Automatic, Fine-Grained Algorithmic Choice for Differential Privacy

Keywords: Statistical Disclosure, Differential Privacy, Private Algorithms, Dynamic Choice

Releasing some information about a private database, or *statistical disclosure*, is certainly beneficial for society — for example, a healthcare company allowing researchers to analyze their patient databases. However, such disclosures are fraught with accidental privacy leaks; the most famous example being Netflix [5], and more recent examples being AirBnB and Instagram. As we move towards an increasingly data-driven future, privacy leaks become an immediate concern. