**Penggunaan *Local Search* untuk Penyusunan**

**Jadwal Kuliah**

****

Tugas Besar 1 IF3170 Inteligensi Buatan

Semester I tahun 2016/2017

Disusun Oleh :

Davin Prasetya 13514003

Nikolas Wangsaputra 13514048

Nathan James Runtuwene 13514083

Jovian Christianto 13514101

Alvin Junianto Lan 13514105

**IF3170 – Inteligensi Buatan**

**Program Studi Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika**

**INSTITUT TEKNOLOGI BANDUNG**

1. **Spesifikasi**

Dalam setiap kegiatan perlu adanya pengaturan waktu dan tempat yang baik, dalam kasus ini konteks yang digunakan adalah penjadwalan mata kuliah di perkuliah IF Institut Teknologi Bandung. Penjadwalan disini meliputi waktu pelaksanaan dan ruangan yang akan digunakan, tujuan dari penjadwalan ini adalah memastikan setiap mata kuliah dapat berlangsung pada waktu dan ruang yang tepat sehingga tidak ada mata kuliah yang jadwalnya akan bertabrakan, yaitu berada pada waktu dan ruangan yang sama.

Spesifikasi dari jadwal yang akan digunakan dalam persoalan kali ini adalah 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.
5. **Dasar Teori**
   1. **Local Search**

Untuk menyelesaikan persoalan yang sudah dideskripsikan pada bagian sebelumnya, akan digunakan algoritma *local search*. *Local search* adalah sebuah algoritma untuk melakukan pencarian solusi dengan metode pembelajaran. Beberapa jenis *local search* yang akan digunakan pada penyelesaian persoalan kali ini adalah :

1. ***Hill Climbing Search***

*HillCclimbing* adalah salah satu teknik local search dimana pencarian solusi dilakukan dengan cara menentukan sebuah solusi sementara, yang kemudian dimodifikasi salah satu elemennya. Bila hasilnya lebih baik, iterasi modifikasi salah satu elemen akan dilanjutkan sampai tidak ada modifikasi yang dapat memperbaiki solusi lagi.

Algoritma hill climbing merupakan salah satu cara yang sangat efisien untuk menemukan *local maxima*, namun tidak menjamin dapat mencapai *global maxima*.

Berikut adalah langkah-langkah algoritma *Hill Climbing* Search secara umum.

* 1. Tentukan state acak dari permasalahan
  2. Pindah ke state berikutnya
  3. Evaluasi apakah state tersebut lebih baik dari state sebelumnya
  4. Jika lebih baik, maka ambil state tersebut. Jika tidak, tetap pada state sebelumnya
  5. Lakukan langkah b, c, dan d hingga solusi ditemukan / terjadi *local maxima*

1. ***Simulated Annealing***

*Simulated Annealing* merupakan teknik *local search* yang cara kerjanya serupa dengan *Hill Climbing*, namun dengan memperkenalkan parameter baru yaitu Temperatur (T).

Kelemahan dari algoritma *Hill Climbing* adalah solusi dapat terjebak pada *local minima* secara acak, dimana solusi pada *local minima* tersebut seringkali tidak cukup baik untuk digunakan. Dengan *Simulated Annealing,* algoritma dapat memilih langkah buruk sesuai batas (temperatur) yang telah ditentukan, sehingga tidak semua langkah harus memperbaiki solusi yang ada. Dengan cara ini, toleransi pencarian menjadi lebih luas sehingga mengurangi kemungkinan terjebak di *local maxima*. Sebagai catatan, jika temperatur pada *Simulated Annealing* bernilai 0, maka algoritma *Simulated Annealing* akan sama dengan algoritma *Hill Climbing*.

Berikut adalah langkah-langkah algoritma *Simulated Annealing* secara umum.

* 1. Tentukan state acak dari permasalahan
  2. Tentukan temperatur awal dan konstanta pengali untuk menurunkan temperatur.
  3. Pindah ke state berikutnya secara acak
  4. Evaluasi apakah state tersebut lebih baik dari state sebelumnya
  5. Jika lebih baik, maka ambil state tersebut.
  6. Jika lebih buruk, maka hitung kemungkinan state itu akan diambil dengan rumus

Keterangan :

a : *Acceptance Probability* / Probabilitas

Cold : *Cost* lama

Cnew : *Cost* baru

T : temperature

* 1. Nilai probabilitas tersebut akan dibandingkan dengan suatu nilai acak antara 0 – 1. Jika nilai probabilitas tersebut lebih besar dari suatu nilai acak tersebut, maka state tersebut diambil. Begitu juga sebaliknya.
  2. Lakukan langkah c, d, e, f, dan g hingga solusi ditemukan / terjadi *local maxima.* Untuk setiap iterasi, temperatur berkurang dengan nilai temperatur tersebut dikali suatu konstanta pengali.

1. ***Genetic Algorithm***

*Genetic Algorithm* adalah varian lain dari *local search* yang juga melibatkan pilihan acak. Langkah dari *Genetic Algorithm* adalah memilih 2 solusi random, menyilangkan 2 solusi tersebut, kemudian melakukan mutasi (mengubah salah satu elemen pada hasil persilangan).

Berikut adalah langkah-langkah algoritma *Genetic Algorithm* secara umum.

