The instruction of PLDI artifact Adaptfun

The artifact consists of the adaptivity estimator Adaptfun for selected adaptive data analysis programs AND the evaluation part comparing the generalization errors of these programs using mechanisms to show Adaptfun can help to choose the “good” mechanism to lower the generalization error.

The adaptivity estimator Adaptfun

The structure of Adaptfun consists of a dependency graph generator in Ocaml, a weight estimator in Python, an adaptivity estimator in Python.

—------------------------------------------------------------------------------------

The dependency graph generator will generate graphs and generated graphs will be stored. The code of the graph generator is located in /AdaptivityAnalysis/src/adaptDune/bin.

The generated graphs are stored in AdaptivityAnalysis/src/adaptDune/dfg, AdaptivityAnalysis/src/adaptDune/dcfg, AdaptivityAnalysis/src/adaptDune/abscfg

To run the graph generator, go to AdaptivityAnalysis/src/adaptDune : dune exec adaptDune -- -i filename

Filename is the path of the example that implements the data analysis algorithms, stored in AdaptivityAnalysis/src/adaptDune/examples.

For instance, to run the example ifCD.br

dune exec adaptDune -- -i "examples/ifCD.br"

—------------------------------------------------------------------------------------

The weight estimator and adaptivity estimator are located in /AdaptivityAnalysis/src/adaptDune/python

The starting point is the adaptfun.py file, which reads the dependency graphs (in ./dfg, ./abscfg, ./dcfg) generated by the graph generator in last step, and uses a weight infer(/AdaptivityAnalysis/src/adaptDune/python/weight\_estimate.py) to infer the weight of the node for the graphs, and using our algorithm 1 and 2 to estimate the adaptivity from the weighted dependency graph.

To run the python code, simply go to

python/adaptfun.py -e filename -v version

Filename is the path of the example fine,

and version can be 0,1,2,3 as mentioned in the paper and the appendix

* 0: ADAPTFUN
* 1: ALTERNATIVE-A : THE LIGHT AND LOOSE REACHABILITY BOUND COMPUTATION
* 2: ALTERNATIVE-B : NO REACHABILITY BOUND COMPUTATION
* 3: ALTERNATIVE-C : NO CONTROL DEPENDENCY ANALYSIS

—------------------------------------------------------------------------------------

To make it simple to evaluate, we write a shell to automatically run the ocaml graph generator, and then feed the generated graph to the python code to get the estimation of adaptivity.

To run the shell, just go to the folder AdaptivityAnalysis/src/adaptDune:

$sh adaptfun.sh filename version\_number

filename : string of the program name, path should be under the same folder

version\_number : integer range from [0, 3]

For instance, still use ifCD.br as an example.

$ sh adaptfun.sh ./examples/ifCD.br 0

The expected output will be:

---- Adaptivity Analysis on Program : ifCD.br -----./dfg/ifCD.br./dfg/ifCD.br

Total Command Numbers: 5

computation of the DFG total time:0.000297s

Loose Weight Inference:

(3,1)(4,1)(1,1)(0,1)(2,1)

computation of the DCDG total time:0.000591s

Loose Weight Inference:

(3,1)(4,1)(1,1)(0,1)(2,1)

computation of the abscfg total time:0.000194s

computation of the total parsing and graph generation time:0.000824s

--- ADAPTFUN ---

THE LOCAL BOUNDS ARE: [('1', 1), ('1', 1), ('1', 1), ('1', 1), ('1', 1), ('1', 1)]

defaultdict(<class 'list'>, {'z': [(0, None, 'Q')], 'x': [(1, None, 'Q')], 'y': [(4, None, 'Q')], 'w': [(5, None, 'Q')]})

--- REACHABILITY BOUND COMPUTATION TIME: 0.00011897087097167969 seconds ---

--- ADAPTIVITY COMPUTATION TIME: 3.0994415283203125e-05 seconds ---

The Adaptivity From This Graph is: 3

The Total Query Number For This Graph is: 1 + 1 + 1 + 1

The Estimated Generalization Error with an Optimial qurey computation Mechanism is O(3 \* √log(1 + 1 + 1 + 1)/√N )

--- ADAPTIVITY ANALYSIS TIME: 0.011018991470336914 seconds ---

Total command numbers shows the line of codes, the time of generating DFG, DCFG and abscfg are also recorded. The Local bounds are used by reachability bounds mentioned in Section 5 an 7 in the paper to estimate the weight. The time used by performing reachability bound computation is recorded, followed by the time of running our algorithm 1 in the paper AS ADAPTIVITY COMPUTATION TIME. The last few lines of the output give the estimated upper bound on the adaptivity of the program, the estimated upper bound on the total number of the program, the estimated generalization error in theory, and the total time of this analysis.

