

Combining Differential Privacy with Secure Multiparty Computations

This is an implementation of the algorithms described in the paper

<https://dl.acm.org/doi/10.1145/2818000.2818027>. There are three code files each for Average calculation and Correlation calculation written in sharemind's SecreC language. The code files implement 3 different levels of differential private query's on any dataset.

1. Global Budget Differentially Privacy
2. Personalised Differential Privacy
3. Personalised Differential Privacy

The paper is focussed on the algorithms that can be used for introducing differential privacy to existing SMC protocols as well as analysing their computational overhead. Sharemind's SDK is used as a base for SMC protocols.

Requirements to run:

I have used a virtual environment (VirtualBox) to run the Sharemind SDK, which supports the SecreC language.

The VM image with inbuilt SecreC 2 compiler can be downloaded from <https://sharemind.cyber.ee/sharemind-mpc/>.

Once downloaded in the system with SecreC 2 compiler, the code files can be opened using the pre-installed Qt Creator.

The parameters are hardcoded at the top like 'n' and 'L'; they can be changed accordingly, and the code can be run using the keyboard shortcut Ctrl+Fn+F2.

Code files:

There are 3 Code files each for Average and Correlation calculation. The following is a brief explanation of the files.

1. SA_avg.sc/SA_co.sc : It implements the sample and aggregate mechanism. It includes a subroutine which computes f without privacy considerations. This function is called as a black box L times for L disjoint subsets of original dataset T of approximately equal sizes. To ensure that the added noise aligns with the differential privacy definition, it's essential to clip the output values accordingly before introducing the noise. This is an implementation of the global budget version of differential privacy. The values of 'E', 'L', 'n' can be changed as per requirement.

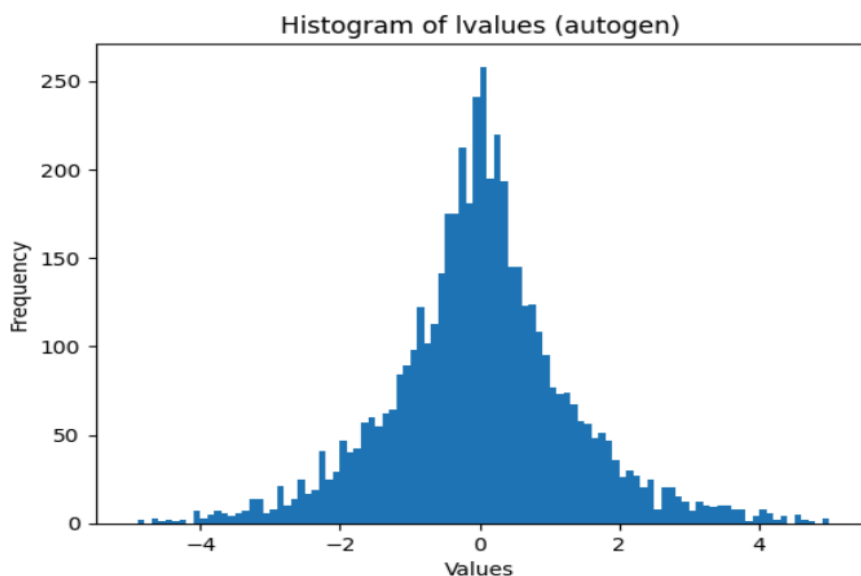
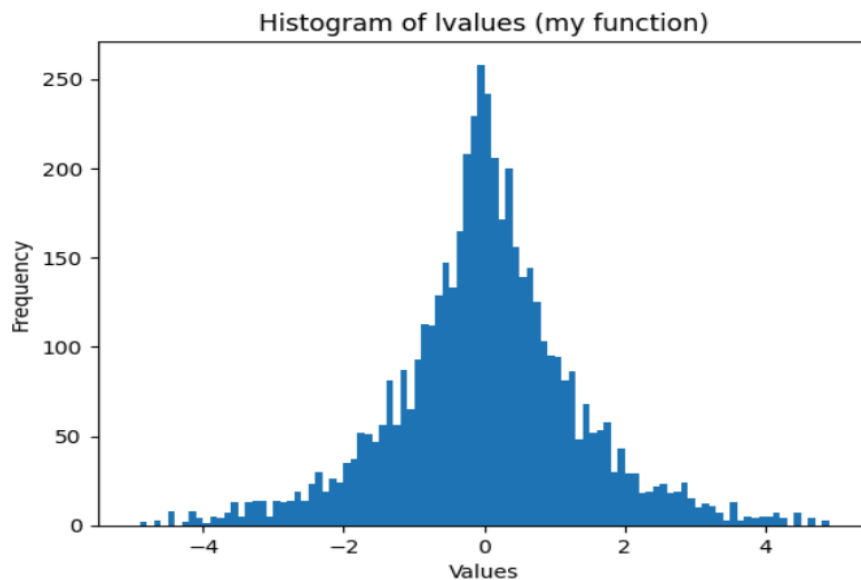
2. PDP_avg.sc/PDP_co.sc : It implements the Personalised Differential Privacy without provenances. This checks whether the current privacy budget allows for the query to be executed and accordingly