* + - * 1. Tentukan ukuran populasi
        2. Tentukan state secara acak untuk setiap populasi
        3. Hitung fitness function untuk setiap populasi
        4. Lakukan selection dengan memilih populasi sesuai nilai dari fitness function
        5. Lakukan cross over dengan menggabungkan 2 populasi dengan batas acak dan menghasilkan populasi yang baru
        6. Lakukan mutation dengan mengganti salah satu nilai state dari masing-masing populasi dengan nilai acak
        7. Lakukan langkah c, d, e, dan f untuk populasi-populasi yang baru hingga menemukan solusi

1. **Implementasi**
   1. **Deskripsi Program**

Penyelesaian untuk mengatur penjadwalan dibuat dalam bahasa pemrograman java , untuk hasil ditampilkan dalam sebuah webapp yang dibuat dalam bahasa pemrograman HTML, PHP dan JavaScript.

Pada tampilan awal program akan ditampilkan pilihan algoritma yang akan digunakan, yaitu *Hill Climbing Search*, *Simulated Annealing*, dan *Genetic Algorithm*. Ada juga masukan jumlah step dari pengguna dan jumlah populasi khusus untuk *Genetic Algorithm* dan temperature awal khusu untuk *Simulated Annealing*

Setelah algoritma dipilih, program akan menjalankan program java dari algoritma yang bersangkutan. Hasil dari eksekusi program ditampilkan berupa tabel hari dan jam. Untuk setiap kotak berisi mata kuliah sesuai hasil eksekusi program. Jumlah bentrok dan persentasi keefektifan penggunaan ruangan juga ditampilkan dalam webapp tersebut.

* 1. **Dokumentasi Program**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class Activity** | | | | |
| String name  Boolean[] day  Int lmtStart  Int lmtFinish  Int duration | | | String room  String tempRoom  Int start  Int tempday | |
| **Nama Fungsi** | **Parameter** | **Output** | | **Deskripsi** |
| setRandomStart | Int, int | - | | Mengeset Attribut Start secara random dari 2 parameter masukan representasi batas awal dan akhir |

|  |  |
| --- | --- |
| **Class Classroom** | |
| String name  Int openTime | Int closedTime  Boolean[] day |

|  |
| --- |
| **Class PairActivity** |
| Activity first;  Activity second; |

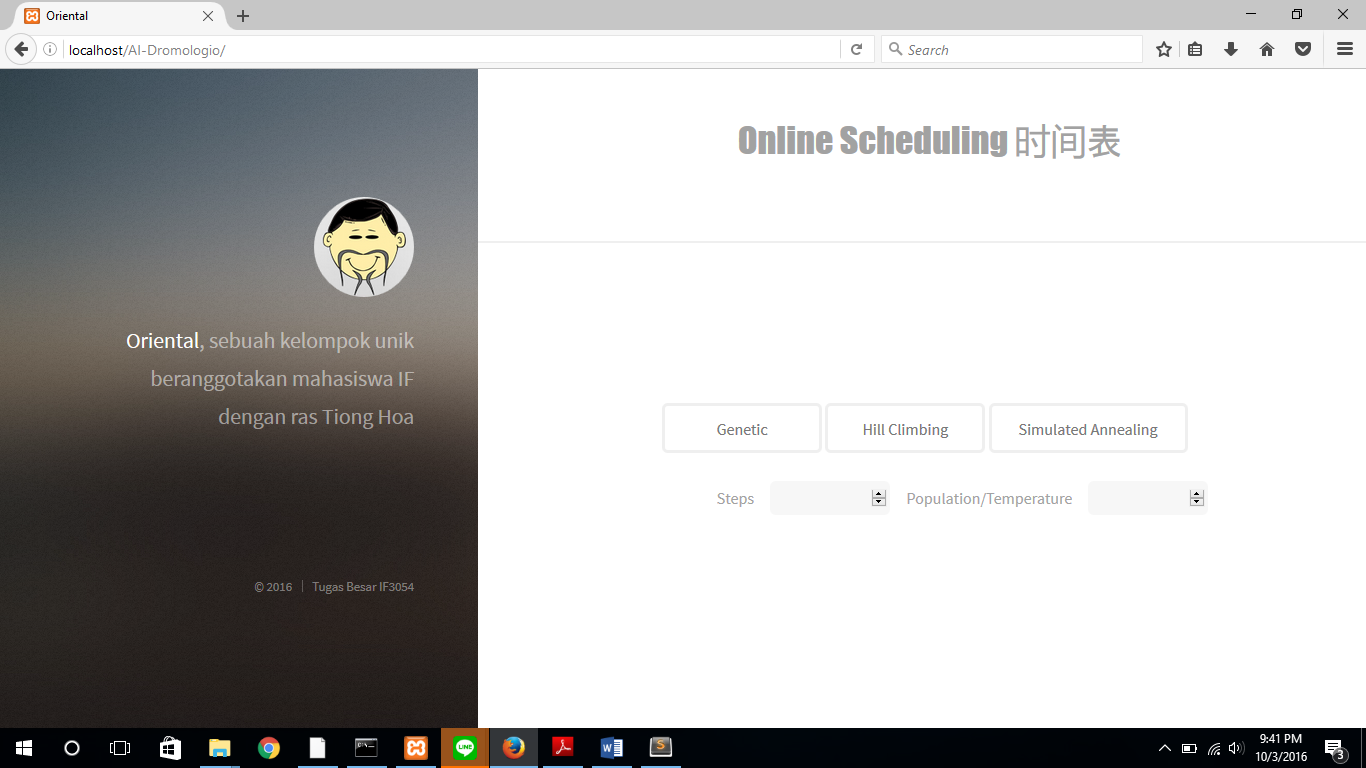
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class CSP** | | | | |
| ArrayList<Activity> arrAct  ArrayList<Classroom> arrClass | | | ArrayList<PairActivity> arrError | |
| **Nama Fungsi** | **Parameter** | **Output** | | **Deskripsi** |
| findClassroom | String | Classroom | | Menemukan Object bertipe Classroom yang memiliki nama sesuai parameter |
| randomClass | - | Classroom | | Mengeluarkan Object bertipe Classroom yang dipilih secara random dari ArrayList of Classroom |
| checkDay | Activity, Classroom | ArrayList<Integer> | | Mengeluarkan sebuah ArrayList of Integer yang merupakan hasil seleksi dari ketersediaan hari Classroom dan Activity |
| randomDay | Activity, Classroom | Integer | | Mengeluarkan sebuah Integer yang merupakan hari yang dipilih secara random dari hasil fungsi checkday |
| setRandomActivity | Activity | - | | Mengeset Object Activity ke suatu hari, tempat, dan mengeset jam mulainya secara random dari opsi yang memungkinkan |
| setRandomAllActivity | - | - | | Melakukan setRandomActivity untuk setiap Activity |
| checkViolation | - | - | | Melakukan pengecekan pelanggaran dari ArrayList of Activity dan Classroom |
| countViolation | - | Int | | Mengeluarkan suatu Integer yang merupakan banyaknya pelanggaran yang terjadi |
| printAllActivity | - | - | | Mengeluarkan semua kondisi Activity ke dalam console. |
| Percentage | - | Double | | Mengeluarkan suatu Double yang merupakan hasil perbandingan jadwal terisi dan jadwal tersedia |

