DOKUMENTASI TUGAS BESAR

IF3170 Inteligensi Buatan

Pengaplikasian *Local Search* untuk Penjadwalan Mata Kuliah

Disusun oleh :

Sri Umay Nur'aini Sholihah	13514007
Faza Thirafi	13514033
Ahmad Fajar Prasetiyo	13514053
Cut Meurah Rudi	13514057
Rio Chandra Rajagukguk	13514082

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

TAHUN 2016

A. DESKRIPSI PERSOALAN

Dalam tugas kali ini kami diharuskan untuk membuat sebuah program penjadwalan dengan mengaplikasikan local search. Progam harus dapat menjadwalkan dengan menggunakan tiga algoritma, yaitu :

1. *Hill-climbing Algorithm*,
2. *Simulated Annealing Algorithm*, dan
3. *Genetic Algorithm*.

Jadwal diberikan dalam bentuk file terpisah dengan format tertentu yang meliputi jadwal matakuliah dan jadwal ruang yang dapat digunakan dengan setiap jadwal memiliki *constraint* tertentu (misal harus dijadwalkan di ruang tertentu pada jam tertentu). Penjadwalan dilakukan dengan menempatkan setiap jadwal kuliah pada ruangan yang sesuai jadwalnya sehingga tidak ada jadwal yang tidak terjadwal (tidak ditempatkan di ruang dan waktu yang tepat) dan tidak ada jadwal yang bentrok dengan jadwal lain.

Program harus menampilkan komponen berikut secara eksplisit :

1. Total jadwal yang terbentuk
2. Total jadwal yang bentrok
3. Presentase keefektifan pengunaan ruangan

Kriteria keberhasilan program adalah apabila program dapat menjadwalkan seluruh jadwal secara efektif (semua jadwal terjadwalkan, tidak ada jadwal yang bentrok, dan jadwal sesuai *constraint*).

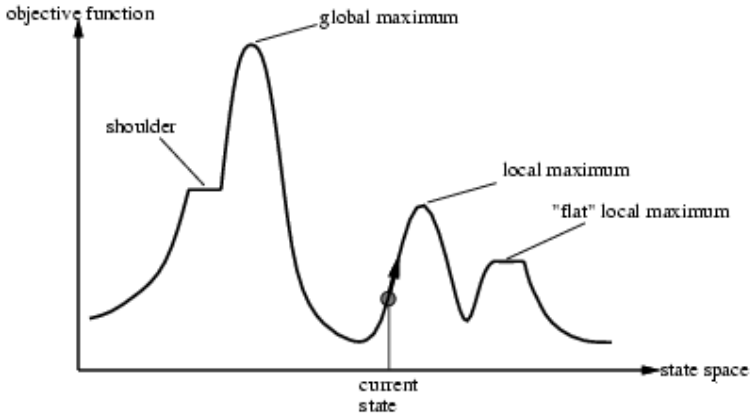
B. TEORI DASAR

Local search adalah salah satu algoritma yang dapat digunakan untuk menyelesaikan permasalahan yang solusinya tidak mementingkan *path* untuk menuju goal, contohnya *pure optimization*. Pendekatan yang dilakukan oleh algoritma *local search* adalah *incremental*, yaitu pendekatan dari keadaan *state* yang *complete*, semua varibel telah di*assign* dengan nilai tertentu (misalnya secara *random*), kemudian dilakukan pengubahan pada variabel yang tidak memenuhi *constraint* sampai keadaan *state*

konsisten, yaitu semua varibel memenuhi *constraint*. Berikut akan dijelaskan varian algoritma yang termasuk dalam *local search* yang digunakan pada tugas ini.

1. *Hill-climbing Algorithm*

   Algoritma *hill-climbing,* disebut juga *greedy local search* merupakan salah satu algoritma varian *local seach.* Disebut *greedy local search* karena algoritma ini memakai prinsip *greedy*, yaitu langkah yang diambil pada proses penyelesaian pada suatu tahap harus lebih ‘baik’ dari tahap sebelumnya. Kekurangan algoritma ini adalah dapat terjebak di *local maxima,* yaitu keadaan di mana proses penyelesaian mencapai suatu *state* yang merupakan ‘kondisi terbaik’ dibanding semua *state* selanjutnya yang bisa di tuju dari *state* tersebut meskipun *state* tersebut sebenarnya bukanlah solusi.



