**IF 3170 INTELEGENSIA BUATAN**

**TUGAS BESAR I PENJADWALAN KEGIATAN MENGUNAKAN ALGORITMA *LOCAL SEARCH***



**Oleh:**

**13514040 Devin Lukianto**

**13514045 Elvina R. K. Situmorang**

**13514058 Jason Jeremy Iman**

**13514077 Atika Azzahra Akbar**

**13511079 Kevin Verdi**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2016**

**BAB I**

**DESKRIPSI PERSOALAN**

1. **Deskripsi Persoalan**

Jadwal yang akan digunakan dalam tugas memiliki spesifikasi 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 mencocokan jadwal terhadap waktu dan ruang dengan tepat

Format jadwal:

Format teks di bawah adalah contoh jadwal yang akan digunakan.

****

Dalam pengerjaan tugas ini terdapat beberapa algoritma yang perlu diterapkan yaitu:

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

**BAB II**

**TEORI ALGORITMA LOCAL SEARCH**

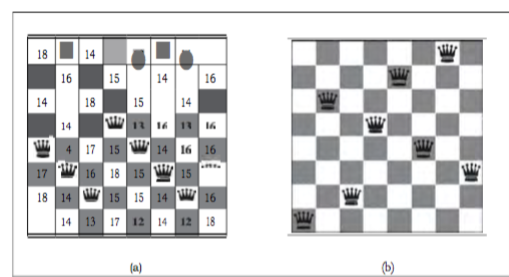
1. ***Hill Climbing Search***

Algoritma Hill Climbing Search adalah salah satu pencarian solusi *local search* yang tidak diperlu dibentuk pohon pencarian. Node pada algoritma hanya perlu menyimpan state dan nilai fungsi objektifnya (nilai *heuristic).*

Algoritma ini juga sering disebut sebagai *greedy local search* karena hanyamemilih *state* tetangga yang menuju ke solusi lebih baik (*good neighbor)* tanpa memikirkan *state* setelahnya yang harus dipilih. Jika tidak lagi ditemukan tetangga yang lebih baik, *current* node akan menjadi solusi paling optimal. Di bawah ini adalah pseudocode untuk fungsi Hill Climbing:

|  |
| --- |
| **function** Hill-Climbing(problem)  state local maximum  current  Make-Node(problem, Initial-State)  loop do  neighbor  Highest-valued successor of current  if neighbor.Value < current.Value then   current.STATE  current  neighboor |

Contoh pemanfaatan algoritma *Hill Climbing* dengan penyelesaian soal 8 *Queens* seperti di gambar bawah ini. Pada gambar (a) nilai *heuristic*-nya adalah 17, tetapi dengan menggunakan algoritma Greedy Descent, hanya dengan 5 tahapan penempatan para ratu berubah menjadi seperti di gambar (b) dengan nilai *heuristic* persoalan berkurang menjadi 1.



Algoritma ini terbukti mampu mendapatkan hasil yang mendapatkan solusi, namun sayangnya solusi tersebut sering kali terjebak contohnya pada *local maximum,* seperti soal 8 *Queens* sebelumnya yang mendapatkan solusi dimana nilai herusiticnya adalah 1*, ridges,* atau *plateaux.*

1. ***Simulated Annealing***

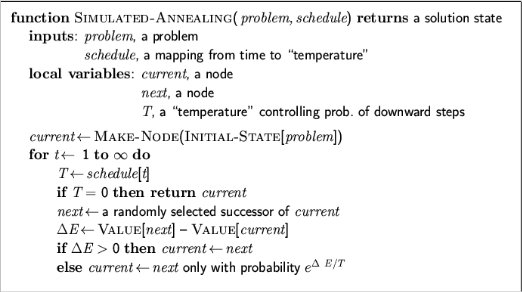
*Hill climbing* adalah algoritma yang tidak pernah mengambil langkah *downhill* atau *bad move*, sehingga dapat stuck pada *local maximum*. Algoritma *purely random walk* dapat selalu menyelesaikan masalah dengan tepat, namun algoritma ini tidak efektif. Sehingga untuk menyelesaikan masalah ini diciptakan algoritma *Simulated Annealing*. Algoritma *Simulated Annealing* adalah sebuah algoritma yang menggabungkan algoritma *hill climbing* dan *random walk*.

Algoritma ini memperbolehkan untuk melakukan *bad move*. Penentuan dari boleh atau tidak melakukan *bad move* dipengaruhi oleh probabilitas. Probabilitas sendiri terdiri dari 3 macam, yaitu

1. Probabilitas tetap, contoh
2. Probabilitas berkurang dipengaruhi oleh faktor waktu
3. Probabilitas berkurang dipengaruhi oleh faktor waktu dan yang meningkat.

Pada tipe ini, probabilitas diperoleh dari persamaan :

Dari tipe 2 dan 3, temperatur juga mempengaruhi probabilitas. Jika temperatur tinggi, maka akan cenderung memperbolehkan semua *moves* (*Random Walk*). Sementara ketika temperatur rendah akan melakukan algortiam *Stochastic Hill-Climbing*. Di bawah ini adalah *pseudocode* untuk algoritmta *Simulated Annealing* :



1. ***Genetic Algorithm***

*Genetic Algorithm* adalah algoritma yang menyelesaikan suatu problema dengan pendekatan yang menyerupai proses *natural selection* dan evolusi biologi. Algoritma ini akan terus menciptakan dan mengubah solusi-solusi baru (spesies). Tiap spesies akan menghasilkan populasi baru yang berevolusi ke solusi yang lebih baik.

Implementasi dari algoritma ini terdiri dari beberapa langkah diantaranya,

1. Membuat populasi baru, menciptakan beberapa solusi *random*

2. *Selection*, memilih solusi terbaik dari populasi tersebut

3. *Crossing*, melakukan perkawinan (penggabungan solusi) dari solusi-solusi yang sudah terbentuk dari hasil *selection*

4. *Mutation*, merubah bagian dari solusi untuk menghasilkan solusi yang baru

Di bawah ini adalah *pseudocode* untuk *Genetic Algorithm*,

|  |
| --- |
| function GeneticAlgorithm()  // start with an initial time  t := 0;  // initialize a usually random population of  // individuals  initpopulation P (t);  // evaluate fitness of all initial individuals of  // population  evaluate P (t);  // test for termination criterion (time, fitness, etc.)  while not done do  // increase the time counter  t := t + 1;  // select a sub-population for offspring  // production  P' := selectparents P (t);  // recombine the "genes" of selected parents  recombine P' (t);  // perturb the mated population stochastically  mutate P' (t);  // evaluate it's new fitness  evaluate P' (t);  // select the survivors from actual fitness  P := survive P,P' (t);  end GA. |