calculated the new individual budgets and updated mask vector before calling the sample and aggregate function. These individual privacy budget updates are independent, thus can be executed parallelly, reducing the overhead.

3. *PDP_withProvenances_avg.sc/PDP_withProvenances_co.sc* : It implements the Personalised Differential Privacy with provenances. The provenance budgets are assumed to be stored in a different table with some value for all provenances. The frequency of values from each provenances is calculated differentially privately, and accordingly the mask vectors and budgets are updated. The algorithm given runs in $O(n \log n)$ time where n is the sum of number of rows in the dataset and provenances tables.

Code Change for Evaluation :

The .ipynb notebook has the relevant graphs as asked during code evaluation. Initially, I have checked the validity of the laplace value generation function by comparing its output with the library function in python. The graphs are also shown below for reference

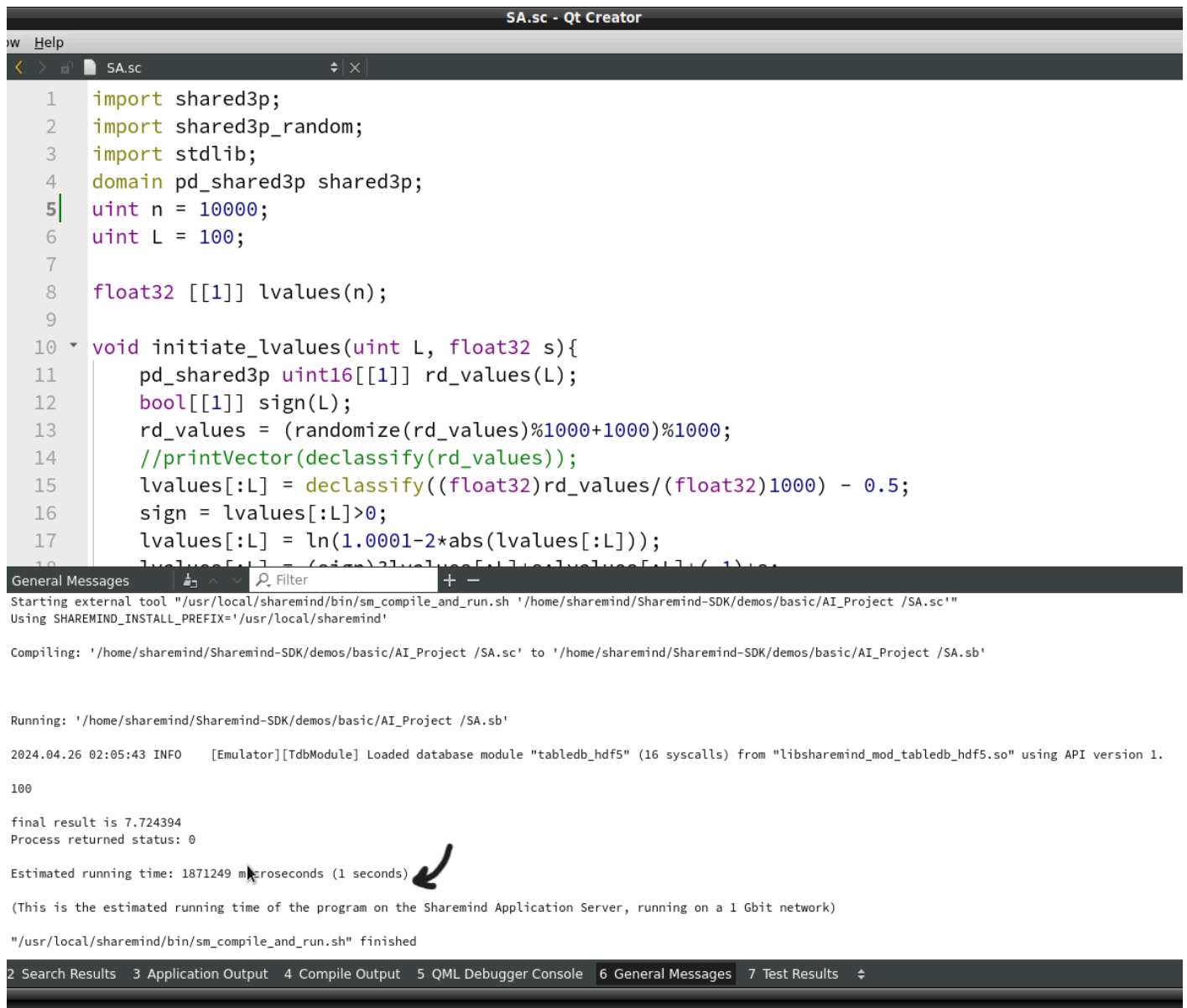


Having validated that, I have adapted the Sample and aggregate function (from SA.sc) to python.

The following code blocks in the jupyter notebook calculate the average over a dataset initialised to 6 (arbitrarily chosen). This calculation is done 1000 times and the outputs are plotted into a histogram. For the histograms, I have used a common scale for the first histogram (Blue), so that the effect of E can be observed with ease. For some of the histograms, I have added a rescaled version of the same histogram so that the distribution is apparent. As we can see, as E is reduced increasing the privacy protection, the output is more varied as compared to a larger value of E. However, in both cases the output is centered around the mean value.

Results :

