LAPORAN TUGAS BESAR 1

IF3170 INTELIGENSI BUATAN

Aplikasi Penjadwalan Kelas dengan Constraint Satisfaction Problem Local Search

Johan Sentosa - 13514026

Yeksadiningrat Al Valentino - 13514055

Christian Anthony Setiawan - 13514085

Kevin Supendi - 13514094

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2016

BAB I

DESKRIPSI MASALAH

Masalah penjadwalan sangat sering dijumpai dalam kehidupan sehari-hari. Dalam penjadwalan, hal yang utama adalah bagaimana waktu dan *resource* yang dimiliki dapat dimanfaatkan sebaik-baiknya. Karena itu program ini dibuat untuk mencari solusi yang efektif dan efisien.

Masalah penjadwalan yang diselesaikan program ini adalah penjadwalan ruang kelas, bagaimana mengatur waktu kuliah agar ruangan tersedia dan tidak dipakai oleh kuliah lain. Spesifikasi masalah penjadwalan kuliah akan dijelaskan pada bagian selanjutnya.

**Jadwal**

Jadwal yang akan digunakan dalam tugas memiliki spesifikasi sebagai berikut :

1. Jadwal adalah sebuah nama kegiatan, waktu, dan tempat.
2. Setiap jadwal memiliki durasi tertentu.
3. Terdapat beberapa jadwal yang ruangannya ditentukan dan beberapa jadwal yang ruangannya bebas.
4. Pencarian dilakukan dengan mencocokkan jadwal terhadap waktu dan ruangan yang tepat.

**Local Search**

Dalam pengerjaan tugas ini terdapat beberapa algoritma yang perlu diterapkan

1. Hill Climbing Search
2. Simulated Annealing
3. Genetic Algorithm

Setiap hasil program penjadwalan program akan dibandingkan dan dihitung berdasarkan seberapa baik algoritma dapat menghindari bentrok antar jadwal kegiatan yang ada.

BAB II

DASAR TEORI

1. Algoritma Hill Climbing

Hill Climbing adalah proses pengujian Local Search yang memulai dari state awal, mengubah salah satu elemen solusi sampai ditemukan state yang lebih baik. Hill Climbing mudah diimplementasi, tetapi dapat terjebak dalam Local Maxima.

Metode Hill Climbing pun bervariasi. Simple Hill Climbing memilih node terdekat pertama, Steepest Ascent membandingkan semua node yang dituju dan memilih yang terdekat dengan solusi. Stochastic memilih node secara random, Random Restart menginisialisasi CSP sehingga tidak terjebak dalam Local Maxima. Pada tugas ini metode yang dipakai adalah Simple Hill Climbing.

Berikut adalah algoritma dari Simple Hill Climbing;

1. Mulai dari keadaan awal, lakukan pengujian: jika merupakan tujuan, maka berhenti; dan jika tidak, lanjutkan dengan keadaan sekarang sebagai keadaan awal.
2. Kerjakan langkah-langkah berikut sampai solusinya ditemukan, atau sampai tidak ada operator baru yang akan diaplikasikan pada keadaan sekarang:

* Cari operator yang belum pernah digunakan; gunakan operator ini untuk mendapatkan keadaan yang baru.
* Lihat keadaan baru tersebut. Jika keadaan baru merupakan tujuan, program berhenti. Jika bukan tujuan, namun nilainya lebih baik daripada keadaan sekarang, maka jadikan keadaan baru tersebut menjadi keadaan sekarang. Jika keadaan baru tidak lebih baik daripada keadaan sekarang, maka lanjutkan iterasi.

1. Algoritma Simulated Annealing

Algoritma Simulated Annealing adalah algoritma Local Search yang menggunakan probabilitas untuk memilih node. Algoritma ini relatif lebih baik dari Hill Climbing karena tidak terjebak Local Maxima. Algoritma Simulated Annealing hampir selalu dapat memberikan solusi optimum jika waktu yang diberikan cukup dan penurunan temperatur tidak terlalu cepat.