**BAB III**

**SOLUSI PERSOALAN**

1. **Pendifinisian Variabel, Domain, dan *Constraints***
2. **Variabel**

Variabel pada persoalan ini adalah Matkul[i]. Dimana tiap variabel akan dilakukan penetapan kelas, jam mulai, dan hari mata kuliah. Contoh pendefinisian variabel adalah sebagai berikut:

Matkul[1] = { Kelas: y, Jam mulai: z, Hari: q}

1. **Domain**

Setiap variable memilik tiga domain yaitu domain kelas (Domain\_kelasi), domain jam mulai (Domain\_jam\_mulaii), dan domain hari (Domain\_harii). Anggota domain untuk tiap variabel berbeda-beda sesuai dengan data yang diberikan. Contoh berikut adalah domain untuk variabel Matkul[1]:

Domain\_kelas1 = {7602}

Domain\_jam\_mulai1 = {07.00, 08.00, 09.00, 10.00, 11.00, 12.00}

Domain\_hari1 = {1, 2, 3, 4, 5}

1. **Constraints**

Untuk tiap variabel terdapat constraints:

1. Jika Matkul[i].hari = Matkul[j].hari, Matkul[i].Jam\_mulai = Matkul[j].Jam\_mulai maka Matkul[i].kelas ≠ Matkul[j].kelas.
2. Jika Matkul[i].hari = Matkul[j].hari, Matkul[i].kelas = Matkul[j].kelas maka :

* Matkul[i].Jam mulai >= Matkul[j].Jam mulai
* Matkul[i].Jam mulai < Matkul[j].Jam mulai + durasi Matkul j
* Matkul[j].Jam mulai >= Matkul[i].Jam mulai
* Matkul[j].Jam mulai < Matkul[i].Jam mulai + durasi Matkul i

1. Matkul[i].hari = hari dimana Matkul[i].kelas diperbolehkan
2. Matkul[i].Jam mulai >= Jam Matkul[i].kelas boleh mulai dipakai
3. Matkul[i].Jam mulai + durasi Matkul i <= Jam terakhir Matkul[i].kelas sebelum ditutup
4. **Source Code**

import sys, random, copy, math, operator

from PyQt4.QtCore import \*

from PyQt4.QtGui import \*

class Jadwal:

def \_\_init\_\_(self, kode, kelas, jam\_mulai, jam\_akhir, durasi, hari):

self.kode = kode

self.kelas = kelas

self.jam\_mulai = int(jam\_mulai[:2])

self.jam\_akhir = int(jam\_akhir[:2])

self.durasi = int(durasi)

self.hari = [int(h) for h in hari]

class Ruangan:

def \_\_init\_\_(self, kelas, jam\_mulai, jam\_akhir, hari):

self.kelas = kelas

self.jam\_mulai = int(jam\_mulai[:2])

self.jam\_akhir = int(jam\_akhir[:2])

self.hari = [int(h) for h in hari]

class varMatkul:

def \_\_init\_\_(self, jadwal, list\_ruangan):

self.kode = jadwal.kode

if (jadwal.kelas == '-'):

self.domain\_kelas = [(list\_ruangan[i].kelas) for i in range(len(list\_ruangan))]

else:

self.domain\_kelas = [jadwal.kelas]

self.domain\_jam\_mulai = [(jadwal.jam\_mulai+i) for i in range(jadwal.jam\_akhir-jadwal.jam\_mulai-jadwal.durasi+1)]

self.domain\_hari = jadwal.hari

self.kelas = self.domain\_kelas[random.randrange(0,len(self.domain\_kelas))]

self.jam\_mulai = self.domain\_jam\_mulai[random.randrange(0,len(self.domain\_jam\_mulai))]

self.hari = self.domain\_hari[random.randrange(0,len(self.domain\_hari))]

self.durasi = jadwal.durasi

def mutate(self, mutate\_chance):

if (int(random.random() \* mutate\_chance) == 1):

if (int(random.random() \* 2) == 1):

self.kelas = self.domain\_kelas[random.randrange(0,len(self.domain\_kelas))]

if (int(random.random() \* 2) == 1):

self.jam\_mulai = self.domain\_jam\_mulai[random.randrange(0,len(self.domain\_jam\_mulai))]

if (int(random.random() \* 2) == 1):

self.hari = self.domain\_hari[random.randrange(0,len(self.domain\_hari))]

def print\_jadwal(self):

print self.kode, self.kelas, self. jam\_mulai, self.jam\_mulai+self.durasi, self.hari

def createSpecies(list\_jadwal, list\_ruangan):

list\_matkul = []

for jadwal in list\_jadwal:

list\_matkul.append(varMatkul(jadwal, list\_ruangan))

return list\_matkul

def createPopulation(list\_jadwal, list\_ruangan):

numSpecies = 100

population = []

for i in range(numSpecies):

species = createSpecies(list\_jadwal, list\_ruangan)

population.append([species,checkConstraint(species, list\_ruangan)])

return population

# Validasi conflict untuk nyari neighboor

def conflictHariMatkul(matkul1, matkul2):

if (matkul1.hari == matkul2.hari):

return True

else:

return False

def conflictJamMatkul(matkul1, matkul2):

if (matkul1.jam\_mulai >= matkul2.jam\_mulai and matkul1.jam\_mulai < matkul2.jam\_mulai+matkul2.durasi) or (matkul2.jam\_mulai >= matkul1.jam\_mulai and matkul2.jam\_mulai < matkul1.jam\_mulai+matkul1.durasi):

return True

else:

return False

def conflictHariRuangan(matkul, ruangan):

if (matkul.hari not in ruangan.hari):

return True

else:

return False

def conflictJamRuangan(matkul, ruangan):

if (matkul.jam\_mulai < ruangan.jam\_mulai) or (matkul.jam\_mulai+matkul.durasi > ruangan.jam\_akhir):

return True

else:

return False

def checkConstraintMatkul(matkul1, matkul2):

constraintBroken = 0

if (matkul1.hari == matkul2.hari and matkul1.kelas == matkul2.kelas):

if (matkul1.jam\_mulai >= matkul2.jam\_mulai and matkul1.jam\_mulai < matkul2.jam\_mulai+matkul2.durasi):

constraintBroken = 1

if (matkul2.jam\_mulai >= matkul1.jam\_mulai and matkul2.jam\_mulai < matkul1.jam\_mulai+matkul1.durasi):

constraintBroken = 1

return constraintBroken

def checkConstraintRuang(matkul, list\_ruangan):

constraintBroken = 0

ruangan = 0

for ruang in list\_ruangan:

if (ruang.kelas == matkul.kelas):

ruangan = ruang

break

if (matkul.hari not in ruangan.hari):

constraintBroken = 1

else:

if (matkul.jam\_mulai < ruangan.jam\_mulai):

