```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 31 15:27:01 2019

@author: hernan
LIA - MODO International Lab
EA Seminar
Version 1.0 31-07-2019

Some statistics for multi-objective algorithms
Asumes results are kept under folder outfname
    outfname
        run1
            pop_g0.txt
            pop_g1.txt
            ...
        run2
        ...

The format of pop_gx.txt file is as follows
v_1 v_2 ... v_nvar f_1 f_2 ... f_nobj r_0 r_1
where
  v indicated a variable
  f indicates fitness
  r indicates rank

Fitness values are located at positions nvar, nvar+1 ... nvar+nobj-1

Rank r_0 is associated to dominace ranking.
Thus solutions belonging to a particular front can be found by
looking at the r_0 value, r_0 >=0 (first front r_0 = 0).
Rank r_0 is located at position nvar+nobj
"""

import pandas as pd

import numpy as np
import matplotlib.pyplot as plt
from hv import HyperVolume
from moea_metrics import cmetric


def scatter_fronts_gen(run, gen, nvar, nobj, outfname):
    """Scatter plot of all fronts at the initial generation in a given run

    The plot is saved in file pscatter_fronts_genX.png
    where X is the generation number gen

    Parameters
    ----------
    run: integer
        Run number to plot results from
    gen: integer
        Generation number to plot results from
```

```
    nvar: integer
        Number of variables
    nobj: integer
        Number of objectives
    outfname: string
        Folder name where results are collected

    Returns
    -------
    figure
        The created figure

    """

    print("- scatter fronts gen ")
    # to change default colormap
    plt.rcParams["image.cmap"] = "Set1"
    # to change default color cycle
    plt.rcParams['axes.prop_cycle'] = plt.cycler(color=plt.cm.Set1.colors)
    xcol=nvar    # x values column
    ycol=nvar+1  # y  values column
    catcol=nvar+nobj # categories column, to color points

    fname= outfname + "/run" + str(run) + "/pop_g" + str(gen) +".txt"
    #print(fname)

    df = pd.read_csv(fname, sep=" ", header=None)
    categories = np.unique(df[catcol])
    fig = plt.figure()
    for i in categories:
        # Select points in front i
        x = df[df[catcol]==i][[xcol]]
        y = df[df[catcol]==i][[ycol]]
        plt.scatter(x, y, label=str(i), s=100, edgecolor='black',
                    linewidth='0.5')

    ncol = int(len(categories)/10)
    if len(categories)%10 > 0:
        ncol += 1
    plt.legend(loc='upper left',title= 'Fronts', ncol=ncol,
               bbox_to_anchor=(1, 1))
    plt.title("Run " + str(run) + " - Generation " + str(gen))
    fname = outfname+"/pscatter_fronts_gen"+str(gen)+".png"
    fig.savefig(fname, format="png", bbox_inches='tight')
    return fig

def scatter_first_front_genlist(run, gen_list, nvar, nobj, outfname):
    """Scatter plot of first front at various generation in a given run

    The plot is saved in file pscatter_firs_front_gen_list.png

    Parameters
    ----------
    run: integer
        The run number to plot results from
```

```
    gen_list: list
        List of generation to plot results from
    nvar: integer
        Number of variables
    nobj: integer
        Number of objectives
    outfname: string
        Folder name where results are collected

    Returns
    -------
    figure
        The created figure
    """

    print("- scatter first front genlist ")
    # to change default colormap
    plt.rcParams["image.cmap"] = "Set1"
    # to change default color cycle
    plt.rcParams['axes.prop_cycle'] = plt.cycler(color=plt.cm.Set1.colors)
    xcol=nvar    # x values column, first objective
    ycol=nvar+1  # y  values column, second objective
    catcol=nvar+nobj # categories column, to color points
    fig = plt.figure()
    for gen in gen_list:
        fname= outfname + "/run" + str(run) + "/pop_g" + str(gen) +".txt"
        df = pd.read_csv(fname, sep=" ", header=None)
        # Select points in front 0
        x = df[df[catcol]==0][[xcol]]
        y = df[df[catcol]==0][[ycol]]
        plt.scatter(x, y, label=str(gen), s=100, edgecolor='black',
                    linewidth='0.5')

    ncol = int(len(gen_list)/10)
    if len(gen_list)%10 > 0:
        ncol += 1
    plt.legend(loc='upper left',title= "Generation", ncol=ncol,
               bbox_to_anchor=(1, 1))
    plt.title("Front 0 - Run " + str(run) )