|  |  |  |  |
| --- | --- | --- | --- |
| **Class Hillclimbing** | | | |
| Int violation  Int steps | | | |
| **Nama Fungsi** | **Parameter** | **Output** | **Deskripsi** |
| selectStep | - | Activity | Mengeluarkan suatu Object Activity yang merupakan Activity dengan kemunculan pelanggaran terbanyak |
| setSaveState | Activity, String, Int, Int | - | Mengeset attribute Activity untuk dilakukan backtracking |
| run | - | - | Menjalankan fungsi dasar Hillclimbing secara keseluruhan |

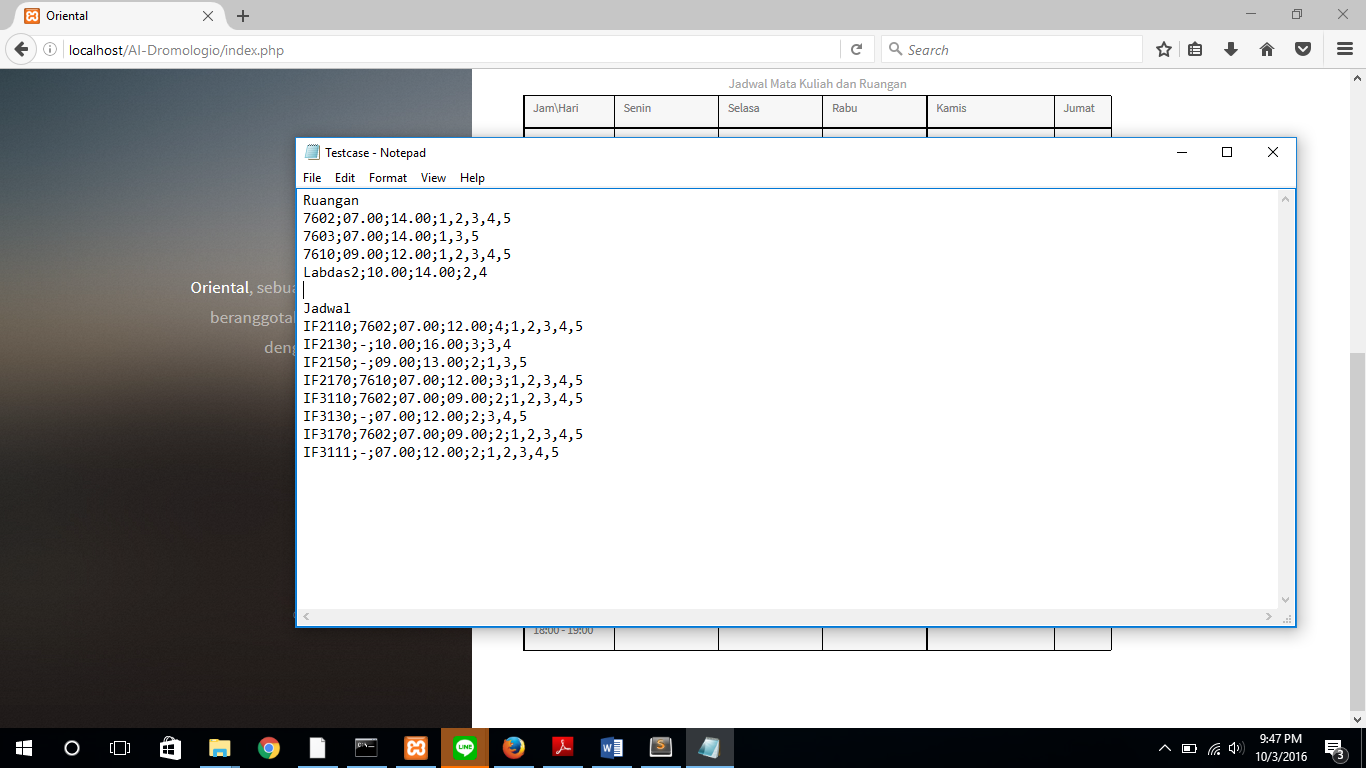
|  |  |  |  |
| --- | --- | --- | --- |
| **Class SimulatedAnnealing** | | | |
| Int violation  Int steps  Double temperature  Double r | | | |
| **Nama Fungsi** | **Parameter** | **Output** | **Deskripsi** |
| selectStep | - | Activity | Mengeluarkan suatu Object Activity yang merupakan Activity dengan kemunculan pelanggaran terbanyak |
| setSaveState | Activity, String, Int, Int | - | Mengeset attribute Activity untuk dilakukan backtracking |
| run | - | - | Menjalankan fungsi dasar SimulatedAnnealing secara keseluruhan |

|  |  |  |  |
| --- | --- | --- | --- |
| **Class Genetic** | | | |
| ArrayList <CSP> population  Int populationSize  Int steps  Int maximumViolation | | | |
| **Nama Fungsi** | **Parameter** | **Output** | **Deskripsi** |
| Genetic  (Constructor) | Int populationSize,  Int steps,  ArrayList<Activity> activities, ArrayList<Classroom> classrooms | - | Menset nilai atribut dan juga membuat populasi sebanyak populationSize secara random |
| countTotalFitness | - | int | Menghitung nilai fitness total populasi, nilai fitnessmasing-masing dihitung dengan maximumViolation dikurangi CountViolation |
| selection | Int totalFitness | CSP | Memilih secara random (Roulete) salah satu dari populasi |
| crossover | ArrayList<Activity> activities1, ArrayList<Activity> activities2 | ArrayList<Activity> | Melakukan penyilangan activities1 dan activities2 pada index random |
| mutate | CSP csp | - | Mengassign ulang salah satu variabel pada CSP |
| breed | - | CSP | Melakukan “perkembangbiakan” pada 2 CSP random dan menghasilkan CSP baru |
| newGeneration | - | int | Melakukan breed sebanyak populationSize kali |
| findBestCSP | - | CSP | Mencari set dengan violation paling sedikit |
| run | - | CSP | Menjalankan fungsi dasar Genetic secara keseluruhan dan mengembalikan set activity yang didapat paling efektif |