2. *Simulated Annealing Algorithm*

   Berbeda dengan *Hill-Climbing* yang selalu mencari langkah yang menuju *state* yang paling ‘menjanjikan’, *simulated annealing* menggunakan probabilitas untuk menentukan langkah selanjutnya jika hasil evaluasi *next step* lebih rendah dari sebelumnya. Pada *simulated annealing* terdapat varibel T (Temperatur) yang mempengaruhi probabilitas pengambilan keputusan yang nilainya di*assign* dan nilainya akan berkurang sesuai dengan fungsi waktu. Ide utama dari algoritma ini adalah untuk mengurangi kemungkinan terjebak pada *local maxima* seperti yang dialami algoritma *hill-climbing* dengan mengizinkan algoritma memilih langkah yang ‘buruk’, namun diharapkan akan menuju solusi, berdasarkan perhitungan probabilitas fungsi penerimaan (*Acceptance Probability Function*) yang besarnya:

$$P = \exp\left(\frac{-\Delta E}{k_b T}\right) = e^{\left(-\Delta E / k_b T\right)}$$

dengan ΔE adalah selisih nilai evaluasi dari state awal dengan state calon suksesor, kb adalah konstanta Boltzmann dan T adalah Temperatur pada state awal. Langkah 'buruk' ini hanya diambli pada tahap-tahap awal sebab varibel T yang terus berkurang mengikuti fungsi waktu sangat mempengaruhi probabilitas dari *state* yang akan dipilih selanjutnya, semakin kecil nilai T atau semakin mendekati solusi, maka algoritma *simulated annealing* ini akan menjadi semakin *greedy*.

3. *Genetic Algorithm*

   Jika pada dua algoritma sebelumnya setiap langkah menuju suatu *state* merupakan langka dari sebuah *state* sebelumnya, maka pada *genetic algorithm* state selanjutnya pada suatu langkah terbentuk dari kombinasi dua buah *state* lain. Pada algoritma ini pada awalnya akan di*generate* n buah *state* yang disebut populasi, setiap satu *state* populasi tersebut disebut dengan individu, individu-individu inilah yang akan berpasangan (dikombinasikan) unutk membentuk sebuah *state* baru. Langkah penyelesain dengan *genetic algorithm* ini adalah sebagai berikut.

   1. Men-*generate* n buah state (populasi).
   2. Melakukan penghitungan keadaan setiap *state* dengan *fitness function. Fitness function* ini merupakan fungsi unutk mengetahui seberapa 'baik' keadaan suatu *state*, semakin tinggi nilainya berdasarkan *fitness function* maka keadaannya semakin baik.
   3. Melakukan pemilihan (*selection*) terhadap individu-individu untuk selanjutnya dilakukan proses reproduksi individu (*state*) baru. Keterpilihan individu pada tahap ini dipengaruhi dipengaruhi oleh nilai *fitnes function*nya, semakin tinggi nilai *fitness function*nya makan semakin besar kemungkinan suatu individu terpilih pada tahap ini. Namun nilai dari fitnes function ini sifatnya hanya mempengaruhi (kemungkinannya), bukan menentukan (misalnya dipilih yang paling tinggi nilainya) seperti pada algoritma *greedy*.

4. Melakukan reproduksi individu (*state*) baru dari individu yang telah terpilih pada tahap sebelumnya. Reproduksi dialakukan dengan cara *crossover*, yaitu dua individu masing-masing dipecah menjadi dua bagian, dan satu bagian dari individu satu digabungkan dengan satu bagian dari individu lainnya sehingga terbentuk individu baru yang berbeda. Selain reproduksi juga dapat dilakukan mutasi, yaitu dengan mengubah keadaan individu (*state*) pada bagian (variabel) tertentu secara acak.
5. Selanjutnya kembali ke tahap nomor 2 sampai solusi ditemukan.

C. IMPLEMENTASI

1. Hill-climbing Algorithm

```
function HILL-CLIMBING(problem) return local maximum
   current <- MAKE-NODE(problem,INITIAL-STATE)
   loop do
        neighbor, a highest value successor of current
        if neighbor.VALUE < current.VALUE then return
   current.STATE
        current <- neighbor
```

Implementasi hill climbing dimulai dari MAKE-NODE yang dalam java diimplementasikan sebagai method yang menerima input yang dibaca dari test case. Kemudian method MAKE-NODE akan memasukkan seluruh course kedalam objek schedule secara acak. Hill climbing dimulai ketika memasuki proses looping, pada proses looping pertama-tama akan dicari hari dan jam pada objek schedule yang memiliki konflik terbanyak. Pada hill climbing yang diimplementasikan ini, VALUE pada fungsi diatas direpresentasikan sebagai jumlah konflik yang terjadi pada pasangan hari dan jam di objek schedule. Setelah ditemukan tuple nilai hari, jam dan objek schedule yang memiliki nilai konflik yang terbesar, dilakukan pemindahan pada course tersebut kelokasi yang memiliki kepadatan jumlah course paling sedikit. Kepadatan

jumlah course direpresentasikan sebagai jumlah course yang terdapat pada suatu hari pada beberapa jam yang berurutan, disini digunakan jumlah sks untuk menentukan jumlah waktu yang dicari kepadatan course-nya. Proses looping ini terus dilakukan sampai jumlah konflik tidak lagi membaik.

