# AFIT SCP Python

# Python Script

import os import random import re

import subprocess

from statistics import mean import timeit

import networkx as nx

*# Regex’s to find solutions in AFIT SCP output*

solution Reg Ex = re. compile( r’ Theu Solution u Set: u \[((?:[0 -9]+ ,\ s)\*[0 -9]+) \]’) costReg Ex = re. compile( r’ Totalu cost: u ([0 -9]+)$’)

*# generic wrapper to use timeit with functions that*

*# require arguments*

def timeitWrapper ( function , \* args , \*\* kwargs): def wrapped ():

return function (\* args , \*\* kwargs) return wrapped

*# generic function that returns time elapsed of timeit*

def timeitFunction ( repNum , function , \* args , \*\* kwargs): funcWrapper = timeitWrapper ( function , \* args , \*\* kwargs) run Time = timeit. timeit( funcWrapper , number= rep Num ) return run Time

*# Create a random instance of our problem domain based on parameters*

*#[0] - Graph Size*

*#[1] - Max Degree*

*#[2] - Min Degree*

*#[3] - cost min*

*#[4] - cost max*

def createSCP ( paramsIn ): graph Size = paramsIn [0] max Connections = paramsIn [1] min Connections = paramsIn [2] costMin = paramsIn [3] costMax = paramsIn [4]

*# Generate random degree sequence and make sure the sequence can create a graph*

sequenceList = []

while not nx. is\_ graphical ( sequenceList ) or len ( sequenceList ) == 0: sequenceList = []

for seq Count in range (0 , graph Size ):

sequenceList . append ( random . randint( min Connections , m ax Connections )) graph = nx. random \_ degree\_ sequence\_ graph ( sequenceList , tries = 20)

graph = graph . to\_ directed ()

adjList = nx. generate\_ adjlist ( graph )

*# create total set and the subsets*

totalSet = []

totalSet. extend ( range (0 , graph Size))

subsets = []

for line in adjList:

neighbors = list( map ( int , line. split () [1:])) subsetCost = random . randint( costMin , costMax ) temp Item = [ neighbors , subsetCost ]

subsets. append ( temp Item )

return totalSet , subsets

*# write random problem into AFIT SCP format*

def writeSCP ( totalSetIn , subsetsIn , fileNameIn ):

setCoverStr = []

setCoverStr . append ("{{ u {0} u}}". format(" u". join ( map ( str , totalSetIn )))) setCoverStr . append ("")

setCoverStr . append ("{") for subset in subsetsIn :

setCoverStr . append (" uu ({{ u {0} u }} ,{1})". format(" u". join ( map ( str , subset [0])), subset

*'* [1]))

*→*

setCoverStr . append ("}")

with open ( fileNameIn , ’w ’) as outFile: for line in setCoverStr :

outFile. write( line + ’\ n’)

*# run AFIT SCP*

def run SCP ( prob FileName , resultFileName , program FileN am e ): try :

dir\_ path = os. path . dirname( os. path . realpath (\_\_ file\_\_ ))

subprocess . check\_ output ([" scp Problem . bat", dir\_path , prob FileName , resultFileName ,

*'* program FileN am e ])

*→*

except subprocess . C alled ProcessError : i = 0

*# find results in AFIT SCP output*

def analyzeResults ( resultFileNam e ): dataDict = {}

setList = [] cost = 0

solution = False

with open ( resultFileName , ’r’) as fileIn : for line in fileIn :

if not solution :

solution Match = solution Reg Ex . match ( line) if solution Match :

solution = True

setList = list( map ( int , solution Match . group (1). split(",u")))

else:

costMatch = costReg Ex . match ( line) if costMatch :

cost = int( costMatch . group (1))

dataDict[" solution Set "] = setList dataDict[" dronesPicked "] = len ( setList) dataDict[" cost"] = cost

return dataDict

*# create 1 random problem and solve it using AFIT SCP*

*# returns data from AFIT SCP*

def testSCP ( paramsIn ):

prob FileName = " random SetCover . txt" resultFileName = " scp Result. txt" totalSet , subsets = createSCP ( paramsIn )

writeSCP ( totalSet , subsets , prob FileName )

run Data = {}

programs = [" defaultSC PSolver . jar", " h 1 SCPSolver . jar", " h 2 SCPSolver . jar"] for program in programs:

run Time = timeitFunction (1 , runSCP , prob FileName , program [0: -4] + resultFileName ,

*'* program )

