README: SGF Tool

version 0.10a

1 Introduction

This document is the README file for the Synthetic Generation Framework (SGF). We refer the reader to the following publication for details on techniques used by the tool.

> *Plausible Deniability for Privacy-Preserving Data Synthesis.*
> Vincent Bindschaedler, Reza Shokri, and Carl Gunter.
> Proceedings of the VLDB Endowment, 2017.

Contact: bindsch2 (at) illinois (dot) edu

2 Overview

The SGF tool is a synthetic data generator. Starting from a sensitive input dataset, it produces a synthetic dataset in a privacy-preserving way. Two kinds of privacy guarantees are supported: (k, γ)-plausible deniability, and (ε, δ)-differential privacy. We refer the reader to the VLDB publication for definitions and discussions of those guarantees.

The tool consists of three executable programs: `sgfinit`, `sgfgen`, and `sgfextract`.

- `sgfinit` is used to convert the input data into a format suitable for data synthesis.
- `sgfgen` is used to perform the actual synthesis of the data, it will produce a set of candidate synthetic records.
- `sgfextract` takes a set of candidate synthetic records produced by `sgfgen` and extracts a subset of synthetic records.

3 Setup & Compilation

The tool itself is written in C++11. The cross-platform build system `cmake`[1] is required to compile the source code.

Dependencies. There are no external dependencies other that header-only libraries included in the archive.

Portability. The code has been written so that it is portable. Note: the tool has only thoroughly tested on Linux with x86_64 CPU.

[1] https://cmake.org/

To setup and compile the tool on a UNIX system, you may use the following commands which assume that the archive file sgf-0.10a.tgz is located in the current directory.

```
$ tar xf sgf-0.10a.tgz                    # extract files
$ cd sgf
$ mkdir bin && cd bin                     # out of source build
$ cmake ../source && make && cd ..        # compile with cmake
```

The three compiled binary files `sgfinit`, `sgfgen`, and `sgfextract` will be located inside the bin sub-directory. The build type can be specified by using the flag `-DCMAKE_BUILD_TYPE=DEBUG` for a debug build or the flag `-DCMAKE_BUILD_TYPE=RELEASE` for a release build.

4 Example

The archive includes a pre-processed set of data files derived from the American Community Survey 2013 (ACS13) data.[2]

In this section, we use this data as an example to illustrate how the tool may be used. We defer to Sections 5 and 6 for details about the available configuration options and the input data format supported.

4.1 Pre-processing

The input dataset must be pre-processed into a set of input files that can be fed into the tool. The format of the input files is specified in details in Section 6. In what follows we provide a brief overview. The tool requires the following input files.

- records: a csv file containing records to use for synthesis.
- stats: a csv file containing records to use to train the generative model.
- attrs: a file describing the attributes of the dataset and the possible values that each attribute can take.
- grps: a file describing a binning strategy for attributes (used to reduce the complexity of the generative model).
- dag: a file describing a Directed Acyclic Graph (DAG) which captures statistical dependencies between attributes; vertices represent attributes; edges represent dependencies.
- order: a file providing a valid topological order[3] for the graph described in the dag file.

The location of the input files is specified through the mandatory `dataprefix` configuration option. This input file path prefix is the same for all six input files. For example, if the prefix is set to data/acs13, the tool will expect to find the records file using file-path data/acs13_records.csv and the dag file using file-path data/acs13_dag.csv.

[2] https://www.census.gov/programs-surveys/acs/

[3] https://en.wikipedia.org/wiki/Topological_sorting

The data records from the input dataset are to be partitioned into two disjoint sets, one for records and the other stats. The format of those files is illustrated by the provided ACS13 sample example:

```
age,wc,edu,ms,occ,rel,race,sex,hpw,waob,incc
15,1,19,1,5,1,3,1,48,1,2
31,1,21,3,18,1,5,2,40,1,1
13,1,17,1,19,2,5,1,45,1,2
16,3,17,1,19,1,5,1,40,1,1
50,4,23,1,14,1,5,1,40,1,2
...
```

Specifically, records and stats include a header line and then one record per line.