constraintBroken = 1

if (matkul.jam\_mulai+matkul.durasi > ruangan.jam\_akhir):

constraintBroken = 1

return constraintBroken

def checkConstraint(list\_matkul, list\_ruangan):

totalConstraintBroken = 0

for i in range(len(list\_matkul)):

totalConstraintBroken += checkConstraintRuang(list\_matkul[i],list\_ruangan)

for j in range(i+1, len(list\_matkul)):

totalConstraintBroken += checkConstraintMatkul(list\_matkul[i],list\_matkul[j])

return totalConstraintBroken

def getConstraintMatkul(matkul1, matkul2):

constraintBroken = []

if (matkul1.hari == matkul2.hari and matkul1.kelas == matkul2.kelas):

if (matkul1.jam\_mulai >= matkul2.jam\_mulai and matkul1.jam\_mulai < matkul2.jam\_mulai+matkul2.durasi):

if (matkul1.jam\_mulai + matkul1.durasi > matkul2.jam\_mulai + matkul2.durasi):

durasi = matkul1.durasi - (matkul1.jam\_mulai + matkul1.durasi) + (matkul2.jam\_mulai + matkul2.durasi)

else:

durasi = matkul1.durasi

for i in range(durasi):

constraintBroken.append([matkul1.hari, matkul1.jam\_mulai+i])

if (matkul2.jam\_mulai >= matkul1.jam\_mulai and matkul2.jam\_mulai < matkul1.jam\_mulai+matkul1.durasi):

if (matkul2.jam\_mulai + matkul2.durasi > matkul1.jam\_mulai + matkul1.durasi):

durasi = matkul2.durasi - (matkul2.jam\_mulai + matkul2.durasi) + (matkul1.jam\_mulai + matkul1.durasi)

else:

durasi = matkul2.durasi

for i in range(durasi):

constraintBroken.append([matkul2.hari, matkul2.jam\_mulai+i])

return constraintBroken

def getConstraintRuang(matkul, list\_ruangan):

constraintBroken = []

ruangan = 0

for ruang in list\_ruangan:

if (ruang.kelas == matkul.kelas):

ruangan = ruang

break

if (matkul.hari not in ruangan.hari):

for i in range(matkul.durasi):

constraintBroken.append([matkul.hari, matkul.jam\_mulai+i])

else:

if (matkul.jam\_mulai < ruangan.jam\_mulai):

for i in range(ruangan.jam\_mulai - matkul.jam\_mulai):

constraintBroken.append([matkul.hari, matkul.jam\_mulai+i])

if (matkul.jam\_mulai+matkul.durasi > ruangan.jam\_akhir):

for i in range(matkul.jam\_mulai+matkul.durasi-ruangan.jam\_akhir):

constraintBroken.append([matkul.hari, ruangan.jam\_akhir+i])

return constraintBroken

def getConstraint(list\_matkul, list\_ruangan):

totalConstraintRuangan = []

totalConstraintMatkul = []

for i in range(len(list\_matkul)):

totalConstraintRuangan += getConstraintRuang(list\_matkul[i],list\_ruangan)

for j in range(i+1, len(list\_matkul)):

totalConstraintMatkul += getConstraintMatkul(list\_matkul[i],list\_matkul[j])

return totalConstraintRuangan, totalConstraintMatkul

def copySpecies(species):

new\_species = []

for i in range(len(species)):

new\_species.append(copy.copy(species[i]))

return new\_species

#HILL CLIMBING

def highestValueNeighboor(list\_matkul, list\_ruangan):

successor = copySpecies(list\_matkul)

for i in range(len(list\_matkul)):

for j in range(i+1, len(list\_matkul)):

if checkConstraintMatkul(list\_matkul[i], list\_matkul[j]) != 0:

if conflictHariMatkul(list\_matkul[i], list\_matkul[j]):

while (successor[i].hari == successor[j].hari):

successor[i].hari = successor[i].domain\_hari[random.randrange(0,len(successor[i].domain\_hari))]

elif conflictJamMatkul(list\_matkul[i], list\_matkul[j]):

while (successor[i].jam\_mulai == successor[j].jam\_mulai):

successor[i].jam\_mulai = successor[i].domain\_jam\_mulai[random.randrange(0,len(successor[i].domain\_jam\_mulai))]

if checkConstraintRuang(successor[i], list\_ruangan) != 0:

ruangan = 0

for ruang in list\_ruangan:

if (ruang.kelas == list\_matkul[i].kelas):

ruangan = ruang

break

if conflictHariRuangan(list\_matkul[i], ruangan):

if sameValueinLists(ruangan.hari, successor[i].domain\_hari):

while (successor[i].hari not in ruangan.hari):

successor[i].hari = successor[i].domain\_hari[random.randrange(0,len(successor[i].domain\_hari))]

else:

successor[i].kelas = successor[i].domain\_kelas[random.randrange(0,len(successor[i].domain\_kelas))]

if conflictJamRuangan(successor[i], ruangan):

if ruangan.jam\_mulai in successor[i].domain\_jam\_mulai:

while conflictJamRuangan(successor[i], ruangan):

successor[i].jam\_mulai = successor[i].domain\_jam\_mulai[random.randrange(0,len(successor[i].domain\_jam\_mulai))]

else:

successor[i].kelas = successor[i].domain\_kelas[random.randrange(0,len(successor[i].domain\_kelas))]

return successor

#SA

def searchNeighbour(list\_matkul,list\_ruangan):

successor = copySpecies(list\_matkul)

i = random.randrange(0,(len(list\_matkul)))

successor[i].kelas = successor[i].domain\_kelas[random.randrange(0,len(successor[i].domain\_kelas))]

successor[i].hari = successor[i].domain\_hari[random.randrange(0,len(successor[i].domain\_hari))]

successor[i].jam\_mulai = successor[i].domain\_jam\_mulai[random.randrange(0,len(successor[i].domain\_jam\_mulai))]

return successor

def SHC(list\_matkul,list\_ruangan):

successor = copySpecies(list\_matkul)

i = random.randrange(0,(len(list\_matkul)))

j = random.randrange(0,(len(list\_matkul)))

while (i==j):

i = random.randrange(1,(len(list\_matkul)))

j = random.randrange(1,(len(list\_matkul)))

if checkConstraintMatkul(list\_matkul[i], list\_matkul[j]) != 0:

if conflictHariMatkul(list\_matkul[i], list\_matkul[j]):

successor[i].hari = successor[i].domain\_hari[random.randrange(0,len(successor[i].domain\_hari))]

elif conflictJamMatkul(list\_matkul[i], list\_matkul[j]):

while (successor[i].jam\_mulai == successor[j].jam\_mulai):

successor[i].jam\_mulai = successor[i].domain\_jam\_mulai[random.randrange(0,len(successor[i].domain\_jam\_mulai))]