2. Simulated Annealing Algorithm

   Pada program yang dibuat, algoritma *Simulated Annealing* diimplementasikan sebagai berikut. Jadi, untuk Temperatur (T) yang menjadi parameter jumlah langkah (*step*) diinisialisasi dengan nilai awal (dalam program 100) dengan nilai pengurangan T setiap *step* sebesar 1 (dalam hal ini, pengurangan temperatur bersifat linear, bukan eksponensial). Untuk bagian evaluasi per langkah, yang menjadi parameter evaluasi adalah jumlah konflik pada keadaan sementara jadwal.

   Untuk langkah-langkah algoritmanya, semua parameter diinisialisasi terlebih dahulu. Lalu, masuk ke *loop* yang akan mencari solusi terbaik. Loop akan berhenti jika nilai T sudah 0. Pada awal langkah, dicari suksesor dari *state* jadwal. Untuk menentukannya, mirip dengan yang dilakukan pada algoritma *Hill-Climbing*. Jika hasil evaluasi calon suksesor state yang didapat lebih besar dari evaluasi state sementara, maka langkah dilanjutkan. Namun, jika hasil evaluasi calon suksesor lebih kecil (lebih buruk), maka untuk menentukan apakah pencarian solusi dilanjutkan atau tidak, nilai *Acceptance Probability Function* (P) dihitung juga. Nilai P ini akan dibandingkan dengan sebuah nilai random yang dibangkitkan dengan rentang 0.0 ~ 1.0. Jika probabilitas P lebih besar dari nilai random, pencarian dilanjutkan. Namun jika lebih kecil, maka pencarian diberhentikan. Begitu seterusnya hingga salah satu kondisi pemberhentian loop terpenuhi.

3. Genetic Algorithm

   Dalam pembuatan program penjadwalan dengan *Genetic Algorithm*, mula-mula program akan men*generate* 70 jadwal *complete* (disebut juga dengan individu) secara acak, kemudian jadwal direpresentasikan dalam tipe data *string* untuk memudahkan melakukan operasi yang ada dalam algoritma. Individu yang sudah dirubah dalam repepresentasi *string* tersebut akan dioperasikan satu sama lain untuk mengahasilkan individu baru dengan melakukan proses reproduksi dengan cara *cross-over*. Sebelum melakukan operasi reproduksi antar individu dengan *cross-over*, program akan menghitung ‘kualitas’ suatu individu dengan *fitness function*.

Nilai dari fitness function mempengaruhi keterpilihan individu pada setiap tahap. Semakin besar nilai *fitness function*nya, semakin besar kemungkinan individu untuk terus bertahan hidup (terpilih di setiap tahap). Sebaliknya semakin sedikit nilai *finess function*nya semakin kecil kemungkinan suatu individu unutk terpilih (bertahan hidup), dan individu yang tidak terpilih akan langsung dibuang (seperti seleksi alam). Untuk setiap individu baru yang dihasilkan, ada suatu fungsi untuk merubah *String* dalam bentuk individu tersebut kedalam bentuk represetasi data jadwal untuk kemudian dicek apakah individu baru yang terbentuk memenuhi *constraint* yang ada atau tidak. Program akan berhenti setelah semua jadwal terjadwal (*complete*) dan memenuhi *constraint* (*consistent*), atau saat program sudah melakukan iterasi sebanyak n kali meskipun belum ditemukan jadwal yang memenuhi *constraint*.

4. GUI

   Graphical User interface (GUI) dari program penjadwalan ini merupakap *Web Based Application*, dimana bahasa pemrograman yang digunakan berupa html, css, php, dan javascript. Fitur yang ada pada GUI antara lain :

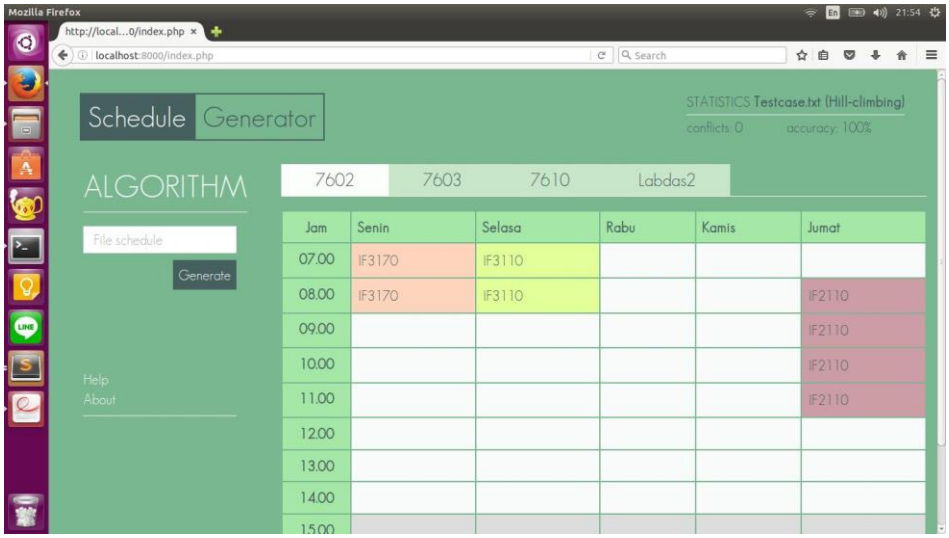
   1. Pengguna dapat memilih file jadwal (dalam format .txt)
   2. Pengguna dapat memilih algoritma apa yang ingin digunakan
   3. Program akan menampilkan hasil penjadwalan dalam format setiap ruangan
   4. Program akan menampilkan jumlah mata kuliah yang bentrok dan presentase penggunaan ruangannya

