :

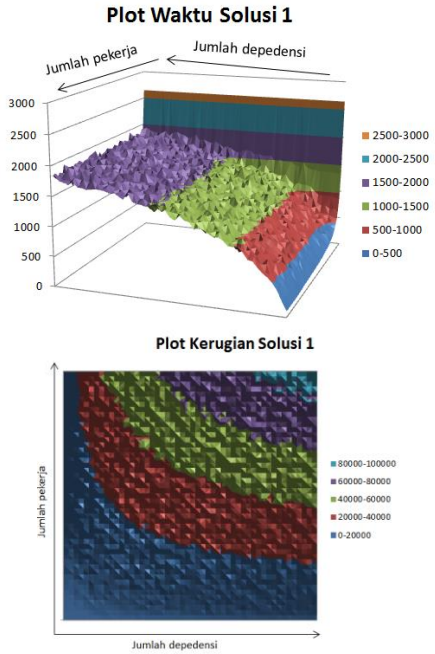
TABLE VIII. TABEL CONTOH ANTRIAN PROSES

Fungsi	***Deskripsi***
Himpunan Kandidat	Seluruh ID Job
Fungsi seleksi	Pilih job yang dapat dikerjakan yang belum dikerjakan berdasarkan algoritma pemilihan tugas
Fungsi kelayakan	-
Fungsi Objectif	Waktu yang dibutuhkan minimum

Implementasi algoritma greedy akan dilakukan menggunakan Bahasa C++ karena ketersedian STL yang sangat mempermudah implementasi. Algoritma greedy hanya akan diimplementasikan sekali dengan fungsi seleksi yang diubah-ubah untuk tiap solusinya.

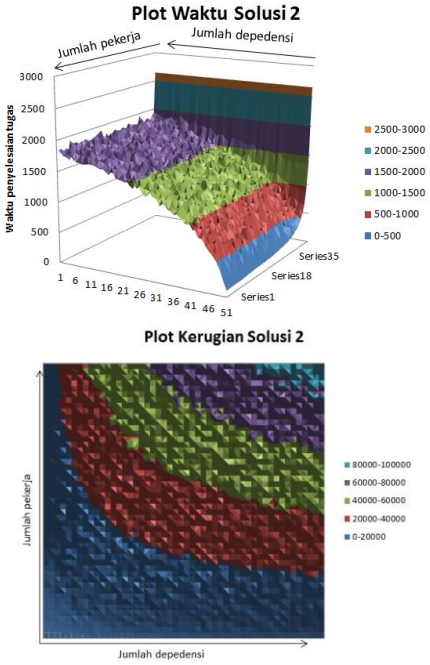
VII. HASIL EKSPERIMEN

A. *Hasil Eksperimen Solusi 1*



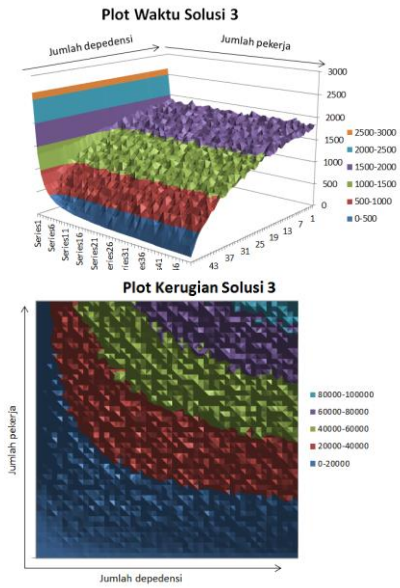
Jumlah pekerja yang tinggi meningkatkan waktu pengerjaan secara signifikan hingga ke titik tertentu yakni 5 pekerja. Jumlah depedensi tugas yang tinggi sangat mempengaruhi waktu pengerjaan. Kerugian waktu sangat rendah dengan jumlah pekerja yang rendah. Kerugian waktu dipengaruhi secara signifikan dengan meningkatnya depedensi tugas pada jumlah pekerja yang tinggi.

