Literate Programming for evaluating the Results of my Bachelors Thesis

Jakob Hauser

June 22, 2021

Contents

1 Environment setup

importing the needed libraries

```
import pygmo
import optuna
import pickle
from collections import defaultdict
import sklearn
```

```
import statistics
from sklearn.preprocessing import MinMaxScaler
from pathlib import Path
```

custom class for my studies

```
def get_objectives(trials, MOO = True ):
    '''takes a list of trials and returns their objective values note that the Corr ge
    objective_list = []
    for trial in trials:
        if MOO:
            trial_objectives = [ 1 - trial.user_attrs['test_pearson'], trial.values[1]]
        else:
            trial_objectives = [ 1 - trial.user_attrs['test_pearson'], trial.user_attrs

        objective_list.append(trial_objectives)

    return objective_list

class my_study:

    def __init__(self, seed, sampler_type, trials, objectives, study):
        self.seed = seed
        self.sampler_type = sampler_type
        self.trials = trials
        self.objective_list = objectives
        self.name = self.seed + '_' + self.sampler_type
        self.optuna_study = study
        self.ranked_fronts = []




        self.fronts = None
        self.pareto_optimal_front = None
        self.hv = None
        self.hv_history = []

    def __repr__(self):
        return self.name
```

```
def non_dominated_sorting(self):
    # print(self.objective_list)

    self.fronts = pygmo.fast_non_dominated_sorting(self.objective_list)[0]
    for index, front in enumerate(self.fronts):
        self.ranked_fronts += [ (index, self.objective_list[i]) for i in front]




    self.pareto_optimal_front = [self.objective_list[i] for i in self.fronts[0]]
    # print(self.pareto_optimal_front)

    # self.dev_fronts = pygmo.fast_non_dominated_sorting(self.dev_objective_list)[

    # self.dev_pareto_optimal_front = [self.dev_objective_list[i] for i in self.de
def calculate_hv_history(self , ref_point):

    for i in range(1, len(self.objective_list)):
        if i == 1:
            pareto_front =[self.objective_list[0]]
        else:
            trial_subset = self.objective_list[:i]
            fronts = pygmo.fast_non_dominated_sorting(trial_subset)[0]
            pareto_front = [trial_subset[i] for i in fronts[0]]
        hv_i = pygmo.hypervolume(pareto_front)
        self.hv_history.append(hv_i.compute(ref_point))



def calculate_hv(self, ref_point):
    hv = pygmo.hypervolume(self.pareto_optimal_front)
    self.hv = hv.compute(ref_point)


    # self.dev_hv = pygmo.hypervolume(self.dev_pareto_optimal_front)
    # print(f"dev:{self.dev_hv.compute(dev_ref_point)}")
```

2 Loading the Optuna studies

```
base_path = Path("/home/jakob/src/siamese_attention_thesis/studies/")
seeds = [str(i) for i in range(3)]
sampler_names = ["RandomSampler", "MOTPESampler", "NSGAIISampler", "TPESampler"]
multi = ["RandomSampler", "MOTPESampler", "NSGAIISampler"]
trial_number = "100"
```

```
all_test_objectives = []
all_random_objectives = []
studies = []
for seed in seeds:
    for sampler in sampler_names:
            study_name = seed + "_" + sampler + "_" + trial_number + ".pkl"
            study_path = base_path / study_name

            with open(study_path, 'rb') as handle:
                study = pickle.load(handle)
                trials = study.trials
                moo = True if sampler in multi else False
                objectives = get_objectives(trials, moo)
                if sampler == 'RandomSampler':
                    all_random_objectives += objectives


                all_test_objectives += objectives
                studies.append(my_study(seed, sampler, trials, objectives, study))

all_hv = pygmo.hypervolume(all_test_objectives)
reference_point = all_hv.refpoint(1)

sum_hv = defaultdict(list)
for study in studies:
    # print(study)
    study.non_dominated_sorting()
    study.calculate_hv_history(reference_point)
    study.calculate_hv(reference_point)
```

```
    # print(study.sampler_type)
    # print(study.hv)
    sum_hv[study.sampler_type].append(study.hv)
for i in sum_hv.keys():
    print(i)
    print((sum(sum_hv[i]) / len(sum_hv[i])))
```

3 Plotting the pareto fronts

```
pareto_front_df = pd.DataFrame(columns=('Sampler', 'Seed', 'PCC','Flops','Front Rank'))

# sns.color_palette("flare", as_cmap=True)

for study in studies:
    for rank, trial in study.ranked_fronts:

        row_dict = {'Sampler': study.sampler_type, 'Seed' : study.seed, 'PCC': 1- trial
        pareto_front_df = pareto_front_df.append(row_dict, ignore_index = True)

sns.relplot(data = pareto_front_df, x='PCC', y='Flops', alpha = 0.8, row = 'Seed', col
plt.show()
```

4 Plotting the mean hypervolume for each iteration step

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# def append_hv_history_to_df(study):
#     row_dict = {'sampler_type': study.sampler_type, 'sampler_name':study.name, 'iter
#     df = pd.DataFrame(row_dict)
#     for index, hv in enumerate(study.hv_history):
#         row_dict = {'sampler_type': study.sampler_type, 'sampler_name':study.name, '
#         df = df.append({'sampler_name':study.name, 'iteration' : index + 1, 'hypervo