The other four files, i.e., attrs, grps, dag, order are metadata files that describe various aspect of the input dataset and control the generative model. They must be consistent with records and stats. Section 6 provides details.

4.2 Setup

The tool will produce all outputs in a specified working directory called `workdir`. The working directory path must be specified in the configuration file. It is recommended to create a new working directory for every experiment (i.e., any new dataset or set of parameters).

```
$ mkdir workdir                    # create working directory
```

Initially, the working directory should contain the configuration file, a log configuration file, and the input data files (or a link to them). This can be accomplished as follows.

```
$ cd workdir
$ ln -s ../bin .                              # link to binaries
$ cp ../log.conf .                            # default log conf
$ cp ../examples/acs13/sb.cfg ./my.cfg        # config template
$ ln -s ../examples/acs13/data .              # link to dataset
$ echo "workdir="`pwd` >> ./my.cfg
$ echo "dataprefix="`pwd`/data/acs13 >> ./my.cfg
```

In this example, the configuration file is named my.cfg. The examples/ sub-directory of the archive contains sample configurations files which can used as templates.

There are four mandatory parameters to specify in the config file. The first two are `workdir` and `dataprefix`. (It is recommended to use absolute paths.) The third is the number of attributes in the input dataset. The provided acs13 samples contain 11 attributes, so the config file should contain `attrs=11`. The fourth and last parameter is the mechanism to use. This parameter admits two values: `seedbased` or `marginals`. These correspond to the two models described in the VLDB publication. Note that `marginals` is meant to be used as a baseline and will almost always produce lower quality output than `seedbased`.

Every time `sgfinit`, `sgfgen`, or `sgfextract` is run the tool will produce a log file (named based on the PID) in the logs/ sub-directory of the working directory. This log file can be

used to monitor the progress and will typically contain additional information not written out to the console/stdout.

4.3 Initialization

To initialize the synthesis process, `sgfinit` is used. The program takes only one command-line argument: the path to the configuration file. Assuming the current directory is the working directory, `sgfinit` is invoked as follows.

```
$ ./bin/sgfinit ./my.cfg
```

Upon successful termination, files records.dat and stats.dat will be created at the root of the working directory.

> *Important:* The tool assumes that both the configuration file and the input data files will not subsequently be modified.

4.4 Generation

To synthesize data, `sgfgen` should be run. This should always be done after running `sgfinit`. The config file will customize the generation process through several parameters.

In particular, we can set the number of synthetics to generate (`count`) and the maximum runtime (`runtime`) in seconds.

```
$ echo "count=1000" >> ./my.cfg
$ echo "runtime=3600" >> ./my.cfg
```

With these parameters, `sgfgen` will run until it produces 1000 candidate synthetics, or for at most one hour (whichever occurs first).

Once all configuration parameters have been set, `sgfgen` should be invoked as:

```
$ ./bin/sgfgen ./my.cfg
```

The candidate synthetics produced are written to a file in the gen/ sub-directory of the working directory. The synthesized records at this stage are only candidate synthetics because depending on the target privacy guarantee not all of them can be safely released.

The generation process is embarrassingly parallel and so multiple instance of `sgfgen` can be run in parallel. For example,

```
$ for i in `seq 1 8`; do ./bin/sgfgen ./my.cfg & done
```

will run eight parallel instances of the program. Each instance will produce its own set of synthetic candidates in a file gen/<PID>.out.

4.5 Extraction

Once a bunch of synthetic candidates have been produced, a synthetic dataset can be extracted using `sgfextract`. Only a suitable set of synthetics is extracted so that the target privacy guarantee is achieved.

The tool preserves privacy for both the records in the records and those in the stats file, but the purpose of `sgfextract` is focused on the guarantee with respect to the records in records. The privacy guarantee with respect to stats depends only on the following four configuration options: `ndist`, `ncomp`, `lambda`, `budget` (Section 5) which only impact `sgfgen`.

Note that with the `marginals` mechanism and for certain combinations of parameters with the `seedbased` mechanism, the generative model is independent of the seed, i.e., *seedless*. When this is the case, `sgfextract` will inform the user and the privacy guarantee obtained does *not* depend on records or the size of the extracted dataset.