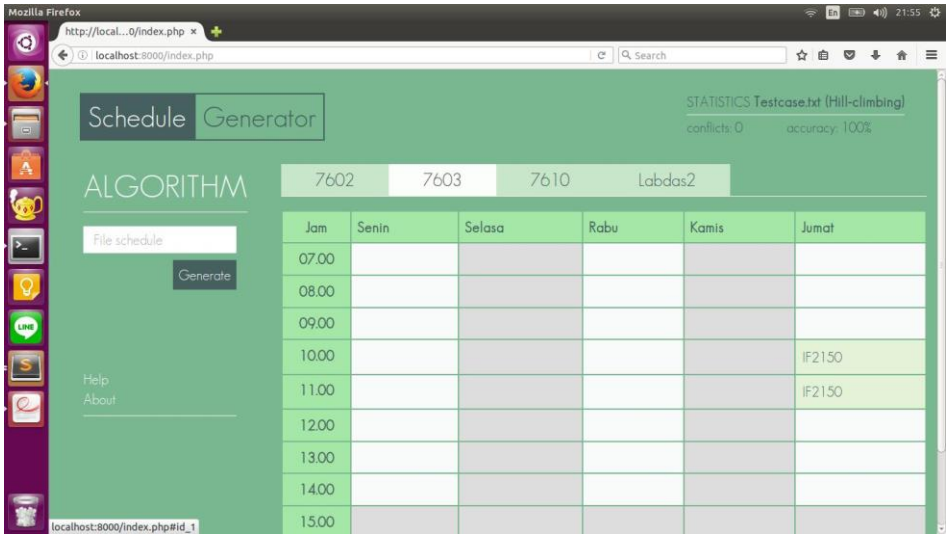
D. PENGUJIAN

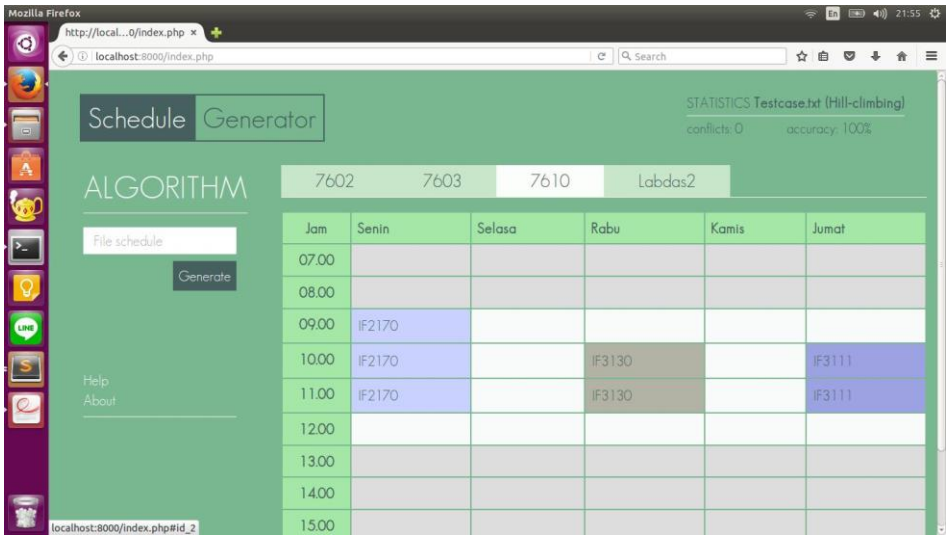
Testcase yang digunakan unutk pengujian ini adalah *testcase* dari asisten sebagai berikut.