if checkConstraintRuang(successor[i], list\_ruangan) != 0:

ruangan = 0

for ruang in list\_ruangan:

if (ruang.kelas == list\_matkul[i].kelas):

ruangan = ruang

break

if conflictHariRuangan(list\_matkul[i], ruangan):

if sameValueinLists(ruangan.hari, successor[i].domain\_hari):

while (successor[i].hari not in ruangan.hari):

successor[i].hari = successor[i].domain\_hari[random.randrange(0,len(successor[i].domain\_hari))]

else:

successor[i].kelas = successor[i].domain\_kelas[random.randrange(0,len(successor[i].domain\_kelas))]

if conflictJamRuangan(successor[i], ruangan):

if ruangan.jam\_mulai in successor[i].domain\_jam\_mulai:

while conflictJamRuangan(successor[i], ruangan):

successor[i].jam\_mulai = successor[i].domain\_jam\_mulai[random.randrange(0,len(successor[i].domain\_jam\_mulai))]

else:

successor[i].kelas = successor[i].domain\_kelas[random.randrange(0,len(successor[i].domain\_kelas))]

return successor

def probability(E,Ei,T):

p = math.exp(-(E-Ei)/T)

p = abs(p)

p=p\*100

if (p>100):

p = 100

return p

#GA

def sameValueinLists(list1, list2):

for x in list1:

if x in list2:

return True

return False

def fitnessTest(populasi):

fitSpecies = []

populasi.sort(key=lambda x: x[1])

for i in range(10):

species = [copySpecies(populasi[i][0]),populasi[i][1]]

fitSpecies.append(species)

return fitSpecies

def crossing(species1, species2):

lenr = random.randrange(0, len(species1))

nspecies1 = copySpecies(species1[:lenr]) + copySpecies(species2[lenr:])

nspecies2 = copySpecies(species2[:lenr]) + copySpecies(species1[lenr:])

return nspecies1, nspecies2

def mutation(population, list\_ruangan):

mutation\_chance = len(population)

for i in range(len(population)):

for j in range(len(population[i][0])):

population[i][0][j].mutate(mutation\_chance)

population[i][1] = checkConstraint(population[i][0], list\_ruangan)

def readFile(filename):

listRuangan = []

listJadwal = []

with open(filename) as file:

r = 0

for line in file:

if (line == 'Ruangan\n'):

r = 1

elif (line == 'Jadwal\n'):

r = 2

elif (line == '\n'):

pass

elif (r == 1):

ltemp = line[:-1].split(';')

ruang = Ruangan(ltemp[0],ltemp[1],ltemp[2],ltemp[3].split(','))

listRuangan.append(ruang)

elif (r == 2):

ltemp = line[:-1].split(';')

jadwal = Jadwal(ltemp[0],ltemp[1],ltemp[2],ltemp[3],ltemp[4],ltemp[5].split(','))

listJadwal.append(jadwal)

return listRuangan, listJadwal

# MainTable Interface

class MainTable(QTableWidget):

def \_\_init\_\_(self, parent=None):

super(MainTable,self).\_\_init\_\_(parent)

self.setRowCount(11)

self.setColumnCount(5)

self.setHorizontalHeaderLabels(['Senin','Selasa','Rabu','Kamis','Jumat'])

self.horizontalHeader().setResizeMode(QHeaderView.Stretch)

self.verticalHeader().setResizeMode(QHeaderView.Stretch)

self.setVerticalHeaderLabels(['07:00','08:00','09:00','10:00','11:00','12:00','13:00','14:00','15:00','16:00','17:00'])

self.setDragDropMode(QAbstractItemView.InternalMove)

# SubTable Interface

class SubTableWidget(QTableWidget):

def \_\_init\_\_(self, parent=None):

super(SubTableWidget,self).\_\_init\_\_(parent)

layout = QHBoxLayout()

layout.setContentsMargins(0,0,0,0)

layout.setSpacing(10)

self.setRowCount(0)

self.setColumnCount(1)

self.horizontalHeader().setResizeMode(QHeaderView.Stretch)

self.verticalHeader().setResizeMode(QHeaderView.Stretch)

self.horizontalHeader().hide()

self.verticalHeader().hide()

self.setEditTriggers(QAbstractItemView.NoEditTriggers)

self.setDragEnabled(True)

self.setAcceptDrops(True)

self.setDragDropOverwriteMode(False)

self.last\_drop\_row = None

self.setLayout(layout)

# Override this method to get the correct row index for insertion

def dropMimeData(self, row, col, mimeData, action):

self.last\_drop\_row = row

return True

def dropEvent(self, event):

sender = event.source()

super(SubTableWidget,self).dropEvent(event)

dropRow = self.last\_drop\_row

selectedRows = sender.getselectedRowsFast()

for \_ in selectedRows:

self.insertRow(dropRow)

sel\_rows\_offsets = [0 if self != sender or srow < dropRow else len(selectedRows) for srow in selectedRows]

selectedRows = [row + offset for row, offset in zip(selectedRows, sel\_rows\_offsets)]

for i, srow in enumerate(selectedRows):

for j in range(self.columnCount()):

item = sender.item(srow, j)

if item:

source = QTableWidgetItem(item)

self.setItem(dropRow + i, j, source)

for srow in reversed(selectedRows):

sender.removeRow(srow)

event.accept()

def getselectedRowsFast(self):

selectedRows = []

for item in self.selectedItems():

if item.row() not in selectedRows:

selectedRows.append(item.row())

selectedRows.sort()

return selectedRows

def addItem(self, newitem):

rowPosition = self.rowCount()

self.insertRow(rowPosition)

self.setItem(rowPosition, 0, newitem)

# Main Graphic User Interface

class MainWindow(QWidget):

def \_\_init\_\_(self, parent = None):

super(MainWindow, self).\_\_init\_\_(parent)

# Layout Type

grid = QGridLayout()

grid.setSpacing(10)

# Font

font = QFont()

font.setPointSize(8)

# Define Main Table

self.table = MainTable()

grid.addWidget(self.table, 0, 0, 1, 3)

# Run Push Button

self.runbut = QPushButton('Run!')

grid.addWidget(self.runbut, 5, 1)

# Window Layout Modifier

self.setLayout(grid)

self.setGeometry(200, 75, 900, 900)

self.setWindowTitle("Scheduling Mata Kuliah")

self.show()

# Radio Buttons

self.groupBox = QGroupBox("Radio Buttons")

self.rb1 = QRadioButton('Hill Algorithm')

self.rb1.setFont(font)

self.rb1.clicked.connect(self.onRadioButton1)

self.indikator = 0 #indikator

self.rb2 = QRadioButton('Simulated Annealing')

self.rb2.setFont(font)

self.rb2.clicked.connect(self.onRadioButton2)

self.rb3 = QRadioButton('Genetic Algorithm')

self.rb3.setFont(font)

self.rb3.clicked.connect(self.onRadioButton3)

grid.addWidget(self.rb1, 2, 1)

grid.addWidget(self.rb2, 3, 1)

grid.addWidget(self.rb3, 4, 1)