In all other cases, `sgfextract` will inform the user of the exact privacy guarantee achieved. The basic privacy guarantee provided by `sgfextract` is (k, γ)-plausible deniability.[4] The value of γ is specified by the `gamma` configuration option because it is required by `sgfgen`. However, the value of k is specified to `sgfextract`. Specifically, to extract a synthetic data with plausible deniability for $k = 50$, `sgfextract` is invoked as followed.

```
$ cat ./gen/*.out > ./cand.out  # get all candidates
$ ./bin/sgfextract ./my.cfg ./cand.out ./ext 50
```

The first command-line argument is the path to the configuration file. The second is the file containing the synthetic candidates produced by `sgfgen`. The third is a file-path prefix for the extracted dataset. The last argument is the value of k which must be a positive integer.

The extracted synthetic dataset is a file called ext.synth which contains those synthetic candidates which pass the privacy test (and thus meet the privacy guarantee) specifically those for which at least 50 plausible seeds could be found. The format of this file is the same as that of the records and stats files.

If the user seeks to obtain a synthetic dataset with differential privacy guarantees (a stronger guarantee than plausible deniability) then `sgfextract` can be run in two additional ways.

The first way is to specify (in addition to k): (1) n the number of desired extracted synthetic records, (2) the value of the privacy parameter ε_0, and (3) an integer t_{inc}. In this case, `sgfextract` should be invoked as: `sgfextract <config> <candidates> <out_prefix> <n> <k> <eps0> <tinc>`. This extracts up to n synthetic records and provides a trade-off curve based on a parameter t, for each value of t (between 1 and $k-1$, by steps of size t_{inc}). The guarantee is of the form $(\varepsilon_t, \delta_t)$-differential privacy. For example:

```
$ ./bin/sgfextract ./my.cfg ./cand.out ./ext 100 5000 0.01 250
```

extracts up to 100 synthetics with $k = 5000$ and $\varepsilon_0 = 0.01$ and displays a trade-off curve for values of t with step size 250.

The second way is to specify (in addition to n the number of desired extracted synthetic records) the overall target privacy guarantee through (1) a security parameter λ such that $\delta \leq 2^{-\lambda}$, and (2) a privacy budget $\varepsilon_{\max}$. In this case, `sgfextract` should be invoked as: `sgfextract <config> <candidates> <out_prefix> <n> <lambda> <eps_max>`. In this case, the tool optimizes the privacy parameters k, ε_0, and t to meet the target guarantee. For example:

```
$ ./bin/sgfextract ./my.cfg ./cand.out ./ext 100 40 1.0
```

[4] See the VLDB publication for a precise definition and some discussion.

extracts up to 100 synthetics such that $\varepsilon \leq 1$ and $\delta \leq 2^{-40}$.

Depending on the parameters, it may happen that few or even no synthetic records are extracted. This is an indication that the parameters are either set inconsistently or their values are so stringent that no synthetic candidate can be extracted while meeting the desired privacy guarantee.

5 Configuration

The configuration file is a text file containing a single section `[all]` and a set of configuration options. Each configuration option is given as a key-value pair on its own line. There is no constraint on the order in which the configuration options appear in the file. Table 1 lists all the available configuration options.

The following is an example of `seedbased` configuration:

```
[all]
workdir=<workdir_path>
dataprefix=<data_path_prefix>
mechanism=seedbased
attrs=11
runtime=3600
count=10000
gamma=4
omega=10
ndist=geom
ncomp=seq
max_ps=25000
max_check_ps=100000
```

where `<workdir_path>` is to be replaced with an absolute path to the working directory and `<data_path_prefix>` is to be replaced with an absolute path prefix to the input data files.

Several parameters such as `omega`, `gamma`, `max_ps`, and `max_check_ps` have *no* effect for marginals synthesis. The following is an example of `marginals` configuration.

```
[all]
workdir=<workdir_path>
dataprefix=<data_path_prefix>
mechanism=marginals
attrs=11
runtime=3600
count=10000
```

[5] See Cynthia Dwork, and Aaron Roth."The algorithmic foundations of differential privacy." 2014.

6 File Formats

The tool requires six input files: records, stats, attrs, grps, dag, and order. This section describes their formats.

The current version only supports discrete attributes, thus continuous attributes must be discretized a priori by the user. The tool expects each attribute to take values in the set $\{1, 2, \ldots, n\}$ for some positive integer n. Remark: there can be at most $32,767$ attributes per dataset and for each attribute $n \leq 32,767$.

> *Remark:* Attributes and their values are always given with respect to the internal representation of the tool and *not* with respect to their meaning or representation in the original dataset. For example, the `sex` is represented as the set $\{1, 2\}$. The tool does not care that (with respect to the original dataset) '1' represents a female and '2' represents a male.

6.1 Records and Stats

The files records and stats provide data for synthesis and the training of the generative model, respectively. These two files should be in CSV format with a header line describing the attributes, and one record per line. For example, the first eight lines of example/acs13/data/acs13_stats.csv are the following.

```
age,wc,edu,ms,occ,rel,race,sex,hpw,waob,incc
17,8,18,1,2,2,5,2,20,1,1
29,4,19,1,7,1,5,2,30,1,1
35,1,21,5,6,1,5,2,40,1,1
38,1,16,4,2,1,4,1,40,3,1
5,1,13,5,2,14,5,1,40,1,1
3,1,19,5,13,16,5,2,35,1,1
17,1,22,1,3,1,5,1,20,4,1
```

Typically, the original dataset can simply be split into two disjoint subsets, one for records and the other for stats. The attributes and attribute values of records and stats must be consistent with the other four input files.

6.2 Attrs and Grps

The attrs file details one attribute per line (in order) with the following format:

`<attribute_name>,<first_value>,<second_value>,...<last_value>`.

In the current version, if an attribute can take n different values, then these values are: $1, 2, \ldots, n$. For example, the content of example/acs13/data/acs13_attrs.csv is the following.

```
age,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,$
wc,1,2,3,4,5,6,7,8
edu,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,$
ms,1,2,3,4,5
occ,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,$
```

```
rel,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
race,1,2,3,4,5
sex,1,2
hpw,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,$
waob,1,2,3,4,5,6,7,8
incc,1,2
```

In the aforementioned snippet, attributes values `age`, `edu`, `occ`, `hpw`, and `hpw` are truncated to fit within the width of this page. All four attributes admit values larger 21. For instance, age ranges from 1 to 79.

The grps file details a binning strategy one attribute per line (in order) with the following format: `<bin_for_value_1>,<bin_for_value_2>,...,<bin_for_value_n>`. The bins are specified using their index in the set $\{1, 2, \ldots, k\}$ where $k \geq 1$ is the total number of bins for that attribute. The i^{th} line corresponds to the i^{th} attribute in the same order as the attrs file. For example, the content of example/acs13/data/acs13_grps.csv is the following.

```
1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,$
1,2,3,4,5,6,7,8
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,3,4,5,6,7
1,2,3,4,5
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,2$
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
1,2,3,4,5
1,2
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,$
1,2,3,4,5,6,7,8
1,2
```

Lines for attributes `age` (line 1), `occ` (line 5), and `hpw` (line 9) are truncated to fit within the width of this page. This example uses different numbers of bins and sizing for each attribute. For instance, the values for attribute `ms` (marital status, line 4) are not binned, i.e., each value is in its own bin. Observe that for $j = 1, 2, \ldots, 5$ value j is in bin j. In contrast, for attribute `age`, values 1 through 10 (line 1) are all in the same bin (bin number 1) whereas values 11 to 20 are in bin 2.

We refer the reader to the VLDB publications for details on the binning strategy and how it used by the generative model. Note that the binning has no impact on the output synthetic dataset format.

6.3 DAG and Order

The generative model requires some knowledge of the statistical dependencies between the dataset attributes. This information is specified through the dag file as a description of a Directed Acyclic Graph (DAG).