Cara kerja Simulated Annealing adalah dengan mengiterasi pencarian node, jika dievaluasi hasilnya lebih baik, maka node itu yang diteruskan, jika lebih buruk, maka node dipilih dengan probabilitas. Probabilitas bergantung dari temperatur, dan temperatur semakin lama semakin berkurang.

1. Pilih nilai temperatur yang tinggi
2. Inisialisasi persoalan secara random
3. Pilih dan evaluasi suatu node, jika lebih baik ambil node itu, jika tidak maka ambil sesuai perhitungan probabilitas, lalu kurangi temperatur
4. Ulangi sampai temperatur = 0
5. Algoritma Genetik

Algoritma Genetik, sesuai dengan namanya, memakai pendekatan menggunakan analogi genetika. Setiap node diumpamakan seperti kromosom, dan diukur menggunakan fitness test. Setiap kali iterasi, akan menghasilkan beberapa kromosom yang disebut generasi. Generasi berikutnya didapat dari hasil persilangan maupun mutasi kromosom yang sudah ada.

Fitness test mengukur apakah kromosom dekat dengan solusi atau tidak.

BAB III

ANALISIS PEMECAHAN MASALAH

Untuk pemilihan domain room, dilakukan seleksi terlebih dahulu. Room yang tidak memiliki setidaknya satu slot sepanjang durationnya, atau tidak memilki hari buka yang sama dengan ketersediaan hari course, tidak akan dimasukan ke domain.

1. Hill Climb

csp.initAllVars();  
 **int** idxConflict = csp.findFirstConflicts();  
 **int** conflict = csp.countConflicts();  
 **int** newConflict ;  
 **int** counter = 0;  
 **while**((conflict>0)&&(counter<1000)){  
 csp.initSingleVar(idxConflict);   
 newConflict = csp.countConflicts();  
 **if**(newConflict < conflict){  
 idxConflict = csp.findFirstConflicts();  
 conflict = newConflict;  
 counter = 0;  
 }  
 **else**{  
 counter++;  
 }  
}  
 **return** csp;

1. Simulated Annealing

**float** SimulatedAnnealing::Temperature(**float** T,**int** count){  
 **return** T \* pow(0.95,(**double**)count);  
};  
  
**float** SimulatedAnnealing::acceptance(**int** delta, **float** temperature){  
 **return** 1/(1+exp(-delta/temperature));  
};  
  
**float** SimulatedAnnealing::rng(){  
 **int** r = rand()%100000;  
 **return** (**float**)r/100000;  
}  
  
SchedCSP\* SimulatedAnnealing::getSolution(){  
 csp.initAllVars();  
 **int** idxConflict = csp.findFirstConflicts();  
 **int** conflict = csp.countConflicts();  
 **int** newConflict ;  
 **int** counter = 0;  
 **int** delta ;  
 **int** T = 1;  
 **while**((conflict>0)&&(counter<1000)){  
 csp.initSingleVar(idxConflict);   
 newConflict = csp.countConflicts();  
 delta = newConflict - conflict;  
 **if**((newConflict < conflict)||(rng()>acceptance(delta,Temperature(T,counter)))){  
 idxConflict = csp.findFirstConflicts();  
 conflict = newConflict;  
 counter = 0;  
 }  
 **else**{  
 counter++;  
 }  
 }  
  
 **return** &csp;  
   
   
};

1. Algoritma Genetik

initializePopulation();  
**for** (**int** i = 0; i < GeneticAlgorithm::GENERATION; ++i)  
{  
 **if** (getLeastConflicts() == 0)  
 {  
 **break**;  
 }  
 **for** (**int** i = 0; i < GeneticAlgorithm::EXTRA\_POPULATION / 2; ++i)  
 {  
 pair<**int**, **int**> p = selectChromosomePair();  
 crossover(p); // every crossover generate 2 pops  
 **double** r;  
 r = (**double**)rand() / RAND\_MAX; // probability type random, 0 sampe 1  
 **if** (r < GeneticAlgorithm::MUTATION\_PROBABILITY) {  
 mutate(p.first);  
 }  
 r = (**double**)rand() / RAND\_MAX; // probability type random, 0 sampe 1  
 **if** (r < GeneticAlgorithm::MUTATION\_PROBABILITY) {  
 mutate(p.second);  
 }  
 }  
 cullPopulation();  
 rerankPopulation();  
}  
applyBestCandidate();

Selection menggunakan random pair dan elite population, crossover menggunakan Uniform crossover, mutate akan memindahkan satu course(jam, hari, kelas) ke random yang lain.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

1. Implementasi

Implementasi program dibagi menjadi beberapa modul yang masing-masing mempunyai fungsi-fungsinya tersendiri.