The paper is aimed at determining the time complexity of adding differential privacy to secure multiparty computations. So to evaluate my implementation of the paper, I have used plotted the estimated running times of my simulations. These running times are shown in the runtime terminals as shown.



```
SA.sc - Qt Creator
SA.sc
1 import shared3p;
2 import shared3p_random;
3 import stdlib;
4 domain pd_shared3p shared3p;
5 uint n = 10000;
6 uint L = 100;
7
8 float32 [[1]] lvalues(n);
9
10 void initiate_lvalues(uint L, float32 s){
11     pd_shared3p uint16[[1]] rd_values(L);
12     bool[[1]] sign(L);
13     rd_values = (randomize(rd_values)%1000+1000)%1000;
14     //printVector(declassify(rd_values));
15     lvalues[:L] = declassify((float32)rd_values/(float32)1000) - 0.5;
16     sign = lvalues[:L]>0;
17     lvalues[:L] = ln(1.0001-2*abs(lvalues[:L]));
18     lvalues[:L] = (sign)*lvalues[:L];
19 }

General Messages
Starting external tool "/usr/local/sharemind/bin/sm_compile_and_run.sh" '/home/sharemind/Sharemind-SDK/demos/basic/AI_Project /SA.sc'
Using SHAREMIND_INSTALL_PREFIX='/usr/local/sharemind'

Compiling: '/home/sharemind/Sharemind-SDK/demos/basic/AI_Project /SA.sc' to '/home/sharemind/Sharemind-SDK/demos/basic/AI_Project /SA.sb'

Running: '/home/sharemind/Sharemind-SDK/demos/basic/AI_Project /SA.sb'

2024.04.26 02:05:43 INFO [Emulator][TdbModule] Loaded database module "tabledb_hdf5" (16 syscalls) from "libsharemind_mod_tabledb_hdf5.so" using API version 1.

100

final result is 7.724394
Process returned status: 0

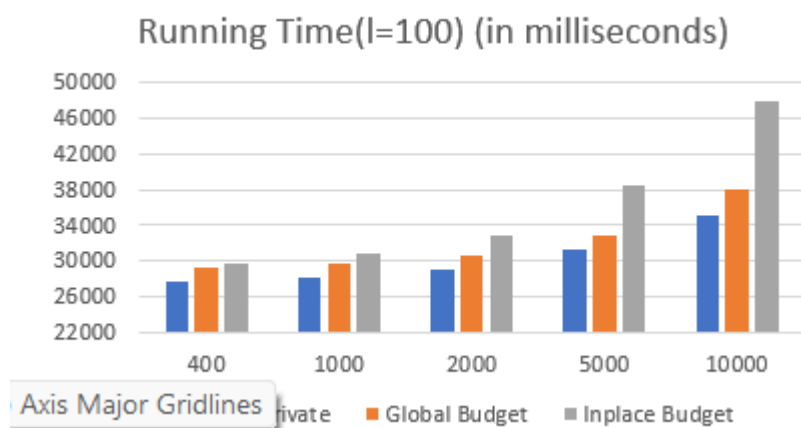
Estimated running time: 1871249 microseconds (1 seconds)
(This is the estimated running time of the program on the Sharemind Application Server, running on a 1 Gbit network)

"/usr/local/sharemind/bin/sm_compile_and_run.sh" finished

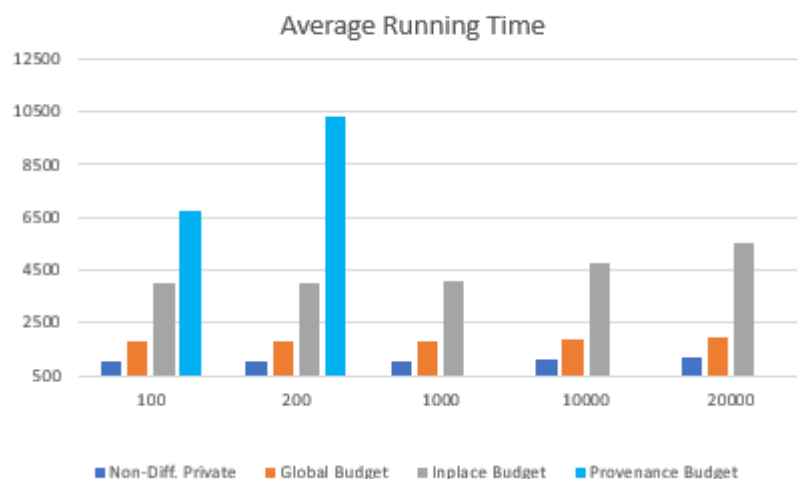
2 Search Results 3 Application Output 4 Compile Output 5 QML Debugger Console 6 General Messages 7 Test Results
```

The non differentially private time taken on my system is approximated by commenting out the lower and upper limit comparisons and laplace noise formation and addition. The global budget version corresponds to SA.sc described earlier. The following graphs represent the time complexities of the algorithms implemented.

For Correlation Calculation:



For Average Calculation:



Conclusion:

It is possible to combine differential privacy with Secure multiparty computations and create a doubly secure framework which protects the interests of the individuals as well as the mass data providers. The computational overhead of such a framework is not prohibitive.