The format of the file is the following. There is one attribute per line such that the i^{th} line corresponds to the i^{th} attribute. Specifically, line i contains a list of the attribute index of each *parent* of attribute i. Parent attributes are a subset of those which influence the value of

attribute i and so have an outgoing edge to attribute i in the DAG. We refer the reader to the VLDB publication for details on strategies to determine this parent set. Concretely, each line has the following format: `[<attr_idx_1>, ..., <attr_idx_n>], <merit_score>`, where the merit score is a non-negative real number that indicates the strength of the statistical dependency (larger scores indicate stronger relationship). The graph represent in dag *must* be acyclic. The content of example/acs13/data/acs13_dag.csv is the following.

```
0
5,0.08408
1,6,0.033552
1,6,0.27198
3,0.10942
1,0.1142
10,0.20778
5,0.072546
1,5,0.052399
3,5,0.036026
1,3,9,0.1212
```

For instance, attribute `age` (line 1) has no dependencies and thus the merit score is 0. In contrast, attribute `ms` (line 4) depends on attributes `age` (index/line 1) and `rel` (index/line 6) with a merit score of 0.27198. The last attribute `incc` (income class, line 11) depends on attributes `age` (index/line 1), `edu` (index/line 3), and `hpw` (index/line 9).

In addition to the description of the DAG itself, the tool requires a valid graph traversal order. This *must* be a topological order.[6] (Otherwise, `sgfgen` will terminate with an error without producing any synthetics.) The order used is specified through the order file. This file simply contains a ordered list of attributes, one per line. The i^{th} line of the file denotes (by index) the i^{th} attribute visited. The content of example/acs13/data/acs13_order.csv is the following.

```
1
6
3
4
5
2
10
7
8
9
11
```

The sequence of attribute to be visited is: `age` (index 1), `rel` (index 6), `edu` (index 3), `ms` (index 4), `occ` (index 5), `wc` (index 2), `waob` (index 10), `race` (index 7), `sex` (index 8), `hpw` (index 9), and `incc` (index 11).

[6] https://en.wikipedia.org/wiki/Topological_sorting

	Key	Type/Values	Default	Description
Required	workdir	string	None	Path to working directory.
	mechanism	`seedbased`, `marginals`	None	Synthesis mechanism
	dataprefix	string	None	Path prefix to the data files
	attrs	positive integer	None	Number of attributes in input dataset
Basic	count	positive integer	1048576	Number of synthetics to generate
	runtime	positive integer	7200 (2h)	Maximum generation time (seconds)
	verbose	positive integer	16	Verbosity level (min: 0, max: 32)
	rngseed	integer	0 (random seed)	Deterministic seed for RNG (useful for reproducibility).
Privacy	gamma	real number > 1	4.0	Closeness parameters of plausible seeds (γ). Record d is a plausible seed with respect to seed s, candidate synthetic y, and generative model $\mathcal{M}$ if it holds that: $\gamma^{-1} \leq \frac{\Pr\{y = \mathcal{M}(s)\}}{\Pr\{y = \mathcal{M}(d)\}} \leq \gamma$.
Generative Model	omega	positive integer	attrs	Closeness of synthetics to their seeds (ω). Number of resampled attributes is: `attrs` - ω. When $\omega =$ `attrs` the model is seedless.
	ndist	`none`, `lap`, `geom`	`lap`	Noise distribution for the generative model. None (no privacy), Laplace, or Geometric.
	ncomp	`seq`, `adv`	`seq`	Composition strategy to make the generative model privacy preserving. Sequential composition, or advanced composition[5]
	budget	real number > 0	1.0	Value of ε for (ε, δ)-DP for the generative model.
	lambda	real number > 0	60	Value of λ such that $\delta = 2^{-\lambda}$ for (ε, δ)-DP for the generative model. Only meaningful when (`ncomp=adv`).
	dir_hyperp	real number > 0	1.0	Dirichlet hyper-parameter for learning of the generative model (α)
Optimize	max_ps	integer $\geq k$	1000	Maximum number of plausible seeds to look for (0 for no max)
	max_check_ps	integer $\geq k$	100000	Maximum number of dataset records to check for plausible seeds (0 for no max)

Table 1: List of configuration parameters. All options have a default value, except for the four required parameters.