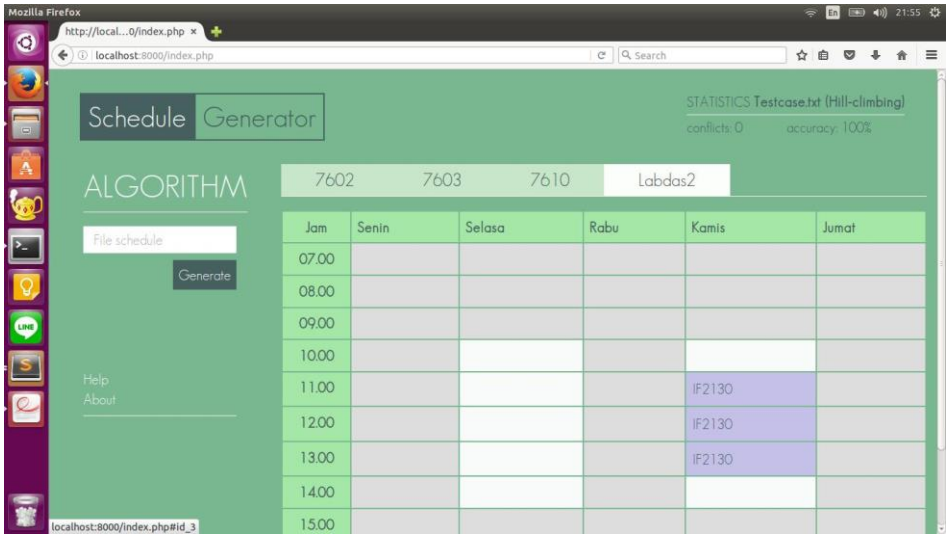
```
Ruangan
7602;07.00;14.00;1,2,3,4,5
7603;07.00;14.00;1,3,5
7610;09.00;12.00;1,2,3,4,5
Labdas2;10.00;14.00;2,4

Jadwal
IF2110;7602;07.00;12.00;4;1,2,3,4,5
IF2130;-;10.00;16.00;3;3,4
IF2150;-;09.00;13.00;2;1,3,5
IF2170;7610;07.00;12.00;3;1,2,3,4,5
IF3110;7602;07.00;09.00;2;1,2,3,4,5
IF3130;-;07.00;12.00;2;3,4,5
IF3170;7602;07.00;09.00;2;1,2,3,4,5
IF3111;-;07.00;12.00;2;1,2,3,4,5
```