B. *Hasil Eksperimen Solusi 2*



Jumlah pekerja yang tinggi meningkatkan waktu pengerjaan secara signifikan hingga ke titik tertentu yakni 5 pekerja. Jumlah depedensi tugas yang tinggi sangat mempengaruhi waktu pengerjaan. Kerugian waktu sangat rendah dengan jumlah pekerja yang rendah. Kerugian waktu dipengaruhi secara signifikan dengan meningkatnya depedensi tugas pada jumlah pekerja yang tinggi.

C. *Hasil Eksperimen Solusi 3*

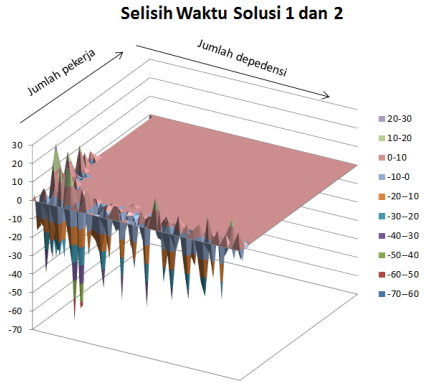


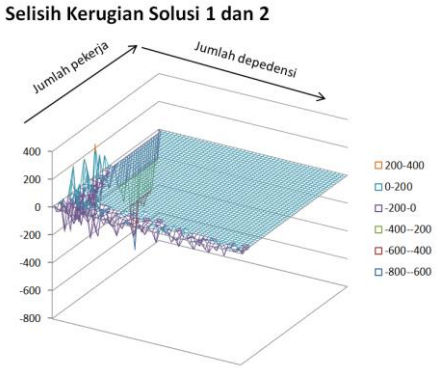
Jumlah pekerja yang tinggi meningkatkan waktu pengerjaan secara signifikan hingga ke titik tertentu yakni 5 pekerja. Jumlah depedensi tugas yang tinggi sangat mempengaruhi waktu pengerjaan. Kerugian waktu sangat rendah dengan jumlah pekerja yang rendah. Kerugian waktu

dipengaruhi secara signifikan dengan meningkatnya depedensi tugas pada jumlah pekerja yang tinggi.

D. Selisih Hasil Eksperimen Solusi 1 dan 2

Selisih = hasil 1 – hasil 2

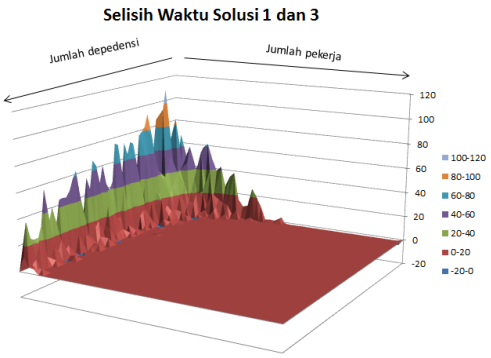


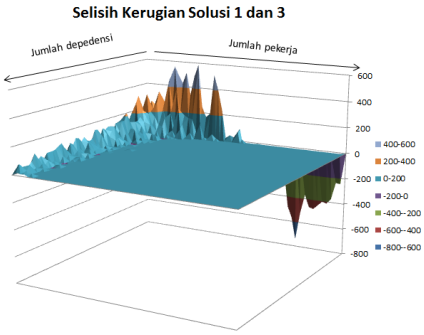


Solusi 1 dibanding Solusi 2 menghasilkan selisih yang sulit untuk dilihat. Pada kasus jumlah depedensi tepat 0, Solusi 1 jauh lebih baik dibanding solusi 2. Pada kasus Jumlah pekerja banyak, solusi 1 dan 2 tidak memiliki perbedaan yang signifikan. Pada kasus jumlah pekerja sedikit, kedua solusi bertukaran posisi.