#     return df
hv_history= None
```

```
def hv_history_to_df(study):
    df = None
    row_dict = {'sampler_type': [study.sampler_type], 'iteration' : [0], 'hypervolume'
    df = pd.DataFrame(data = row_dict)
    for index, hv in enumerate(study.hv_history):
        row_dict = {'sampler_type': study.sampler_type, 'iteration' : index + 1, 'hype
        df = df.append(row_dict, ignore_index = True)

    return df



hv_history = pd.DataFrame(columns=('sampler_type','iteration', 'hypervolume'))
for study in studies:
    hv_history = hv_history.append(hv_history_to_df(study))
hv_mean_history = hv_history.groupby(['sampler_type','iteration']).mean()
# print(hv_history.head())
sns.set_theme(style="darkgrid")
plt.figure()
lines =sns.lineplot(
    data = hv_mean_history,
    x = 'iteration',
    y = 'hypervolume',
    hue= 'sampler_type',
    palette = 'bright'

)
lines.set(xlabel="Iterations", ylabel="Mean Hypervolume", )
plt.show()
```

5 Plotting best SOO results vs nearest best MOO result with less FLOPs

```
def get_n_best_SOO(study, n):
    results = sorted(study.objective_list, key = (lambda x : x[0]))

    return results[:n]

def get_faster_trial(study, flops):
```

```
    faster_trials = [ i for i in study.objective_list if i[1] <= flops]

    if len(faster_trials) == 0:
        return sorted(study.objective_list, key = (lambda x: x[1]))[0]
    elif len(faster_trials) == 1:
        best = sorted(faster_trials, key = (lambda x : x[0]))
        return best
    else:
        best = sorted(faster_trials, key = (lambda x : x[0]))[0]
    return best



to_beat_dict = defaultdict(list)
answers = defaultdict(list)
n = 1
number_of_seeds = 3
for study in studies:
    if study.sampler_type == "TPESampler":
        to_beat_dict[study.seed] = get_n_best_SOO(study, n)

# print(to_beat_dict)

for i in multi:
    answers[i] = [[0.0, 0.0] ]*n
# print(answers)
for study in studies:
    if study.sampler_type != "TPESampler":

        to_beat = to_beat_dict[study.seed]
        for index, best in enumerate(to_beat):
            # print(best)


            faster_trial = get_faster_trial(study, best[1])
            # print(faster_trial)


            answers[study.sampler_type][index][0] +=  faster_trial[0]
```

```
            answers[study.sampler_type][index][1] +=  faster_trial[1]
```

```
for key in answers.keys():
    for index, score  in enumerate(answers[key]):
        answers[key][index][0] = score[0] / number_of_seeds
        answers[key][index][1] = score[1] / number_of_seeds

sums = [0.0, 0.0] * n
for key in to_beat_dict.keys():
    lenght = len(to_beat_dict[key])

    for i in to_beat_dict[key]:
        sums[0] += i[0]
        sums[1] += i[1]
    to_beat = [y / 3 for y in sums]

print("to beat")
print(*to_beat)
# print(to_beat_dict)
# print("answer")
print(answers)
```

```
plt.clf()
fig, ax = plt.subplots()
colors = {"RandomSampler": "green", "MOTPESampler": "blue", "NSGAIISampler": "red"}
for i in answers.keys():
    x = 1 - to_beat[0]
    y = to_beat[1]
    u = 1 - answers[i][0][0]  - x
    v =  answers[i][0][1] - y
    print(f"x {x},y {y},u {u},v {v}," )
    ax.arrow( x= x, y=y, dx= u, dy= v, label = i, color = colors[i], length_includes_he
# ax.axis([0,1,0,7000000])
```

```
plt.legend(loc='lower right')
plt.show()
# sums = [0.0, 0.0] * n
# for key in to_beat_dict.keys():
#     lenght = len(to_beat_dict[key])

#     for i in to_beat_dict[key]:
#         sums[0] += i[0]
#         sums[1] += i[1]
#     to_beat = [y / 3 for y in sums]
```

6 violin plot of the hypervolumes

```
hypervol_df = pd.DataFrame(columns=('sampler_type', 'hypervolume'))

for study in studies:

    row_dict = {'sampler_type': study.sampler_type, 'hypervolume' : study.hv}
    hypervol_df = hypervol_df.append(row_dict, ignore_index = True)

plt.figure()


fig = plt.figure()
plt.subplot(121, title='Violin plots of the different Samplers')
sns.violinplot(data=hypervol_df,x ='sampler_type', y = 'hypervolume', cut = 0 )
plt.ylabel('Hypervolume'); plt.xlabel('Algorithm')
plt.subplot(122, title='Box plots of of the different Samplers')
sns.boxplot(data=hypervol_df, x='sampler_type',y='hypervolume')
plt.ylabel('Hypervolume'); plt.xlabel('Algorithm');
plt.tight_layout()
plt.show()
```

7 Kruskal-Wallis H-test for independent samples

```
import scipy.stats
hypervolume_dict = defaultdict(list)
for study in studies:
    if study.sampler_type != 'TPESampler':
```

```
        hypervolume_dict[study.sampler_type].append(study.hv)
kruskal = scipy.stats.kruskal(*[ hypervolume_dict[i] for i in hypervolume_dict.keys()])
print(kruskal)
```

We cannot reject the null Hypothesis that the population median of all groups are equal. While the hypervolume of the final pareto front might be equal it is still interesting to see

8 Spearmans Rank to see whether Flops and PCC are correlated in my data

```
pearson = [i[0] for i in all_random_objectives]
flops = [i[1] for i in all_random_objectives]

spearman = scipy.stats.spearmanr(a=pearson, b=flops, alternative = 'greater')

print(spearman)

print(reference_point)
```