# Run Button Clicked

self.runbut.clicked.connect(self.setTableContentIndicated)

# Label Misc.

self.choosealg = QLabel('Choose Method:')

self.choosealg.setFont(font)

grid.addWidget(self.choosealg, 1, 1)

self.consbroken = QLabel('Constraint Broken: ')

self.consbroken.setFont(font)

grid.addWidget(self.consbroken, 5, 2)

self.efektivitas = QLabel('Efektivitas: ')

self.efektivitas.setFont(font)

grid.addWidget(self.efektivitas, 4,2)

# List of Color

self.licolor = ['red','blue','darkMagenta','darkGreen','magenta','green','darkCyan','darkBlue','cyan','yellow']

# Check Radio Button Checked, change Indikator for Algorithm Choosing

def onRadioButton1(self):

print "BUTTON 1"

self.indikator = 1

def onRadioButton2(self):

print "BUTTON 2"

self.indikator = 2

def onRadioButton3(self):

print "BUTTON 3"

self.indikator = 3

# Set Table Indikator and Run

def setTableContentIndicated(self):

print self.indikator

if self.indikator == 1:

self.setTableContentHill()

elif self.indikator == 2:

self.setTableContentSA()

elif self.indikator == 3:

self.setTableContentGA()

# Coloring

def Coloring(self, varMatkul, soda, i):

item = QTableWidgetItem(soda)

item.setTextColor(QColor(self.licolor[i]))

return item

# Coloring 2

def Coloring2(self, varMatkul, soda, i):

item = QTableWidgetItem(soda)

item.setTextColor(QColor(self.licolor[i]))

return item

# Fill the Table with Data according to Algorithm selected

def setTableContentGA(self):

self.table.clear()

self.table.setHorizontalHeaderLabels(['Senin','Selasa','Rabu','Kamis','Jumat'])

self.table.setVerticalHeaderLabels(['07:00','08:00','09:00','10:00','11:00','12:00','13:00','14:00','15:00','16:00','17:00'])

varMatkul, constraint, lcRuang, lcMatkul, totalRuang = geneticAlgorithm()

listconstraint = lcMatkul + lcRuang

lcu = []

for elem in lcMatkul:

if elem not in lcu:

lcu.append(elem)

jamTerpakai = 0

#Inisiasi

for i in range (0,11):

for j in range (0,5):

self.minitable = SubTableWidget()

self.table.setCellWidget(i, j, self.minitable)

for i in range (len(varMatkul[0])):

jammulai = varMatkul[0][i].jam\_mulai

start = abs(7-jammulai)

lama = varMatkul[0][i].durasi

if (varMatkul[0][i].hari == 1):

for j in range (0, lama):

soda = varMatkul[0][i].kode + ' - ' + varMatkul[0][i].kelas

item = self.Coloring(varMatkul, soda, i)

if (self.table.cellWidget(start, 0).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 0, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,0)

self.minitable.addItem(QTableWidgetItem(item))

start = start+1

jamTerpakai += 1

elif (varMatkul[0][i].hari == 2):

for j in range (0,lama):

soda = varMatkul[0][i].kode + ' - ' + varMatkul[0][i].kelas

item = self.Coloring(varMatkul, soda, i)

if (self.table.cellWidget(start, 1).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 1, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,1)

self.minitable.addItem(QTableWidgetItem(item))

start = start+1

jamTerpakai += 1

elif (varMatkul[0][i].hari == 3):

for j in range (0,lama):

soda = varMatkul[0][i].kode + ' - ' + varMatkul[0][i].kelas

item = self.Coloring(varMatkul, soda, i)

if (self.table.cellWidget(start, 2).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 2, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,2)

self.minitable.addItem(QTableWidgetItem(item))

start = start+1

jamTerpakai += 1

elif (varMatkul[0][i].hari == 4):

for j in range (0,lama):

soda = varMatkul[0][i].kode + ' - ' + varMatkul[0][i].kelas

item = self.Coloring(varMatkul, soda, i)

if (self.table.cellWidget(start, 3).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 3, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,3)

self.minitable.addItem(QTableWidgetItem(item))

start = start+1

jamTerpakai += 1

elif (varMatkul[0][i].hari == 5):

for j in range (0,lama):

soda = varMatkul[0][i].kode + ' - ' + varMatkul[0][i].kelas

item = self.Coloring(varMatkul, soda, i)

if (self.table.cellWidget(start, 4).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 4, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,4)

self.minitable.addItem(QTableWidgetItem(item))

start = start+1

jamTerpakai += 1

self.consbroken.setText('Constraint Broken: ' + str(constraint))

jamTerpakai = jamTerpakai-len(listconstraint)+len(lcu)

self.efektivitas.setText('Efektivitas: ' + str(jamTerpakai\*1.0/totalRuang))

print 'Efektivitas: ' + str(jamTerpakai\*1.0/totalRuang)

z = 0;

for c in range (0, len(listconstraint)):

self.x = listconstraint[z][0] #day

self.y = listconstraint[z][1] #hour

print listconstraint

A = self.table.cellWidget(self.y-7, self.x-1)

for d in range (0, A.rowCount()):

B = A.item(d,0)

B.setBackgroundColor(QColor(255,255,0))

z = z + 1

def setTableContentSA(self):

self.table.clear()

self.table.setHorizontalHeaderLabels(['Senin','Selasa','Rabu','Kamis','Jumat'])

self.table.setVerticalHeaderLabels(['07:00','08:00','09:00','10:00','11:00','12:00','13:00','14:00','15:00','16:00','17:00'])

varMatkul, constraint, lcRuang, lcMatkul, totalRuang = simulatedAnnealing()

listconstraint = lcMatkul + lcRuang

lcu = []

for elem in lcMatkul:

if elem not in lcu:

lcu.append(elem)

jamTerpakai = 0

#Inisiasi

for i in range (0,11):

for j in range (0,5):

self.minitable = SubTableWidget()

self.table.setCellWidget(i, j, self.minitable)

for i in range (len(varMatkul)):

jammulai = varMatkul[i].jam\_mulai

start = abs(7-jammulai) #biar indexnya pas

lama = varMatkul[i].durasi #banyaknya pengulangan buat nyetak

if (varMatkul[i].hari == 1):

for j in range (0, lama):

soda = varMatkul[i].kode + ' - ' + varMatkul[i].kelas

item = self.Coloring2(varMatkul, soda, i)

if (self.table.cellWidget(start, 0).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 0, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,0)

self.minitable.addItem(QTableWidgetItem(item))

jamTerpakai += 1

start = start+1

elif (varMatkul[i].hari == 2):

for j in range (0,lama):

soda = varMatkul[i].kode + ' - ' + varMatkul[i].kelas

item = self.Coloring2(varMatkul, soda, i)

if (self.table.cellWidget(start, 1).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 1, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,1)

self.minitable.addItem(QTableWidgetItem(item))

jamTerpakai += 1

start = start+1

elif (varMatkul[i].hari == 3):

for j in range (0,lama):

soda = varMatkul[i].kode + ' - ' + varMatkul[i].kelas

item = self.Coloring2(varMatkul, soda, i)

if (self.table.cellWidget(start, 2).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 2, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,2)

self.minitable.addItem(QTableWidgetItem(item))

jamTerpakai += 1

start = start+1

elif (varMatkul[i].hari == 4):

for j in range (0,lama):

soda = varMatkul[i].kode + ' - ' + varMatkul[i].kelas

item = self.Coloring2(varMatkul, soda, i)

if (self.table.cellWidget(start, 3).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 3, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,3)

self.minitable.addItem(QTableWidgetItem(item))

jamTerpakai += 1

start = start+1

elif (varMatkul[i].hari == 5):

for j in range (0,lama):

soda = varMatkul[i].kode + ' - ' + varMatkul[i].kelas

item = self.Coloring2(varMatkul, soda, i)

if (self.table.cellWidget(start, 4).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 4, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,4)

self.minitable.addItem(QTableWidgetItem(item))

start = start+1

jamTerpakai += 1

self.consbroken.setText('Constraint Broken: ' + str(constraint))

jamTerpakai = jamTerpakai-len(listconstraint)+len(lcu)

self.efektivitas.setText('Efektivitas: ' + str(jamTerpakai\*1.0/totalRuang))

print 'Efektivitas: ' + str(jamTerpakai\*1.0/totalRuang)

z = 0;

for c in range (0, len(listconstraint)):

self.x = listconstraint[z][0] #day

self.y = listconstraint[z][1] #hour

print listconstraint

A = self.table.cellWidget(self.y-7, self.x-1)

for d in range (0, A.rowCount()):

B = A.item(d,0)

B.setBackgroundColor(QColor(255,255,0))

z = z + 1

def setTableContentHill(self):

self.table.clear()

self.table.setHorizontalHeaderLabels(['Senin','Selasa','Rabu','Kamis','Jumat'])

self.table.setVerticalHeaderLabels(['07:00','08:00','09:00','10:00','11:00','12:00','13:00','14:00','15:00','16:00','17:00'])

varMatkul, constraint, lcRuang, lcMatkul, totalRuang = hillAlgorithm()

listconstraint = lcMatkul + lcRuang

lcu = []

for elem in lcMatkul:

if elem not in lcu:

lcu.append(elem)

jamTerpakai = 0

#Inisiasi

for i in range (0,11):

for j in range (0,5):

self.minitable = SubTableWidget()

self.table.setCellWidget(i, j, self.minitable)

for i in range (len(varMatkul)):

jammulai = varMatkul[i].jam\_mulai

start = abs(7-jammulai) #biar indexnya pas

lama = varMatkul[i].durasi #banyaknya pengulangan buat nyetak

if (varMatkul[i].hari == 1):

for j in range (0, lama):

soda = varMatkul[i].kode + ' - ' + varMatkul[i].kelas

item = self.Coloring2(varMatkul, soda, i)

if (self.table.cellWidget(start, 0).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 0, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,0)

self.minitable.addItem(QTableWidgetItem(item))

jamTerpakai += 1

start = start+1

elif (varMatkul[i].hari == 2):

for j in range (0,lama):

soda = varMatkul[i].kode + ' - ' + varMatkul[i].kelas

item = self.Coloring2(varMatkul, soda, i)

if (self.table.cellWidget(start, 1).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 1, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,1)

self.minitable.addItem(QTableWidgetItem(item))

jamTerpakai += 1

start = start+1

elif (varMatkul[i].hari == 3):

for j in range (0,lama):

soda = varMatkul[i].kode + ' - ' + varMatkul[i].kelas

item = self.Coloring2(varMatkul, soda, i)

if (self.table.cellWidget(start, 2).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 2, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,2)

self.minitable.addItem(QTableWidgetItem(item))

jamTerpakai += 1

start = start+1

elif (varMatkul[i].hari == 4):

for j in range (0,lama):

soda = varMatkul[i].kode + ' - ' + varMatkul[i].kelas

item = self.Coloring2(varMatkul, soda, i)

if (self.table.cellWidget(start, 3).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 3, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,3)

self.minitable.addItem(QTableWidgetItem(item))

jamTerpakai += 1

start = start+1

elif (varMatkul[i].hari == 5):

for j in range (0,lama):

soda = varMatkul[i].kode + ' - ' + varMatkul[i].kelas

item = self.Coloring2(varMatkul, soda, i)

if (self.table.cellWidget(start, 4).rowCount() == 0):

self.minitable = SubTableWidget()

self.minitable.insertRow(0)

self.table.setCellWidget(start, 4, self.minitable)

self.minitable.setItem(0, 0, QTableWidgetItem(item))

else:

self.minitable = self.table.cellWidget(start,4)

self.minitable.addItem(QTableWidgetItem(item))

jamTerpakai += 1

start = start+1

self.consbroken.setText('Constraint Broken: ' + str(constraint))

jamTerpakai = jamTerpakai-len(listconstraint)+len(lcu)

self.efektivitas.setText('Efektivitas: ' + str(jamTerpakai\*1.0/totalRuang))

print 'Efektivitas: ' + str(jamTerpakai\*1.0/totalRuang)

z = 0;

for c in range (0, len(listconstraint)):

self.x = listconstraint[z][0] #day

self.y = listconstraint[z][1] #hour

print listconstraint

A = self.table.cellWidget(self.y-7, self.x-1)

for d in range (0, A.rowCount()):

B = A.item(d,0)

B.setBackgroundColor(QColor(255,255,0))

z = z + 1

# File Open Window Dialogue

def choose\_file(self):

file\_name = QFileDialog.getOpenFileName(self, "Open File", "", "Text document (\*.txt)")

# GA Selected

def geneticAlgorithm():

print("GA")

listRuangan, listJadwal = readFile('Testcase.txt')

fitSpecies = []

for i in range(50):

population = createPopulation(listJadwal, listRuangan)

population += fitSpecies

fitSpecies = fitnessTest(population)

fittestSpecies = [copySpecies(fitSpecies[0][0]),fitSpecies[0][1]]

population = []

for i in range(len(fitSpecies)):

for j in range(i,len(fitSpecies)):

tempSpecies1, tempSpecies2 = crossing(fitSpecies[i][0],fitSpecies[j][0])

population.append([tempSpecies1, checkConstraint(tempSpecies1, listRuangan)])

population.append([tempSpecies2, checkConstraint(tempSpecies2, listRuangan)])

population.append(fittestSpecies)

population.sort(key=lambda x: x[1])

fittestSpecies = [copySpecies(population[0][0]),population[0][1]]

mutation(population, listRuangan)

population.append(fittestSpecies)

population.sort(key=lambda x: x[1])

fittestSpecies = [copySpecies(population[0][0]),population[0][1]]

fitSpecies = fitnessTest(population)

for i in range(len(fittestSpecies[0])):

fittestSpecies[0][i].print\_jadwal()

print 'Constraint broken: ' + str(checkConstraint(fittestSpecies[0], listRuangan))

constraint = checkConstraint(fittestSpecies[0], listRuangan)

lcRuang, lcMatkul = getConstraint(fittestSpecies[0], listRuangan)

totalRuang = 0

for i in range(len(listRuangan)):

totalRuang += (listRuangan[i].jam\_akhir-listRuangan[i].jam\_mulai) \* len(listRuangan[i].hari)

return fittestSpecies, constraint, lcRuang, lcMatkul, totalRuang

# Hill Selected

def hillAlgorithm():

print("HILL")

listRuangan, listJadwal = readFile('Testcase.txt')

problem = createSpecies(listJadwal, listRuangan)

current = copySpecies(problem)

neighboor = highestValueNeighboor(current, listRuangan)

while(checkConstraint(current, listRuangan) > checkConstraint(neighboor, listRuangan)):

current = copySpecies(neighboor)

neighboor = highestValueNeighboor(current, listRuangan)

problem = copySpecies(current)

for i in range(len(problem)):

problem[i].print\_jadwal()

print 'Constraint broken: ' + str(checkConstraint(problem, listRuangan))

constraint = checkConstraint(problem, listRuangan)

lcRuang, lcMatkul = getConstraint(problem, listRuangan)

totalRuang = 0

for i in range(len(listRuangan)):

totalRuang += (listRuangan[i].jam\_akhir-listRuangan[i].jam\_mulai) \* len(listRuangan[i].hari)

return problem, constraint, lcRuang, lcMatkul, totalRuang

# SA Selected

def simulatedAnnealing():

print("SA")

T = 100

listRuangan, listJadwal = readFile('Testcase.txt')

current = createSpecies(listJadwal, listRuangan)

neighbour = searchNeighbour(current, listRuangan)

if (checkConstraint(current,listRuangan)>=checkConstraint(neighbour,listRuangan)):

best = copySpecies(neighbour)

current = copySpecies(neighbour)

else:

best = copySpecies(current)

#mulai SA algorithm

#Random Walk

while (T>0.0001) and (checkConstraint(best,listRuangan)!=0):

neighbour=searchNeighbour(current,listRuangan)

E = checkConstraint(current,listRuangan)

Ei = checkConstraint(neighbour,listRuangan)

if (E>=Ei):

if (checkConstraint(best,listRuangan)>=checkConstraint(neighbour,listRuangan)):

best = copySpecies(neighbour)

current = copySpecies(neighbour)

elif(T!=0): #kalo bad move

p = probability(E,Ei,T)

r = operator.div(100,p)

if (r!=1):

r = random.randrange(1,r)

if (r==1):

current = copySpecies(neighbour)

T = T-0.01

#memulai Stochastic Hill Climbing

if (checkConstraint(best,listRuangan)>0):

neighbour = SHC(best,listRuangan)

while (checkConstraint(best,listRuangan)>(checkConstraint(neighbour,listRuangan))):

best = copySpecies(neighbour)

neighbour = SHC(best,listRuangan)

for i in range(len(best)):

best[i].print\_jadwal()

print 'Constraint broken: ' + str(checkConstraint(best,listRuangan))

constraint = checkConstraint(best, listRuangan)

lcRuang, lcMatkul = getConstraint(best, listRuangan)

totalRuang = 0

for i in range(len(listRuangan)):

totalRuang += (listRuangan[i].jam\_akhir-listRuangan[i].jam\_mulai) \* len(listRuangan[i].hari)

return best, constraint, lcRuang, lcMatkul, totalRuang

# Main Program

def main():

#Window

app = QApplication(sys.argv)

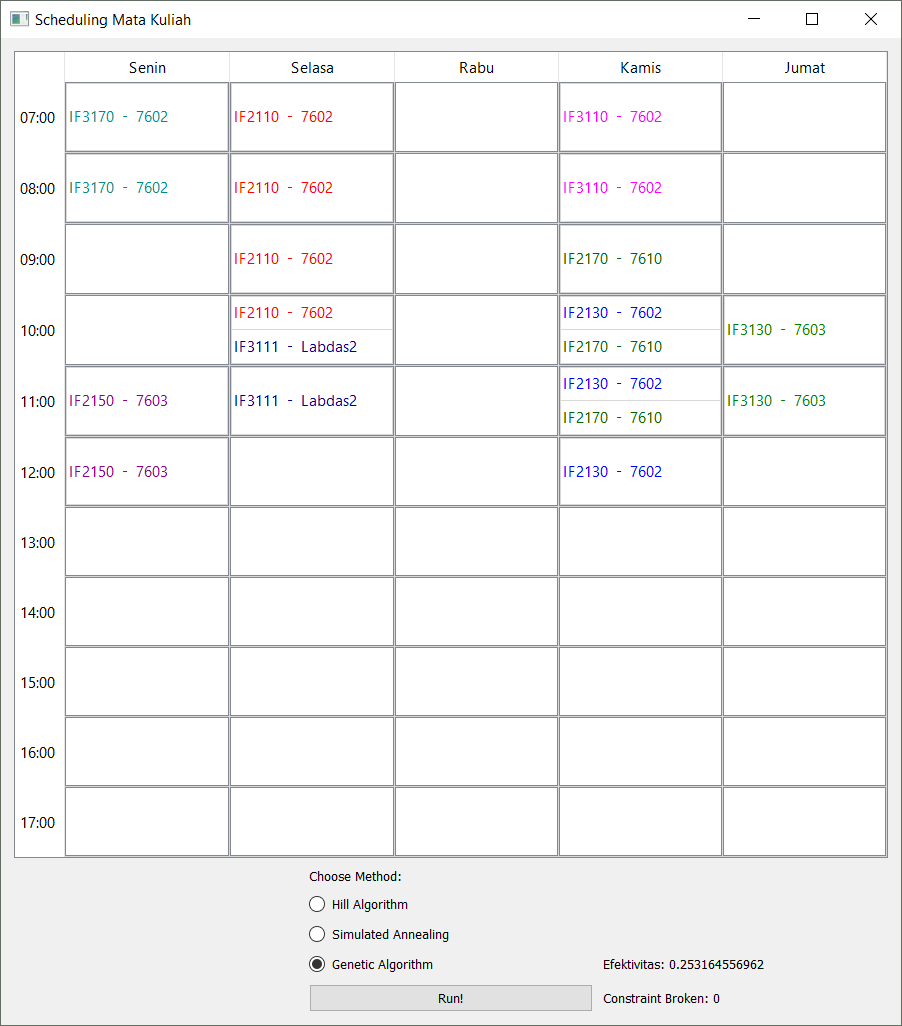
ex = MainWindow()

sys.exit(app.exec\_())