1. Modul CSP

Modul ini berisi kelas-kelas untuk mengatur CSP, menginisialisasi masalah, dan menampung variabel dan constraint. Modul ini terdiri dari kelas SchedCSP dan kelas SchedVar

1. Modul Algoritma

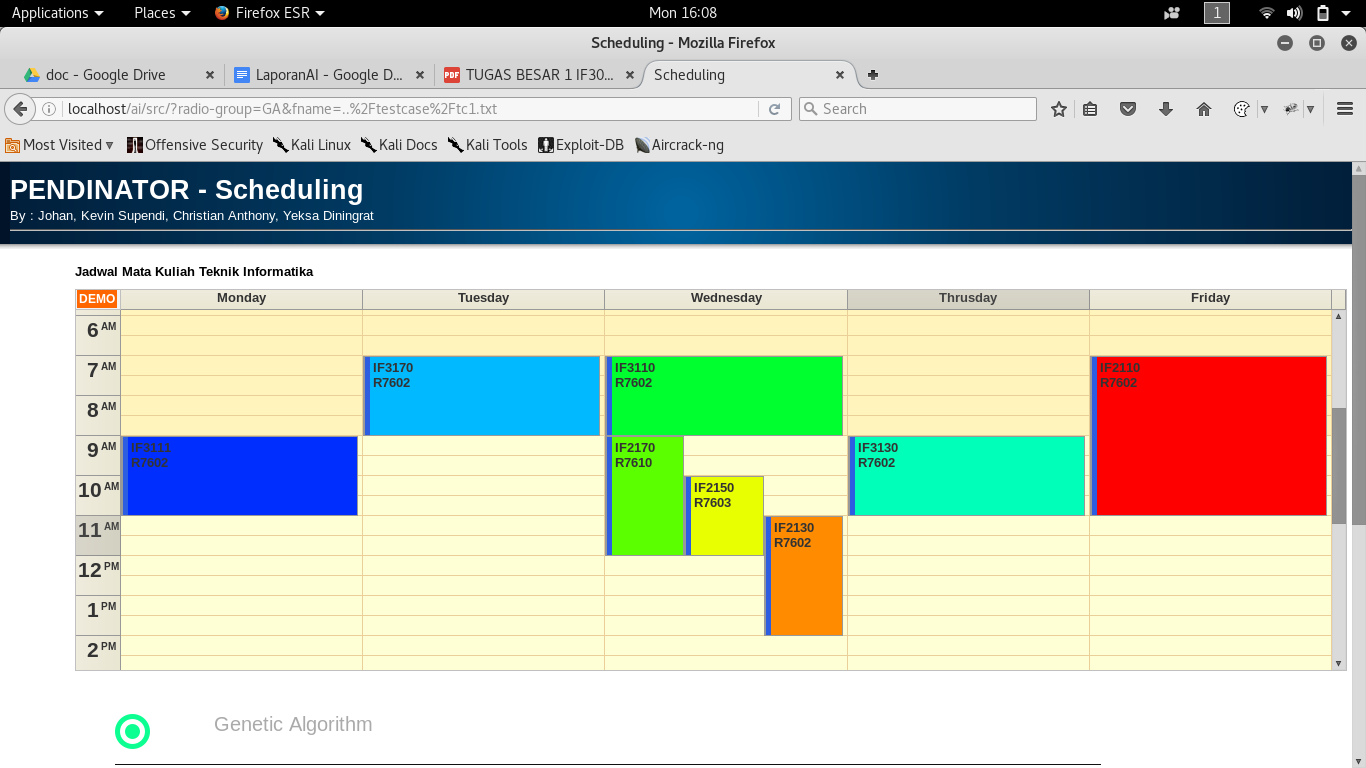
Modul algoritma mengimplementasi 3 algoritma yang digunakan, yaitu GA, SA, dan HC. Hasil pencarian akan disimpan kembali dalam SchedCSP.