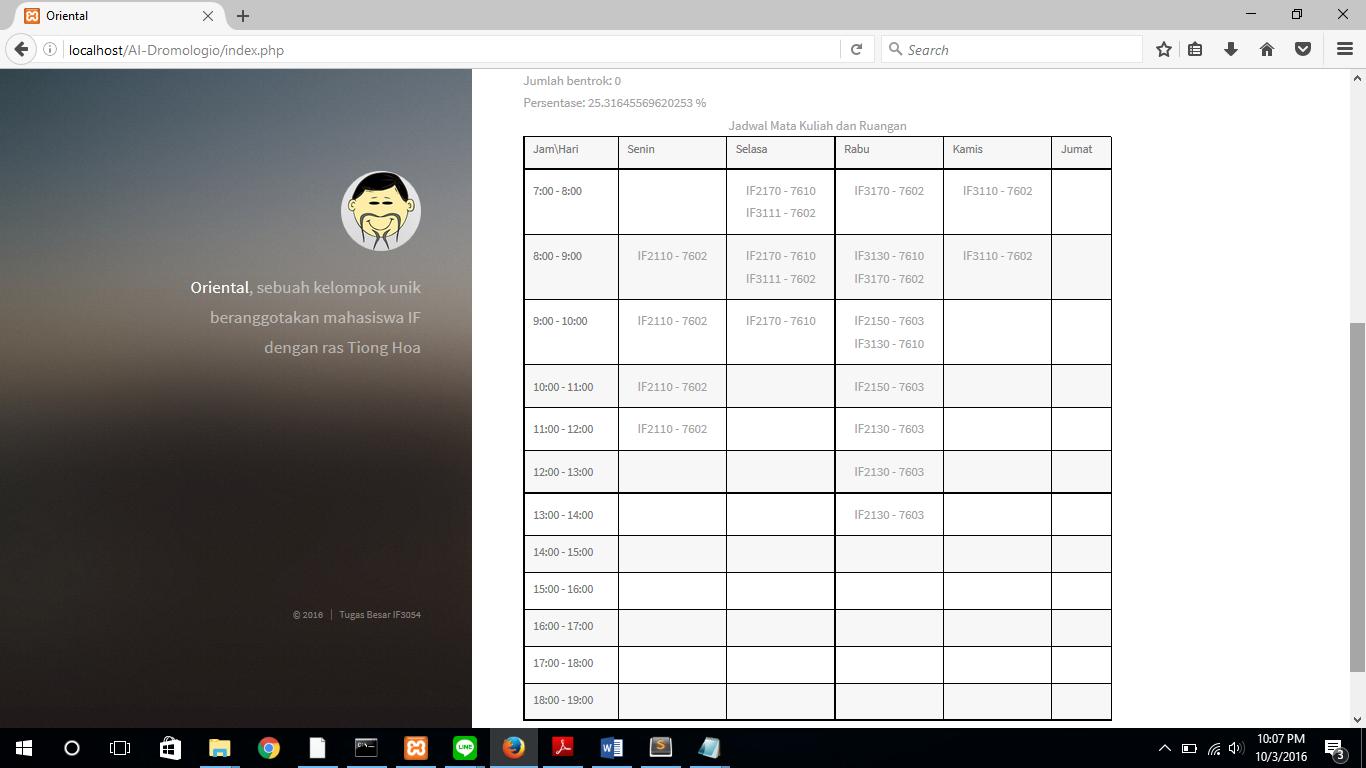
* 1. **Data Uji Coba**



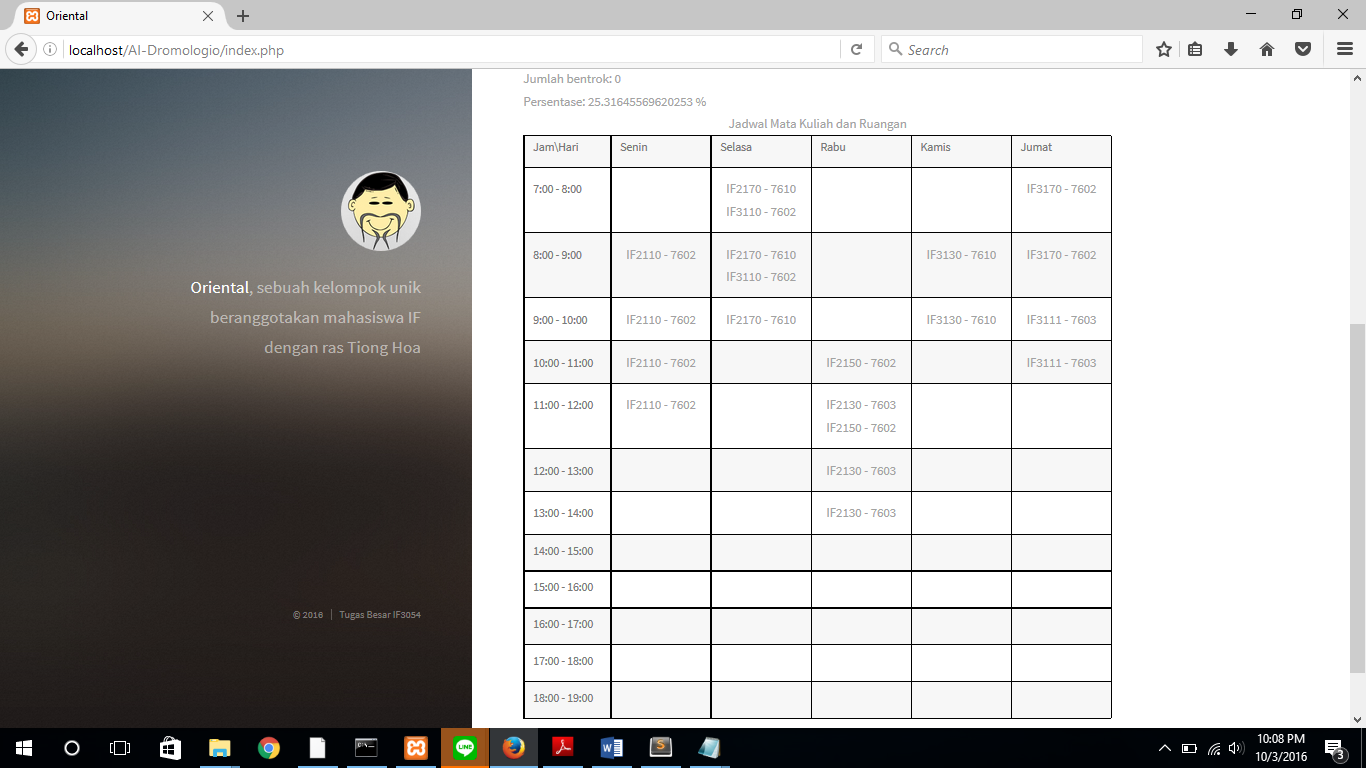
Gambar 1. Halaman Awal Oriental



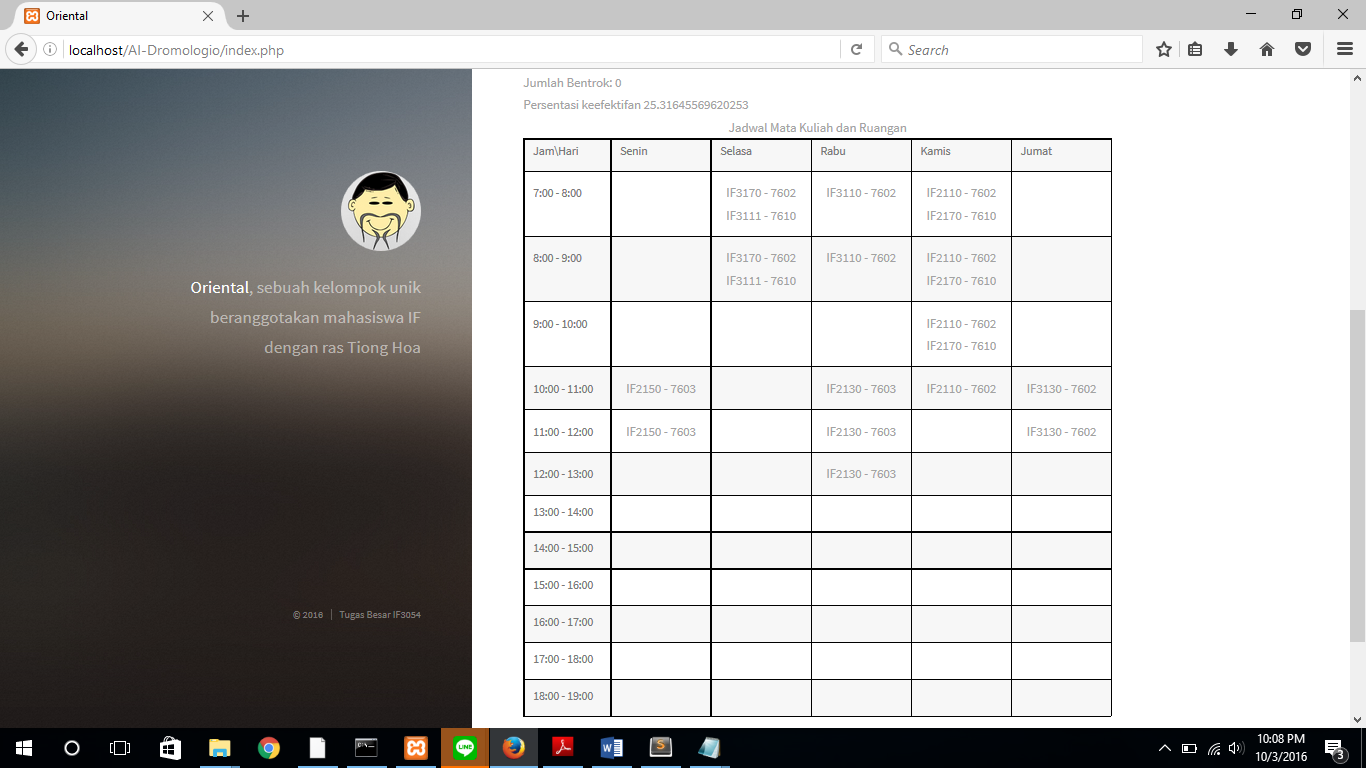
