Data- and Workload-Aware Algorithm for Range Queries: Implementation

October 18, 2014

This document introduces usage details of package "dawa", which includes implementations of algorithms in *A Data- and Workload-Aware Algorithm for Range Queries Under Differential Privacy* as well as some other related algorithms.

1 Install

Run setup.sh from the base directory

2 Requirements

Tested with: Python 2.6.6/2.7.1, Numpy 1.9.1, Scipy 0.11.0, Swig 2.0.4.

3 Quick Start

3.1 Import packages

If the source code is in the base directory of dawa:

```
import interface as dawa
import query              # query generation
```

If the source code is outside of the base directory of dawa:

```
import dawa
from dawa import query  # query generation
```

3.2 Use predefined algorithms

```
hatx = dawa.Algorithm[name].Run(Q, x, epsilon)
```

Right now the valid name strings are (case insensitive):

dawa: the data- and workload- aware algorithm in [1].

efpa: the EFPA algorithm from [2]

identity: using Laplace mechanism to ask each count of the data vector.

mwem: the MEWM algorithm from [4]

p-hp: the P-HP algorithm from [2]

privelet: the Privelet algorithm from [6]

structurefirst: the Mean Structure First algorithm from [7]

3.3 Generate queries

Identity queries: all identity queries on a domain with size n.

```
Q = query.Identity(n)
```

Random range queries: m random range queries on a domain with size n.

```
Q = query.RandomRange(n, m)
```

Random range queries with fixed size: m random range queries on a domain with size n with length l.

```
Q = query.FixSize(n, m, l)
```

Random clustered range queries: randomly select k centers on a domain with size n. For each cluster center c, m range queries are sampled as $[c - |X_l|, c + |X_r|]$ where X_l and X_r are independent random variables from a normal distribution with a standard deviation of dev.

```
Q = query.RandomCenter(n, m, k, stdev)
```

3.4 Run Tests

To run t tests (default: 1) on a list of algorithms $algs$ using query set Q, data vector x, privacy budget ϵ, and random seed $seed$ (default: no seed):

```
res = dawa.Test(Q, x, epsilon, algs, ntest, seed)
```

E.g. run 5 tests with dawa, mwem and identity using 10 random range queries on domain with size 32, data vector x privacy budget 1:

```
Q = query.RandomRange(32, 10)
res = dawa.Test(Q, x, 1, ['dawa','mwem','identity'],5)
```

res is a dict object with 3 keys: l0, l1, l2, which stores the L_0 (max of absolute value), L_1 (sum of absolute value), L_2 (root of sum of squares) distance between the vector of true answers and the vector of noisy answers for each algorithm and each test.

4 Customize algorithms

Advanced users can also customize algorithms using the following method:

```
alg = dawa.AlgorithmBuilder(p_name, p_argv, e_name, e_argv, ratio).
```

The meaning of parameters are:

p_name: name of the partition engine, can be None.

p_argv: parameters of the partition engine. Useless if p_name is None.

e_name: name of the estimate engine.

e_argv: parameters of the estimate engine.

ratio: ratio of the privacy budget to be used for the partition engine, which is 0.5 by default. Useless if p_name is None.

A partition algorithm is generated using the partition engine and given parameters. An estimate algorithm is generated using the estimate engine and given parameters. The final algorithm will first partition the data using the partition algorithm and then estimate each partition using the estimation algorithm. If the partition algorithm is None, the final algorithm is equivalent to the estimate algorithm. E.g.

```
alg = dawa.AlgorithmBuilder('l1partition', None, 'mwem', [20], 0.5)
```

creates an algorithm that first uses half of its privacy budget to partition the dataset using the L1 partition algorithm and then estimates each partition by an MWEM algorithm with 20 rounds. More detailed description on partition/estimate engines and their parameters can be found later in the same section.

4.1 Partition engines[1]

Right now, there are two partition engines implemented: *l1partition* and *l1approx*, which are the partition algorithm and the approximated partition algorithm. There is no parameter for those engines.

4.2 Estimate engines

The package implements and integrates several estimate algorithms. The corresponding estimate engines and their parameters can be found by the following command:

```
dawa.EstimateEngine()
```

The currently implemented estimate engines are as following. Numbers in brackets are default values for the parameters.

privelet [6]: estimate a dataset by asking its wavelet parameters.

EFPA [2]: estimate engine with the EFPA algorithm.

P-HP [2]: estimate engine with P-HP algorithm.

mwem_simple [4] : basic Multiplictive weight mechanism engine.
Parameters:
nrounds(10) - how many rounds are MWEM run.
ratio(0.5) - the ratio of privacy budget used for query selection.

structurefirst [7]: estimate engine with the structure first algorithm.

mwem [4] : MWEM engine with the enhanced update method developed by Frank McSherry.
Parameters:
nrounds(10) - how many rounds are MWEM run.
ratio(0.5) - the ratio of privacy budget used for query selection.
updateround(100) - the number of iterations in each update.

greedyH [1]: assign weights to hierarchical queries greedily according to the given workload. Answer weighted queries and generate an estimated dataset using least square estimator.
Parameters:
branch(2) - the branching factor of the hierarchy
granu(100) - the granularity in numerical search

uniformH [5]: the hierarchy engine with no decay

geometricH [3]: geometric hierarchy engine with the privacy budget decayed with depth.
Parameters:
decay($2^{1/3}$) - the ratio of privacy budget decayed with depth.

identity: estimate a dataset by asking each of its entry with Laplace mechanism.

References

[1] C. Li, M. Hay, G. Miklau, and Y. Wang. A Data- and Workload-Aware Query Answering Algorithm for Range Queries Under Differential Privacy. In *PVLDB*, 7(5): 341–352, 2014

[2] G. Ács, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. In *ICDM*, pages 1–10, 2012.

[3] G. Cormode, M. Procopiuc, E. Shen, D. Srivastava, and T. Yu. Differentially private spatial decompositions. In *ICDE*, pages 20–31, 2012.

[4] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *NIPS*, pages 2348–2356, 2012.

[5] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 3(1-2):1021–1032, 2010.

[6] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. In *ICDE*, pages 225–236, 2010.

[7] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett. Differentially private histogram publication. *The VLDB Journal*, pages 1–26, 2013.