1. Hasil Uji dengan *Hill-climbing Algorithm*

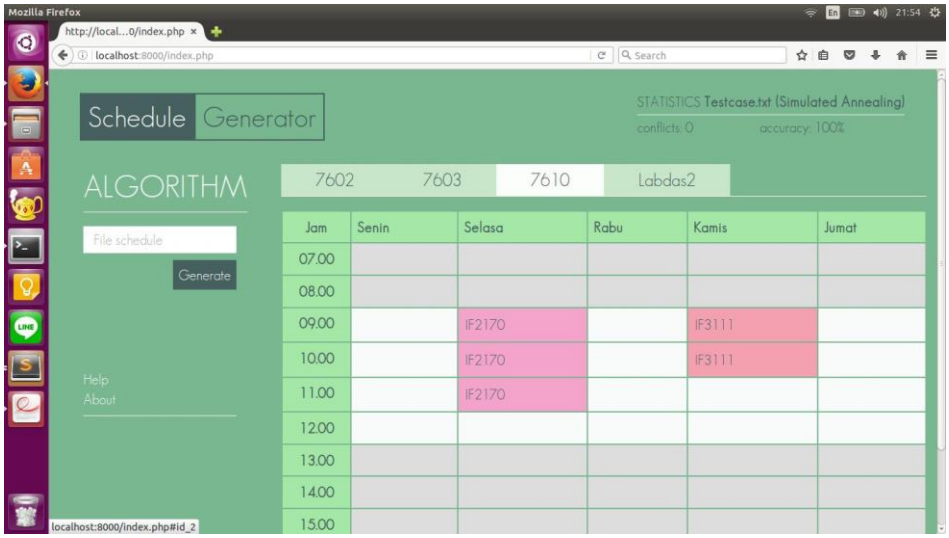


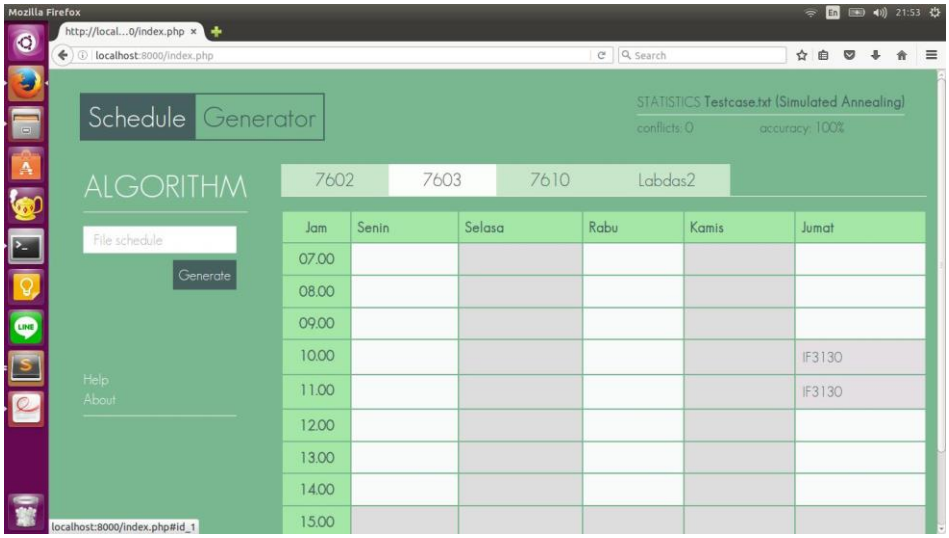

Mozilla Firefox
http://local...0/index.php
localhost:8000/index.php
Search
Schedule Generator
STATISTICS Testcase.txt (Hill-climbing)
conflicts: 0
accuracy: 100%
ALGORITHM
7602
7603
7610
Labdas2
File schedule
Generate
Help
About
Jam
Senin
Selasa
Rabu
Kamis
Jumat
07.00
08.00
09.00
10.00
11.00
12.00
13.00
14.00
15.00
IF2150
IF2150
localhost:8000/index.php#id_1

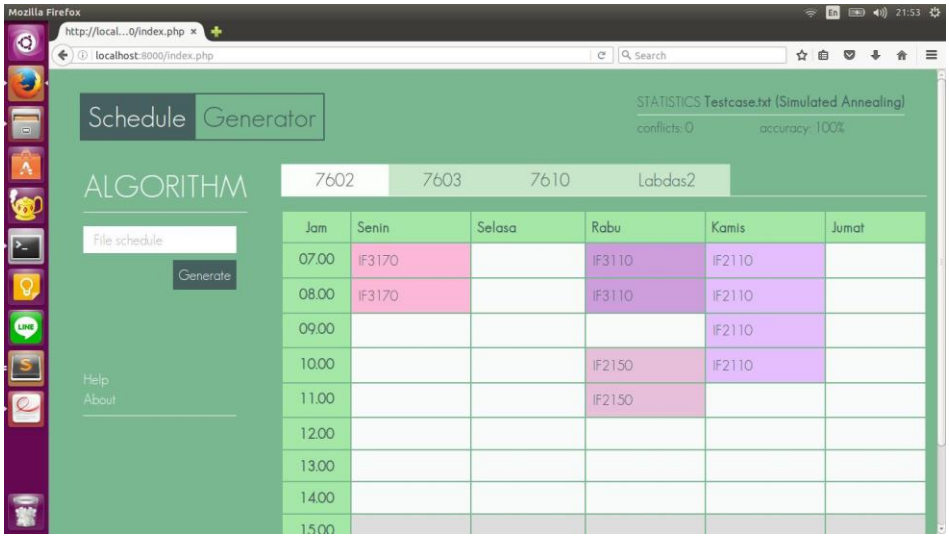

Mozilla Firefox
http://local...0/index.php
localhost:8000/index.php
Search
Schedule Generator
STATISTICS Testcase.txt (Hill-climbing)
conflicts: 0
accuracy: 100%
ALGORITHM
7602
7603
7610
Labdas2
File schedule
Generate
Help
About
Jam
Senin
Selasa
Rabu
Kamis
Jumat
07.00
08.00
09.00
10.00
11.00
12.00
13.00
14.00
15.00
IF2170
IF2170
IF2170
IF3130
IF3130
IF3111
IF3111
localhost:8000/index.php#id_2