Gambar 2. Masukan Uji Coba 1



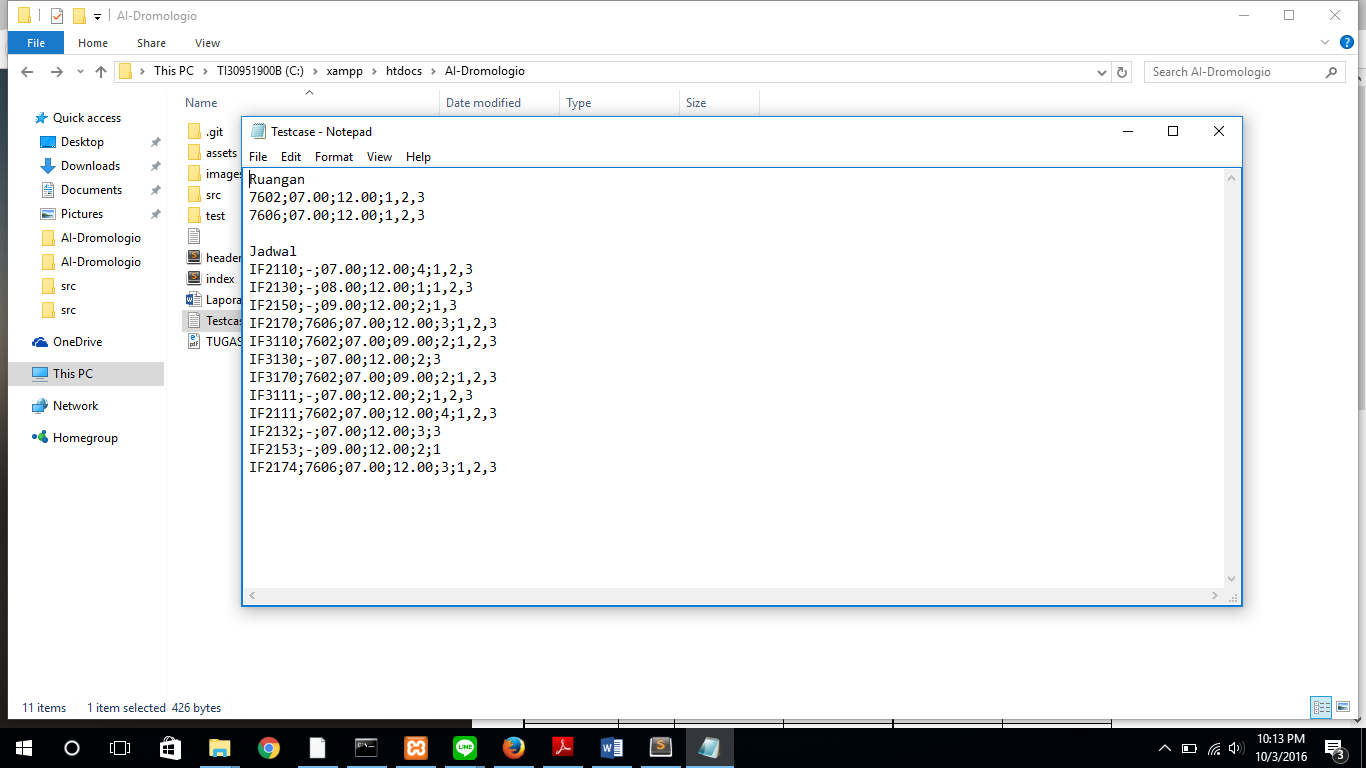
Gambar 3. Hasil Penjadwalan Uji Coba 1 dengan Algoritma *Hill Climbing* (Steps = 3000)



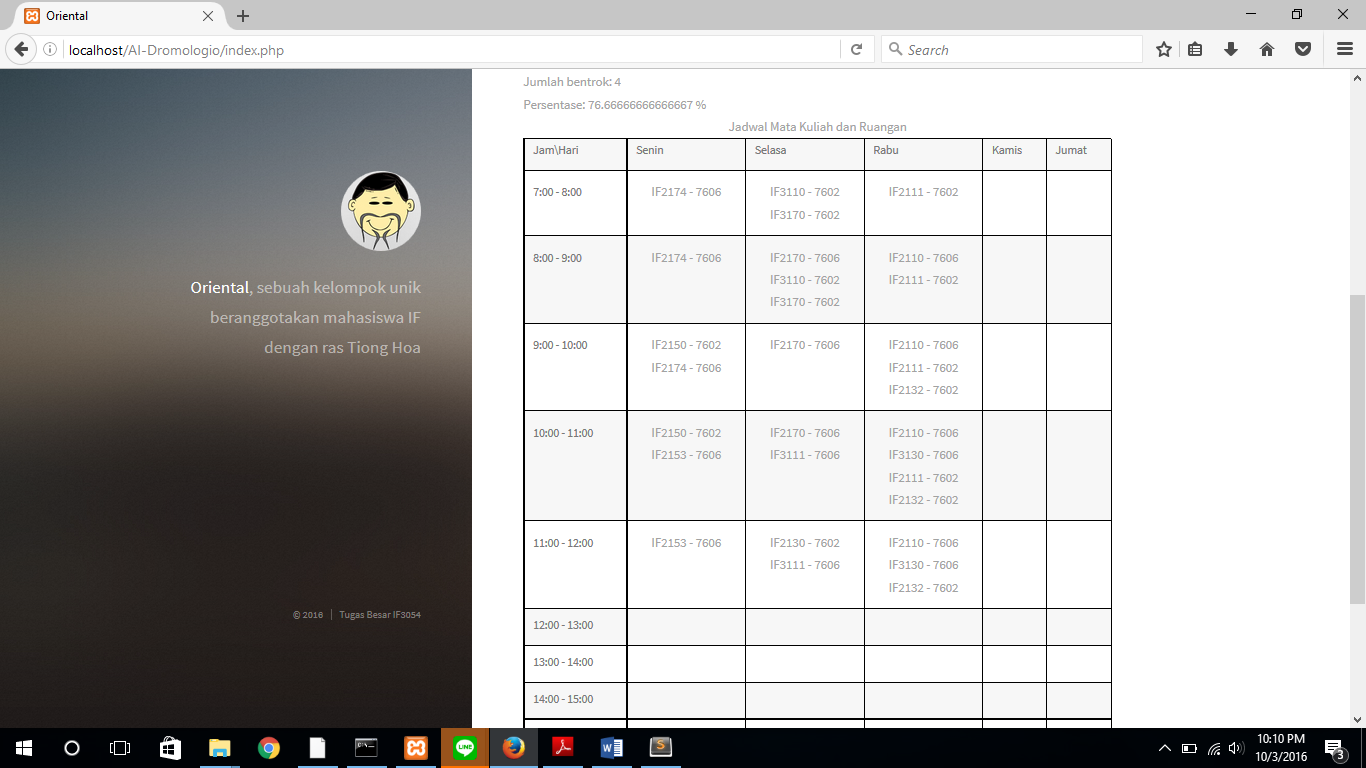
Gambar 4. Hasil Penjadwalan Uji Coba 1 dengan Algoritma *Simulated Annealing* (Steps = 3000, Temperatur = 100)



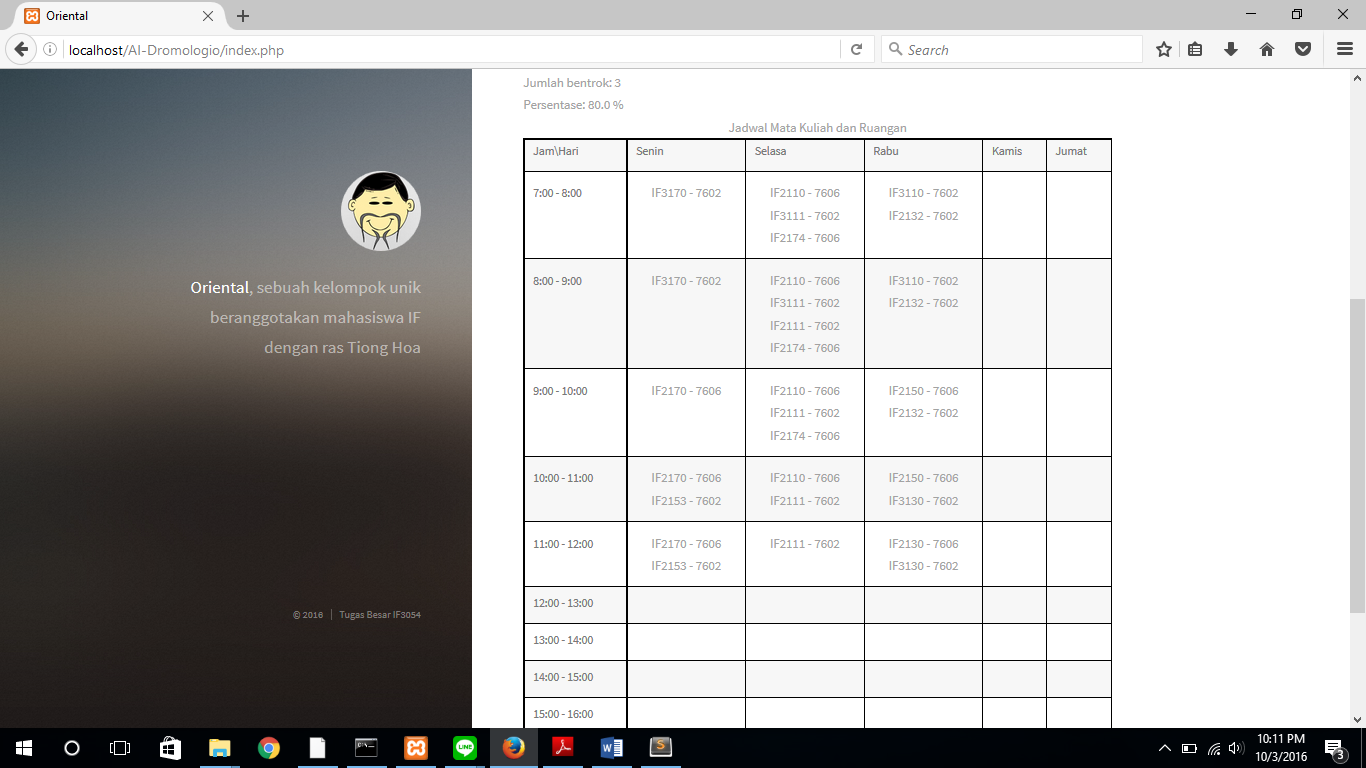
Gambar 5. Hasil Penjadwalan Uji Coba 1 dengan Algoritma *Genetic Algorithm* (Steps = 3000, Populasi = 10)



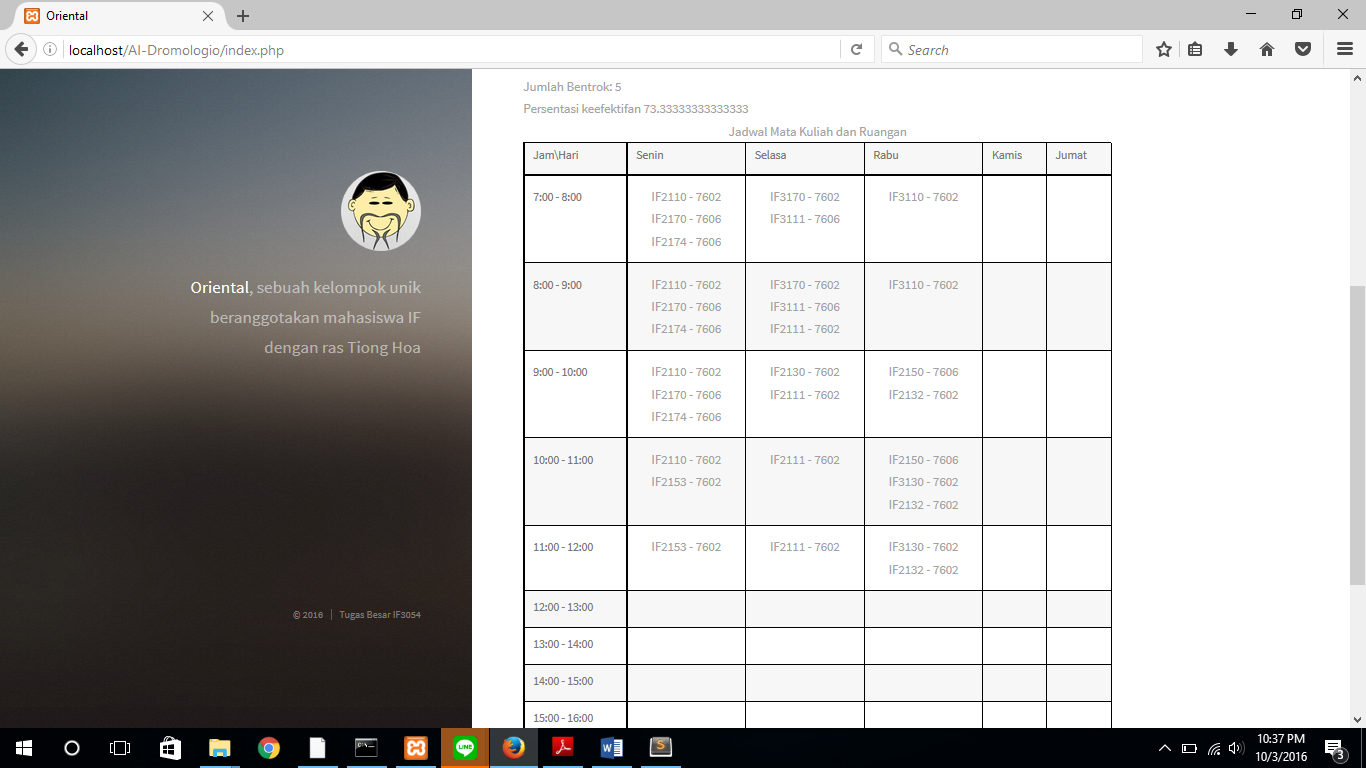
Gambar 6. Masukan Uji Coba 2



Gambar 7. Hasil Penjadwalan Uji Coba 2 dengan Algoritma *Hill Climbing* (Steps = 5000)



Gambar 8. Hasil Penjadwalan Uji Coba 2 dengan Algoritma *Simulated Annealing* (Steps = 5000, Temperatur = 100)



Gambar 9. Hasil Penjadwalan Uji Coba 2 dengan Algoritma *Genetic Algorithm* (Steps = 5000, Populasi = 100)