—------------------------------------------------------------------------------------

To reproduce the results in Table1 in the paper.

Please go to /AdaptivityAnalysis/src/adaptDune

sh adaptfun.sh ./examples/twoRounds.br 0

sh adaptfun.sh ./examples/multiRounds.br 0

sh adaptfun.sh ./examples/lRGD.br 0

sh adaptfun.sh ./examples/mROdd.br 0

sh adaptfun.sh ./examples/mRSingle.br 0

sh adaptfun.sh ./examples/ifCD.br 0

sh adaptfun.sh ./examples/ifVD.br 0

sh adaptfun.sh ./examples/while.br 0 (loop)

sh adaptfun.sh ./examples/whileRV.br 0 (loopRV)

sh adaptfun.sh ./examples/whileVCD.br 0 (loopVCD)

sh adaptfun.sh ./examples/whileMPVCD.br 0 (loopMPVCD)

sh adaptfun.sh ./examples/nestWhileVD.br 0 (loop2VD)

sh adaptfun.sh ./examples/nestWhileRV.br 0 (loop2RV)

sh adaptfun.sh ./examples/nestWhileMV.br 0 (loop2MV)

sh adaptfun.sh ./examples/nestWhileMPRV.br 0 (loop2MPRV)

sh adaptfun.sh ./examples/whileM.br 0 (loopM)

sh adaptfun.sh ./examples/whileMP2.br (loopM2)

sh adaptfun.sh ./examples/????.br 0 (loop2R)

sh adaptfun.sh ./examples/????.br 0 (mR(K,N))

—-------------------- big example, time consuming —--------

sh adaptfun.sh ./big-example/seqCompose.br 0 (seqCom)

sh adaptfun.sh ./big-example/mRCompose.br 0 (mRcom) (around 5000 seconds, feel free to stop at any time, switch to another version instead of 0)

sh adaptfun.sh ./big-example/tRCompose.br 0 (tRcom )

sh adaptfun.sh ./big-example/jumbo.br 0 (jumbo)

sh adaptfun.sh ./big-example/big.br 0 (big)

—------------------------------------------------------------------------------------

To reproduce table 2, the results of the alternative implementations of Adaptfun: AdaptFun-I, AdaptFun-II, AdaptFun-III.

sh adaptfun.sh ./big-example/tRCompose.br 1 (tRcom), Please check, the result is suspicious for trCom.

sh adaptfun.sh ./big-example/jumboS.br 1

sh adaptfun.sh ./big-example/big.br 1

—------------------------------------------------------------------------------------

To reproduce table 3.

Go to /AdaptivityAnalysis/eval/adapt-GnC/src/kevinJamieson folder.

python lil\_UCB.py 10

3.042777800440121 3.0585869955053884 3.5147542384126123 1.42566896105856

python lil\_UCB.py 1000

2.9100123917572995 3.1345906726533097 3.756881915105206 2.1521156835352278

python n-dimensional\_pairwise.py 10

0.33891810668670175 0.1 0.2915852742299023 0.07866265965439519

python n-dimensional\_pairwise.py 1000

0.07265803503427265 0.09999999999999998 0.1002970297029703 0.08552932216298553

python repeated\_query\_sub.py 10

0.8485226005408696 0.8827260208829042 4.893998873810069 3.20855860414123

python repeated\_query\_sub.py 1000

1.9861382470280515 1.0688531822774683 155.23249535035433 11.448460349470057

python best\_arm.py 10

3.6957655856879956 2.3935872469316184 2.367246577697541 1.1778711681353002

python best\_arm.py 1000

2.8287341658859764 1.3744872330013838 2.265150568158498 1.4418052098326748

Go to /AdaptivityAnalysis/eval/adapt-GnC/src/examples folder.

python two\_rounds.py 10

0.0006709400000000012 0.0009066666666666665 0.0005486341322337065 0.0006813222282863508

python two\_rounds.py 1000

0.036375611999999995 0.08315555555555557 0.026096895479734414 0.01619076164289563

python multiple\_rounds.py 10

0.9849 0.985026595026595 0.9763601885262316 0.9692605020419769

python multiple\_rounds.py 1000

0.9225 0.9157742257742258 0.7337122607346099 0.7360888387394103

python multiple\_rounds\_odd.py 10

0.5098565492059739 0.5069092620039783 0.5186530794269879 0.5072574742740423

python multiple\_rounds\_odd.py 1000 (5 mins)