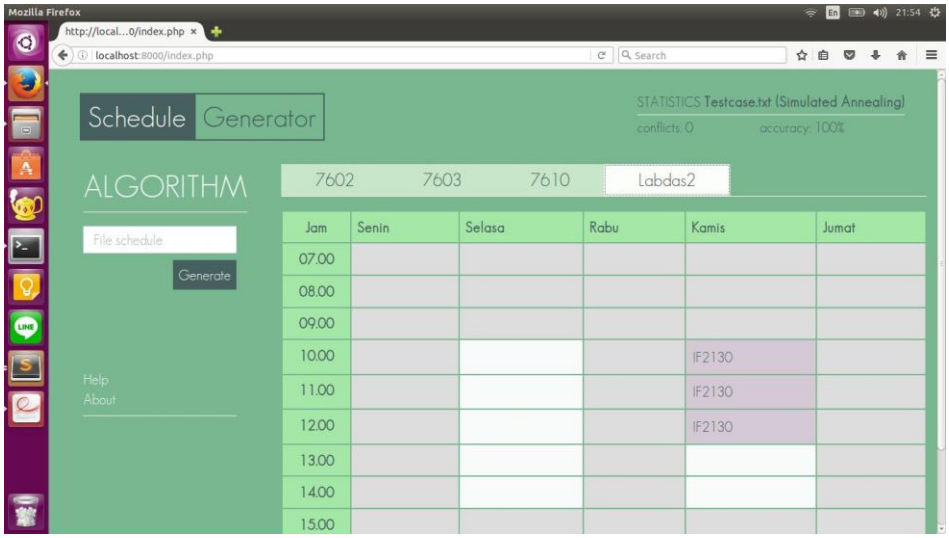

Mozilla Firefox
http://local...0/index.php
localhost:8000/index.php
Search
Schedule Generator
STATISTICS Testcase.txt (Hill-climbing)
conflicts: 0
accuracy: 100%
ALGORITHM
7602
7603
7610
Labdas2
File schedule
Generate
Help
About
Jam
Senin
Selasa
Rabu
Kamis
Jumat
07.00
08.00
09.00
10.00
11.00
12.00
13.00
14.00
15.00
IF2130
IF2130
IF2130
localhost:8000/index.php#id_3