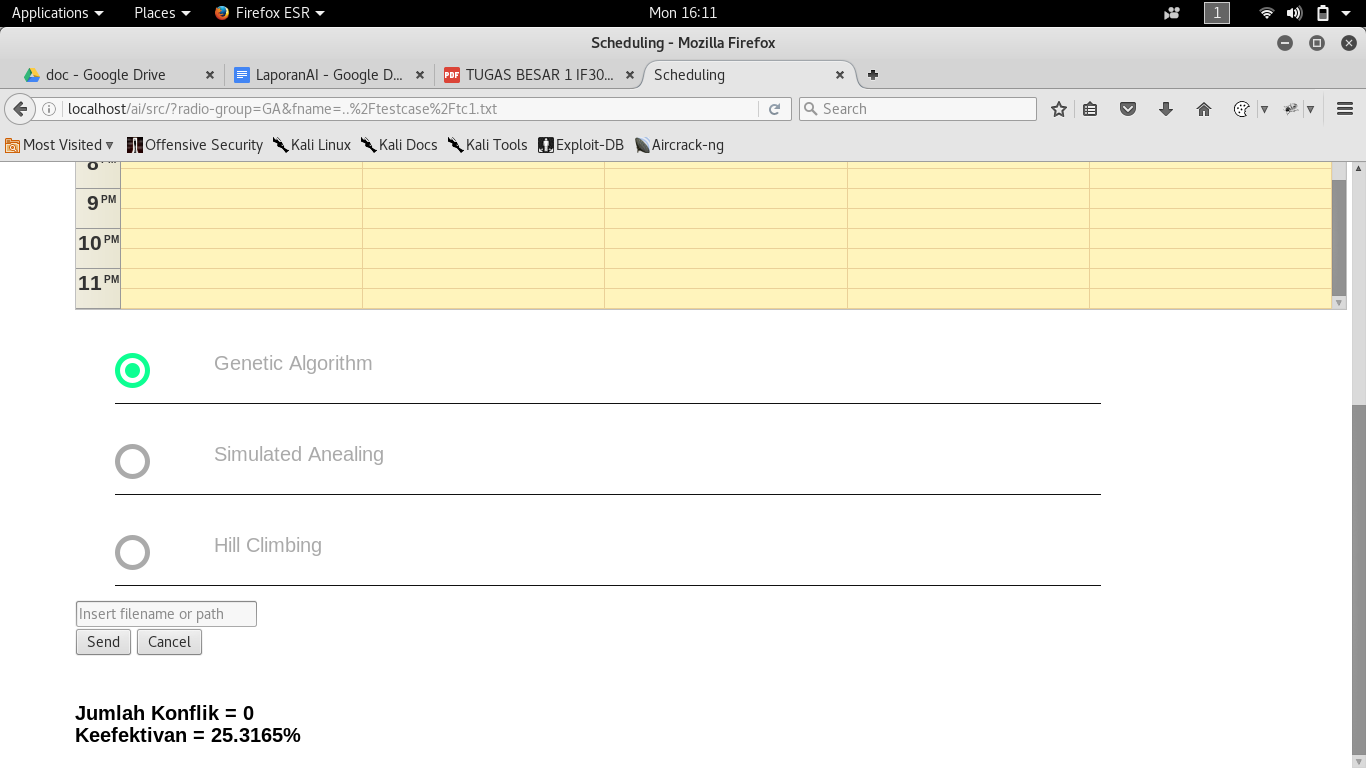
1. Modul GUI

GUI berbasis web menggunakan html dilengkapi php dan javascript. Solusi akan diprint dari SchedCSP dalam bentuk CSV (comma-separated values) yang kemudian akan diparse oleh php dan dimasukkan dalam jadwal. Untuk GUI jadwal menggunakan library *daypilot.js*.