    fname = outfname+"/pscatter_firs_front_gen_list.png"
    fig.savefig(fname, format="png", bbox_inches='tight')
    return fig

def scatter_ithfront_gen_runlist(ifront, run_list, gen, nvar, nobj, outfname):
    """Scatter plot of the ith-front at generation gen in all runs

    The plot is saved in file pscatter_frontF_genX.png
    where F is the front number and X is the generation number gen

    Parameters
    ----------
    ifront: integer
        Front to be plotted
    run_list: list
        The runs to plot results from
```

```
    gen: integer
        Generation number to plot results from
    nvar: integer
        Number of variables
    nobj: integer
        Number of objectives
    outfname: string
        Folder name where results are collected

    Returns
    -------
    figure
        The created figure

    """

    print("- scatter ith front gen runlist ")
    # to change default colormap
    plt.rcParams["image.cmap"] = "Set1"
    # to change default color cycle
    plt.rcParams['axes.prop_cycle'] = plt.cycler(color=plt.cm.Set1.colors)
    xcol=nvar    # x values column
    ycol=nvar+1  # y  values column
    catcol=nvar+nobj # categories column, to color points
    fig = plt.figure()
    for run in run_list:
        fname= outfname + "/run" + str(run) + "/pop_g" + str(gen) +".txt"
        #print(fname)
        df = pd.read_csv(fname, sep=" ", header=None)
        # Select points in front 0
        x = df[df[catcol]==ifront][[xcol]]
        y = df[df[catcol]==ifront][[ycol]]
        plt.scatter(x, y, label=str(run), s=100, edgecolor='black',
                    linewidth='0.5')
    ncol = int(len(run_list)/10)
    if len(run_list)%11 > 0:
        ncol += 1
    int(len(run_list)/11) + 1
    plt.legend(loc='upper left',title= "Run", ncol=ncol,
               bbox_to_anchor=(1, 1))
    plt.title("Front " + str(ifront) + " -  Generation " + str(gen))
    fname = outfname+"/pscatter_front"+str(ifront)+"_gen"+str(gen)+".png"
    fig.savefig(fname, format="png", bbox_inches='tight')
    return fig

def boxplot_first_front_size(nruns, ngen, nvar, nobj, outfname,
                             gen_list, popsize):
    """Boxplot of the size of first front

    The plot is saved in file boxplot_first_front_size.png
    Reads data for all generations
    Plost for the generations specified in gen_list

    Parameters
    ----------
```

```
    nruns: integer
        Number of runs
    ngen: integer
        Number of generations (reads data for all generations)
    nvar: integer
        Number of variables
    nobj: integer
        Number of objectives
    outfname: string
        Folder name where results are collected
    gen_list: list
        List of generations to plot results
    popsize: integer
        Population size

    Returns
    -------
    figure
        The created figure

    """
    print("- boxplot of first front size ")
    front_size = []
    catcol = nvar+nobj
    for r in range(1,nruns+1):
        g_fsize = []
        for g in range(0,ngen+1):
            fname= outfname + "/run" + str(r) + "/pop_g" + str(g) +".txt"
            df = pd.read_csv(fname, sep=" ", header=None)
            x = df[df[catcol]==0]
            g_fsize.append(x.shape[0])
        front_size.append(g_fsize)
    df = pd.DataFrame(front_size, columns=list(range(0,ngen+1)))
    """
    Box plot of front size over the generations
    """
    myFig = plt.figure()
    plt.xlabel("Generarions")
    plt.ylabel("First Front Size")
#    plt.axis([0, ngen, 0, 2*popsize])
    plt.tight_layout()
    plt.grid(False)
    df[gen_list].boxplot()
    #df.boxplot()
#    plt.xticks(genlist, genlist)
#    plt.xticks(x, labels, rotation='vertical')
    myFig.savefig(outfname+"/boxplot_first_front_size.png", format="png",
                  bbox_inches='tight')

    return df

def read_fronts_all_runs_one_gen(outfname, nruns, gen, nvar, nobj):
    """Read data from a given generation for all runs

    Parameters
```

```
    ----------
    outfname: string
        Folder name where results are collected
    nruns: integer
        Number of runs
    gen: integer
        Generation number to get data and plot results
    nvar: integer
        Number of variables
    nobj: integer
        Number of objectives

    Returns
    -------
    List
        Lits of fronts.