2. Hasil Uji dengan *Simulated Annealing Algorithm*

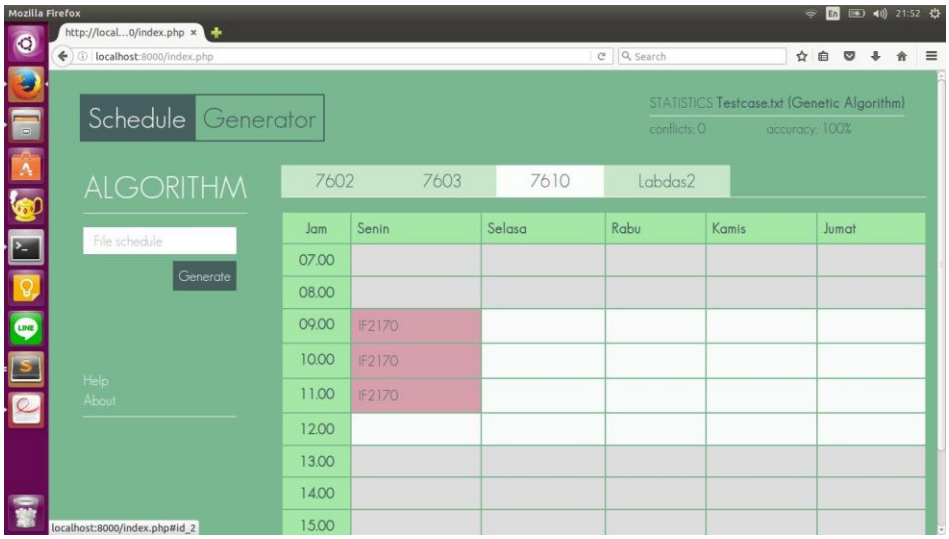


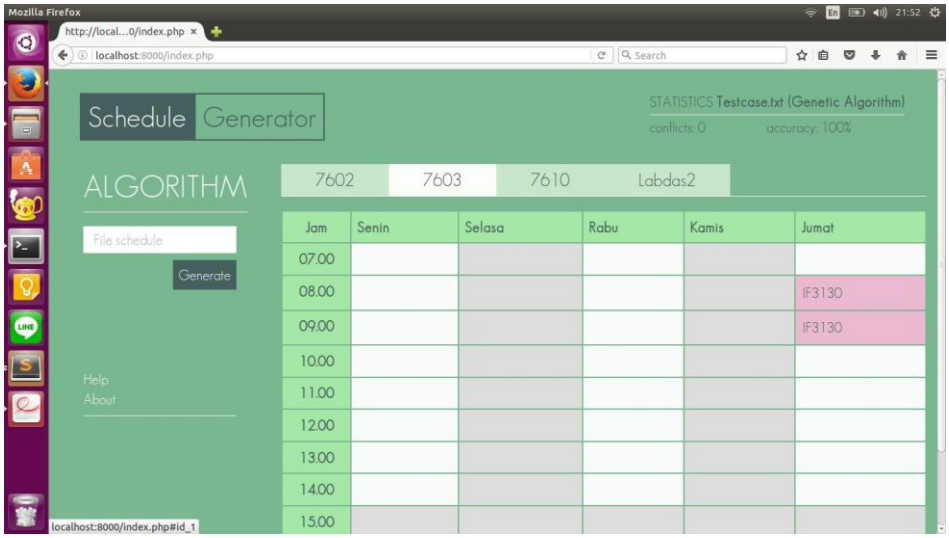


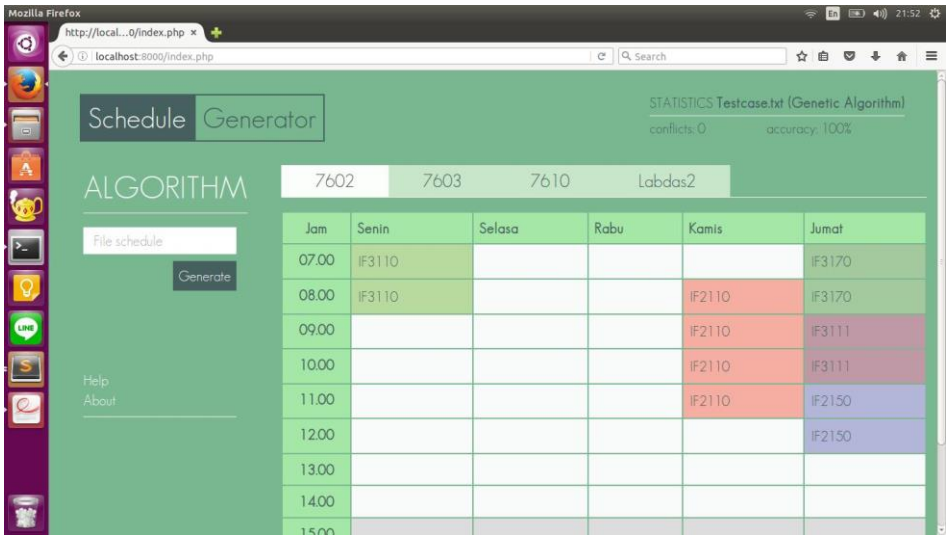


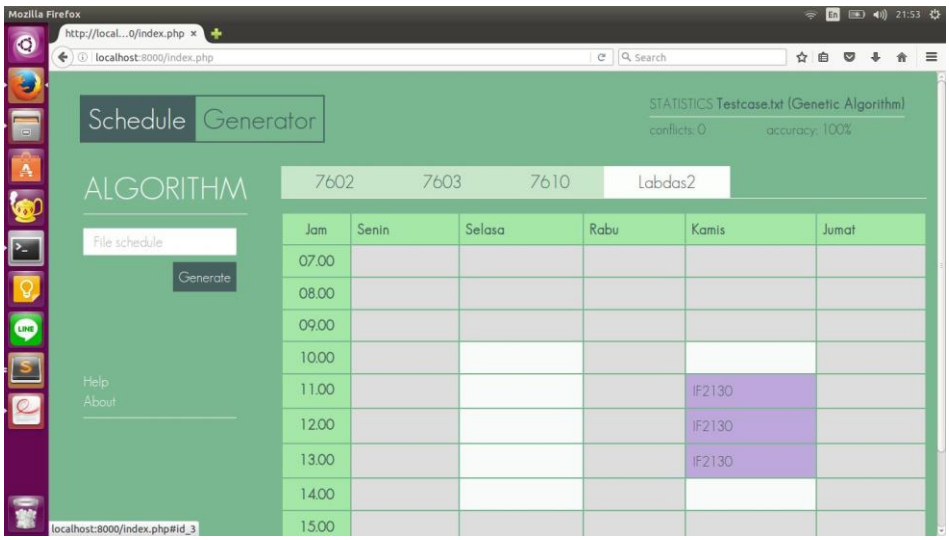


3. Hasil Uji dengan *Genetic Algorithm*




Mozilla Firefox
http://local...0/index.php
localhost:8000/index.php
Search
Schedule Generator
STATISTICS Testcase.txt (Genetic Algorithm)
conflicts: 0
accuracy: 100%
ALGORITHM
7602
7603
7610
Labdas2
File schedule
Generate
Help
About
Jam
Senin
Selasa
Rabu
Kamis
Jumat
07.00
08.00
09.00
10.00
11.00
12.00
13.00
14.00
15.00
IF3130
IF3130
localhost:8000/index.php#id_1


Mozilla Firefox
http://local...0/index.php
localhost:8000/index.php
Search
Schedule Generator
STATISTICS Testcase.txt (Genetic Algorithm)
conflicts: 0
accuracy: 100%
ALGORITHM
7602
7603
7610
Labdas2
File schedule
Generate
Help
About
Jam
Senin
Selasa
Rabu
Kamis
Jumat
07.00
08.00
09.00
10.00
11.00
12.00
13.00
14.00
IF3110
IF3110
IF3170
IF3170
IF2110
IF2110
IF2110
IF2110
IF3111
IF3111
IF2150
IF2150


Mozilla Firefox
http://local...0/index.php
localhost:8000/index.php
Search
Schedule Generator
STATISTICS Testcase.txt (Genetic Algorithm)
conflicts: 0
accuracy: 100%
ALGORITHM
7602
7603
7610
Labdas2
File schedule
Generate
Help
About
Jam
Senin
Selasa
Rabu
Kamis
Jumat
07.00
08.00
09.00
10.00
11.00
12.00
13.00
14.00
15.00
IF2130
IF2130
IF2130
localhost:8000/index.php#id_3