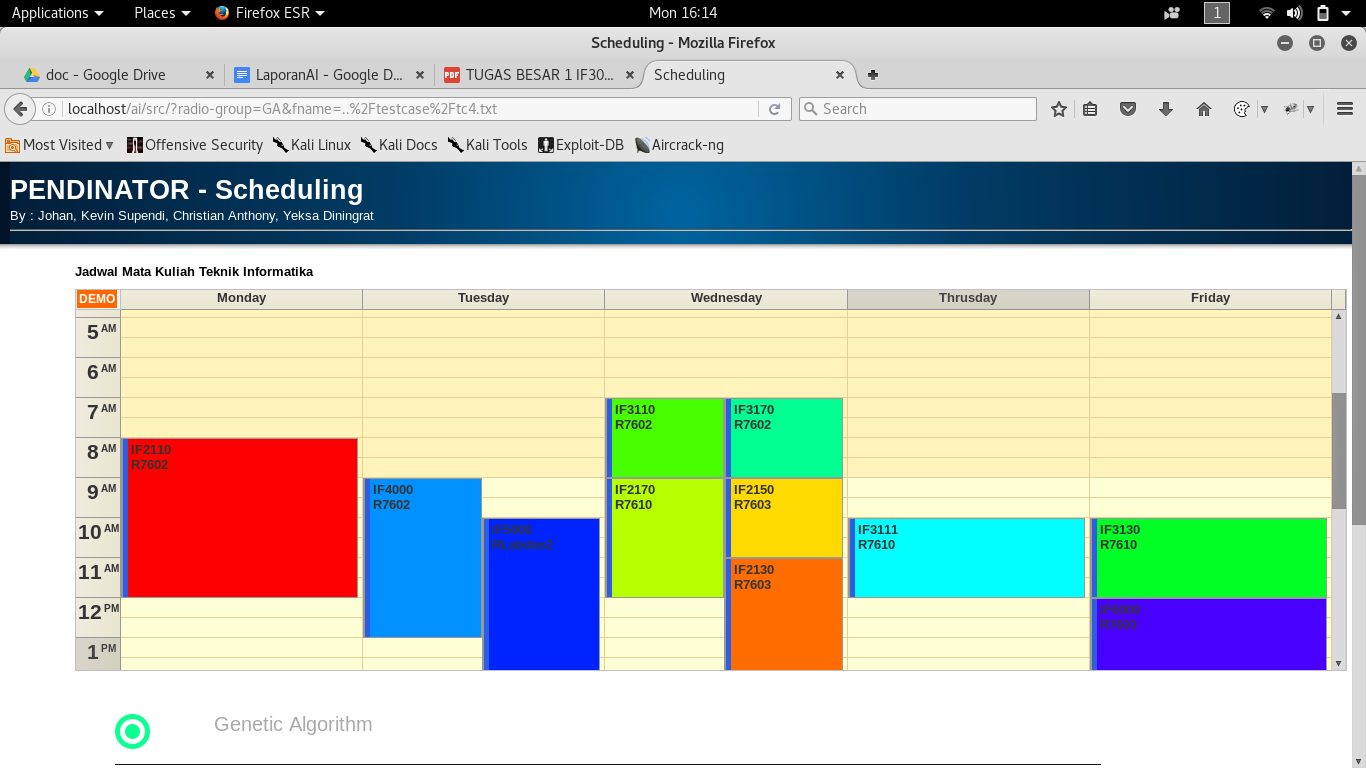
1. Pengujian

Testcase yang diberikan dari deskripsi tugas diuji dan menghasilkan output sebagai berikut :





Kami juga mencoba beberapa testcase lain yang tidak mungkin ada solusi :



BAB V

KESIMPULAN DAN SARAN

Banyak persoalan dapat diselesaikan dengan Problem Solving Agent. Persoalan Scheduling dapat diselesaikan dengan CSP dan hasilnya juga cukup baik.

Hill Climbing efektif untuk testcase yang sederhana, Simulated Annealing cocok untuk persoalan umum tetapi memakan waktu yang lebih lama. Genetic Algorithm juga bagus untuk menelusuri berbagai kemungkinan solusi.