        Each ftont is itself a list of fitness values of individuals.
        The fitness values of an individuals is also a list

    """
    #print("read fronts, ", foutput)
    #print("nvar = ",  nvar, " nobj = ", nobj)
    fronts_all_runs = []
    for run in range(1,nruns+1):
        fname=  outfname + "/run" + str(run) + "/pop_g" + str(gen) +".txt"
        fproblem = open(fname)
        line = (fproblem.readline()).split()
        front = []
        while len(line) > 0 and int(line[nvar+nobj]) == 0:
            fx = [float(fi) for fi in line[nvar:nvar+nobj]]
            front.append(fx)
            line = (fproblem.readline()).split()
        fronts_all_runs.append(front)
        fproblem.close()
    return fronts_all_runs


def boxplot_hv(outfname, refPoint, nruns, gen_list,  nvar, nobj, maxhv=True):
    """Boxplot of Hypervolume in all runs for some given generations
    Hypervolume is maximized assuming that the problem is minimization.
    Thus, the reference point  must be set in the right upper corner of
    the solution points.
    If our problem is maximization, fitness values are multiplied by -1
    The reference point should be set approrpiately
    (right upper corner of the solution points)

    The plot is saved in file boxplot_hv.png

    Parameters
    ----------
    outfname: string
        Folder name where results are collected
    refPoint: list
        The reference point to calculate the hyervolume
    nruns: integer
```

```
        Number of runs
    gen_list: list
        List of generations to plot results
    nvar: integer
        Number of variables
    nobj: integer
        Number of objectives
    maxhv:
        Indicates if hypervolume is to bbe maximized or not.
        Default is True for maximization

    Returns
    -------
    None

    """
    hv = HyperVolume(refPoint)
    vol_list =[]
    for gen in gen_list:
        all_fronts = read_fronts_all_runs_one_gen(outfname, nruns, gen,
                                                  nvar, nobj)
        vol = []
        for i in range(0,nruns):
            if maxhv == True: # hv expects minimization
                front =[]
                for p in all_fronts[i]:
                    front.append([-fi for fi in p])
            else:
                front = all_fronts[i]
           # print("gen = ", gen, " run = ", i+1)
           # print(len(front))
           # print(front)
            hv_value = hv.compute(front)
           # print("hv = ", hv_value)
            vol.append(hv_value)
        vol_list.append(vol)
    list_of_tuples = list(zip(*vol_list))
    df = pd.DataFrame(list_of_tuples, columns=gen_list)
    myFig = plt.figure()
    plt.xlabel("Generarions")
    plt.ylabel("Hypervolume")
    df.boxplot()
    myFig.savefig(outfname+"/boxplot_hv.png", format="png",
                  bbox_inches='tight')

    return 0



def stat_moea(foutput, nruns, gen_list, nvar, nobj, popsize, ngen,
              refPoint, maxhv):
    """Call some statistics for one algorithm

    Parameters
    ----------
```

```
foutput: string
    Folder name where results are collected
nruns: integer
    Number of runs
gen_list: list
    List of generations to plot results
nvar: integer
    Number of variables
nobj: integer
    Number of objectives
popsize: integer
    Population size
ngen: integer
    Numbber of generations
refPoint: list
    The reference point to calculate the hyervolume
maxhv:
    Indicates if hypervolume is to bbe maximized or not.