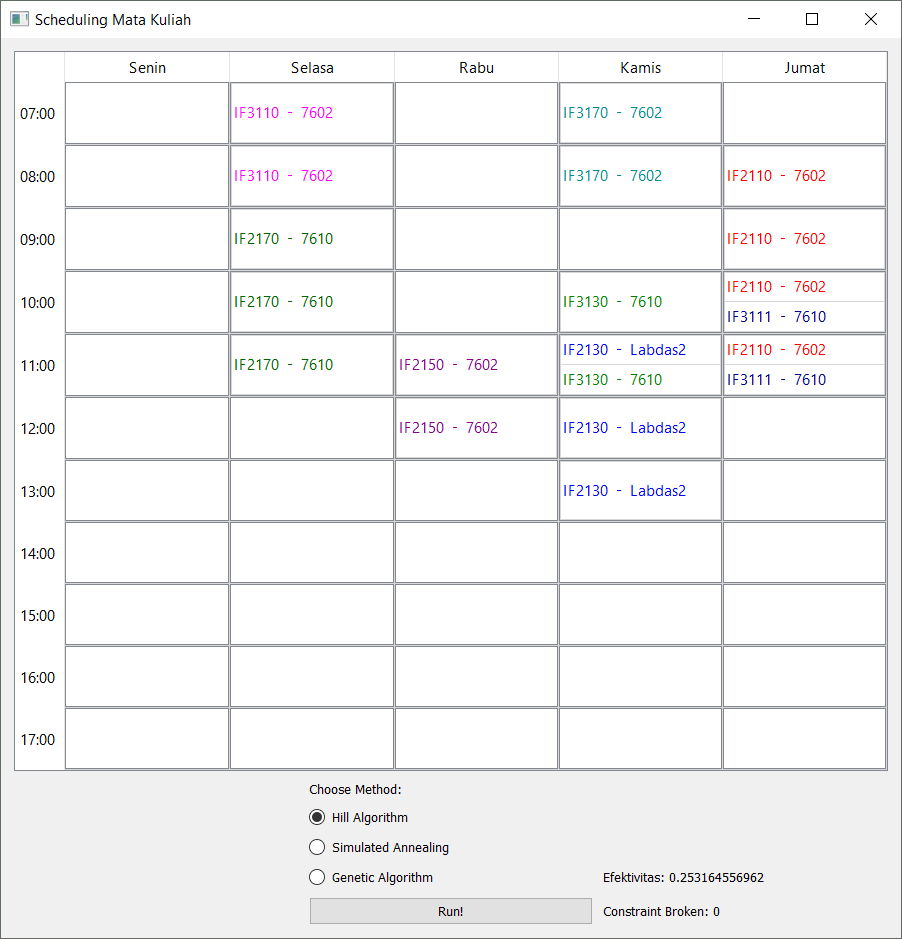
if \_\_name\_\_ == '\_\_main\_\_':

main()

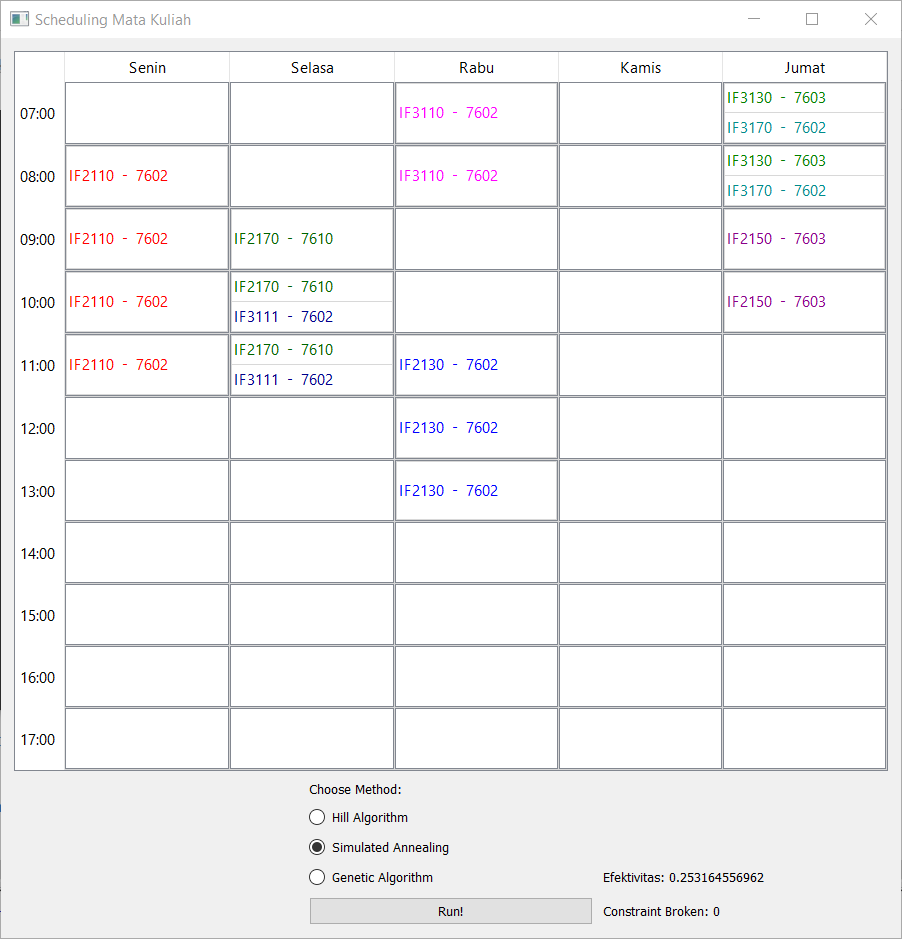
1. **Hasil Program**



**Generic Algorithm**



**Hill Algorithm**

****

**Simulated Annealing**

**BAB IV**

**DAFTAR REFERENSI**

* http://kuliah.itb.ac.id
* Struat, Russell, Peter Norvig. 2003. *Artificial Intelligence A Moderen Approach (Thrid Edition). New Jersey: Pearson Education, Inc.*