E. Selisih Hasil Eksperimen Solusi 1 dan 3

Selisih = hasil 1 – hasil 3

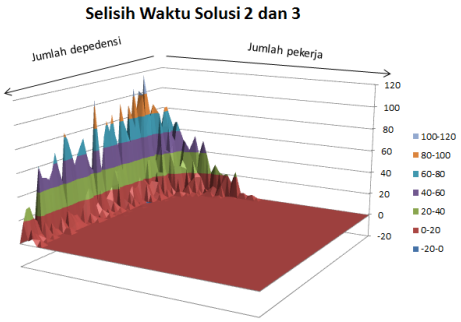


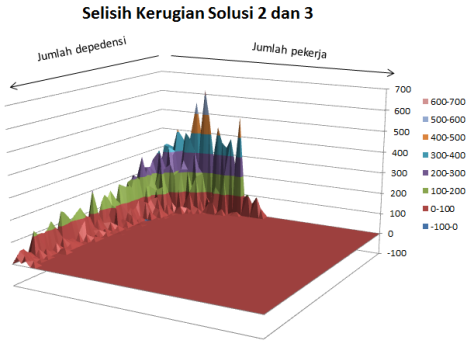


Pada kasus depedensi tepat 0 solusi 1 lebih baik. Pada kasus depedensi rendah solusi 3 jauh lebih baik dari solusi 1. Pada kasus pekerja rendah solusi 3 jauh lebih baik dari solusi 1.

F. Selisih Hasil Eksperimen Solusi 2 dan 3

Selisih = hasil 2 – hasil 3





Pada dominan kasus solusi 3 lebih baik dari solusi 2. Solusi 3 dan 2 tidak jauh berbeda pada kasus jumlah pekerja tinggi dan jumlah depedensi tinggi. Solusi 3 jauh lebih baik pada kasus depedensi rendah dan jumlah pekerja rendah.

VIII. Analisis Hasil

Berdasarkan hasil plot waktu dan kerugian tidak dapat ditentukan algoritma mana yang lebih baik karena hasil yang sangat mirip satu dengan yang lainnya secara visual. Namun, dapat diketahui bahwa jumlah pekerja sekitar 5 adalah jumlah pekerja optimal untuk pekerjaan ini.

Berdasarkan selisih waktu dan kerugian tentu saja dapat dilihat satu per satu. Solusi 1 dibanding solusi 2 menyatakan bahwa solusi 1 dan 2 sama baiknya. Solusi 1 dibanding solusi 3 menyatakan solusi 3 lebih baik pada kasus selain kasus tugas tanpa depedensi. Solusi 2 dibanding solusi 3 menyatakan

solusi 3 lebih baik dari solusi 2. Dari ketiga hasil tersebut dapat dinyatakan bahwa Solusi 3 adalah solusi yang terbaik secara keseluruhan dan Solusi 1 adalah solusi yang terbaik jika tidak ada depedensi tugas.

IX. Simpulan

Solusi 3 adalah solusi yang terbaik secara keseluruhan dan Solusi 1 adalah solusi yang terbaik jika tidak ada depedensi tugas.

Acknowledgment

Candra Ramsi, sebagai penulis ingin menyatakan ucapan terima kasih yang sebesar-besarnya kepada Dr. Nur Ulfa Maulidevi, S.T., M.Sc dan Dr.Ir. Rinaldi Munir, M.T. atau bimbingannya sehingga penulis dapat mengerti konsep-konsep pada kuliah Strategi Algoritma. Terima kasih bagi seluruh orang yang telah memberikan dukungan kepada penulis dalam proses penulisan makalah ini.

References

[1] Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, "Operating System Concepts, Eighth Edition ", Chapter 5

[2] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Greedy/greedyIntro.htm diakses pada 5 Mei 2016

[3] Slide Algorithma Greedy, Rinaldi Munir.

[4] Sumber Pribadi

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 7 Mei 2016

Candra Ramsi - 13514090