    Default is True for maximization

Returns
-------
None
"""

print("Statistics moea")
""" Scatter plot of all fronts at initial generation in a given run """
scatter_fronts_gen(run=1, gen=gen_list[0],
                   nvar=nvar, nobj=nobj, outfname=foutput)
""" Scatter plot of all fronts at last generation in a given run """
scatter_fronts_gen(run=1, gen=gen_list[len(gen_list)-1],
                   nvar=nvar, nobj=nobj, outfname=foutput)
""" Scatter plot of first front at various generation in a given run """
scatter_first_front_genlist(run=1, gen_list=gen_list,
                            nvar=nvar, nobj=nobj,outfname=foutput)
""" Scatter plot of first front at generation 0 in all runs """
scatter_ithfront_gen_runlist(ifront=0, run_list=list(range(1,nruns+1)),
                             gen=gen_list[0], nvar=nvar, nobj=nobj,
                             outfname=foutput)
""" Scatter plot of first front at last generation in all runs """
scatter_ithfront_gen_runlist(ifront=0, run_list=list(range(1,nruns+1)),
                             gen=gen_list[len(gen_list)-1], nvar=nvar,
                             nobj=nobj,outfname=foutput)
""" Boxplot of the size of first front """
boxplot_first_front_size(nruns=nruns, ngen=ngen, nvar=nvar, nobj=nobj,
                         outfname=foutput, gen_list=gen_list,
                         popsize=popsize)
""" Boxplot of Hypervolume in all runs"""
boxplot_hv(foutput, refPoint, nruns, gen_list,  nvar, nobj, maxhv)

return
```

```
"""
  Two algorithm comparison statistcs
"""
def boxplot_cmetric(foutputA, foutputB, nruns, genA, genB, nvar, nobj):
    """Two algorithm comparison statistcs
    Boxplots of Coverage(A,B) and Coverage(B,A)
    A and B are two non-dominated sets.  Allows to stablish which sets
    cover the other. The plot is saved in file cAB_cBA.png

    Parameters
    ----------
    foutputA: string
        Folder name where results of algorithm A are collected
    foutputB: string
        Folder name where results of algorithm B are collected
    nruns: integer
        Number of runs
    genA: int
        Generation for comparison, algorithm A
    genB: int
        Generation for comparison, algorithm B
    nvar: integer
        Number of variables
    nobj: integer
        Number of objectives

    Returns
    -------
    None
    """
    frontsA = read_fronts_all_runs_one_gen(foutputA, nruns, genA, nvar, nobj)
    frontsB = read_fronts_all_runs_one_gen(foutputB, nruns, genB, nvar, nobj)
    cAB = []
    cBA = []
    for i in range(0,nruns):
        cAB.append(cmetric(frontsA[i], frontsB[i]))
        cBA.append(cmetric(frontsB[i], frontsA[i]))
    list_of_tuples = list(zip(cAB,cBA))
    df = pd.DataFrame(list_of_tuples, columns=['cAB','cBA'])
    myFig = plt.figure()
    plt.ylabel("Coverage")
    df.boxplot()
    myFig.savefig(foutputA+"/cAB_cBA.png", format="png", bbox_inches='tight')
    return 0
# Example
#    resultsA="./output_mkp_p2_n100_True_binary0.25_G50/"
#    resultsB="./output_mkp_p2_n100_True_binary0.25_G100/"
#    boxplot_cmetric(foutputA=resultsA, foutputB=resultsB,
#                          nruns=30, genA=98, genB=99,
#                          nvar=100, nobj=2)
```