*→*

dataDict = analyzeResults ( program [0: -4] + resultFileName ) dataDict[" run Time"] = run Time

run Data[ program ] = dataDict return run Data

*# perform multiple iterations of testSCP*

*# return aggregate data*

def repeatSCP ( repeatNum , paramsIn ):

totalData = {}

for testNum in range (0 , repeatNum ): testDict = testSCP ( paramsIn )

if testNum == 0:

for key in testDict: totalData[ key ] = {}

for sub Key in testDict[ key ]: totalData[ key ][ sub Key ] = []

for key in testDict:

for sub Key in testDict[ key ]:

totalData[ key ][ sub Key ]. append ( testDict[ key ][ sub Key ]) finalData = {}

for key in totalData : finalData[ key ] = {}

finalData[ key ][" avg D ronesPicked "] = mean ( totalData [ key ][" dronesPicked "]) finalData[ key ][" avg Cost"] = mean ( totalData [ key ][" cost"])

finalData[ key ][" avg Run Time "] = mean ( totalData [ key ][" run Time"]) return finalData

def updateStatus ( printTxt):

with open (" status. txt", ’w ’) as outFile:

outFile. write("{0} u Complete .\ n". format( printTxt))

*# test cases*

*# SCP Parameters:*

*# Number of drones/ subsets , max comm links , min comm links*

small = [20 , 5 , 2 , 1 , 1]

medium = [40 , 5 , 2 , 1 , 1]

large = [60 , 5 , 2 , 1 , 1]

largeSparse = [60 , 3 , 1 , 1 , 1]

largeDense = [60 , 6 , 4 , 1 , 1]

print(" Small:")

print( repeatSCP (100 , small)) updateStatus (" Small")

print(" Medium :")

print( repeatSCP (100 , medium )) updateStatus (" Medium ")

print(" Largeu Sparse:")

print( repeatSCP (20 , largeSparse )) updateStatus (" Largeu Sparse")

print(" Large:")

print( repeatSCP (20 , large)) updateStatus (" Large")

print(" Largeu Dense:")

print( repeatSCP (20 , largeDense )) updateStatus (" Largeu Dense")

weighted Large = [60 , 5 , 2 , 1 , 10]

w eighted LargeSparse = [60 , 3 , 1 , 1 , 10]

w eighted LargeD ense = [60 , 6 , 4 , 1 , 10]

print(" Weighted u Largeu Sparse:")

print( repeatSCP (20 , w eighted LargeSparse )) updateStatus (" Weighted u Largeu Sparse")

print(" Weighted u Large:")

print( repeatSCP (20 , weighted Large )) updateStatus (" Large")

print(" Weighted u Largeu Dense:")

print( repeatSCP (20 , w eighted LargeD ense )) updateStatus (" Weighted u Largeu Dense")

# Appendix B: Batch Script

java - classpath % 4 edu . afit. scpsolver. Main % 1\% 2 % 1\% 3

# Appendix C: Heuristics

// All code designed by Walter Hayden , 2006

solveSCP () {

Tableau t;

// Fill in Tableau

// sorts the elements within each block

// heuristic 1

t. sortFamilies ( new C ostPerElem entFunction ());

// heuristic 2

t. sortFamilies ( new Elem entsPerFam ily Function ());

// sorts the blocks

t. sortBlocks ( new Elem entsPerB lock Function ());

// Start search for optimal solution

}

/\*\*

\* @ author Walter

\*/

public class C ostPerElem entFunction implements Comparator < Family >

{

/\*\*

\* Used to sort families by cost.

\*/

public C ostPerElem entFunction ()

{

super ();

}

public int compare( Family f1 , Family f2 )

{

return f1 . getC ostPerElem ent () - f2 . getC ostPerElem ent ();

}

@ Override

public String to String ()

{

return " C ostPerElem entFunction : u Sorted u familiesu( j) u from u leastu costu peru

}

*'→* elementutou greatestu costu peru element.";

}

/\*\*

\* @ author Walter

\*

\*/

public class Elem entsPerFam ily Function implements Comparator < Family >

{

/\*\*

\* Used to sort families by cost.

\*/

public Elem entsPerFam ily Function ()

{

super ();

}

public int compare( Family f1 , Family f2 )

{

return f2 . getRSet (). getR (). size () - f1 . getRSet (). getR (). size ();

}

@ Override

public String to String ()

{

return " Elem entsPerFam ily Function : u Sorted u familiesu( j) u from u greatestu

}

*'→* cardinality utou leastu cardinality .";

}