0.5099485827579586 0.5171054182288778 0.4922465992110576 0.509964327220729

python multiple\_rounds\_single.py 10

0.5430569807343817 0.6478397330934909 0.6788191417663382 0.4572300356435887

python multiple\_rounds\_single.py 1000

0.6313581741422624 0.5690697555466334 0.4920381611613707 1.0893733043098313

python logistic\_regression\_gradient\_decent.py 10

0.1042 0.10581 0.10800000000000001 0.1028

python logistic\_regression\_gradient\_decent.py 1000

0.10940000000000001 0.10539740000000002 0.0986 0.1102

python logistic\_regression\_gradient\_decent3de.py 10

0.16999999999999998 0.16293400000000002 0.16499999999999998 0.155

python logistic\_regression\_gradient\_decent3de.py 1000

0.151 0.163568 0.138 0.159

python logistic\_regression\_gradient\_decent3dim.py 10

0.11639999999999999 0.105572 0.11399999999999999 0.10200000000000001

python logistic\_regression\_gradient\_decent3dim.py 1000

0.11239999999999999 0.10540799999999999 0.09880000000000001 0.10599999999999998

python logistic\_regression\_gradient\_decent4dim.py 10

0.1216 0.10566879999999998 0.11599999999999999 0.1016

python logistic\_regression\_gradient\_decent4dim.py 1000

0.1072 0.105214 0.08880000000000002 0.1048

python logistic\_regression\_gradient\_decent4dim.py 10

0.11059999999999999 0.1054746 0.10759999999999999 0.10500000000000001

python logistic\_regression\_gradient\_decent4de.py 1000

0.1066 0.1055902 0.1038 0.1072

def multiple\_rounds (strategy, mechanism, para = Para()):

pre\_ans = [{"para" : para}]

k = 0

while k < para.max\_iteration:

k = k + 1

q = strategy.next\_query(pre\_ans)

if q is None:

break

r = mechanism.get\_answer(q["query"])

if r[0]["answer"] is not None:

a, pre\_ans[0]["answer"] = r[0]["answer"], r[0]["answer"]

para.n\_score = [score if i in para.traced else score + (a - strategy.q\_mean)\* (np.random.choice([0, 1.0], p = [1 - strategy.pr\_1, strategy.pr\_1]) - strategy.q\_mean) for i,score in enumerate(para.n\_score)]

para.c\_score = [score + (a - strategy.q\_mean)\* (np.random.choice([0, 1.0], p = [1 - strategy.pr\_1, strategy.pr\_1]) - strategy.q\_mean) for score in para.c\_score]

for i,score in enumerate(para.n\_score):

if score > max(para.c\_score):

para.traced.add(i)

else:

q = None

break

return r[0]["answer"]

def eval\_multiple\_rounds(n = DATA\_SIZE, cardinality = CARDINALITY, para = Para(), mechanism = mech.Mechanism()):

para = Para(max\_iteration =10 , population = 200, control\_size = 10)

strategy = stg.Strategy(n, q\_mean = MEAN, ada\_freq = {"method": "multiple\_rounds", "method\_param": para}, q\_max = MAX\_QUERY\_NUM, cardinality = para.population)

mechanism.reset()

mechanism.max\_q = para.max\_iteration

mechanism.add\_data({'data': strategy.gen\_data\_integer()})

multiple\_rounds(strategy, mechanism, para)

print(len(para.traced))

q\_done = min(len(strategy.true\_ans\_list), len(strategy.mech\_ans\_list))

mse = np.square(np.subtract(strategy.true\_ans\_list[:q\_done], strategy.mech\_ans\_list[:q\_done]))

rmse = np.sqrt(mse)[-1]

n = 1000

cardinality = 10

runs = 10

beta, tau = 0.05, 1.0

sigma = .015

hold\_frac, threshold, check\_data\_frac = 0.7, 0.05, 0.05

Baseline = mech.Mechanism()

Baseline.add\_params(beta=beta, tau=tau, check\_for\_width=None)

Baseline\_rmse = np.mean([eval\_multiple\_rounds(n = n, cardinality = cardinality, mechanism = Baseline) for \_ in range(runs)])

# Baseline\_rmse = eval\_multiple\_rounds(n = n, cardinality = cardinality, para = para, mechanism = Baseline)

print(Baseline\_rmse)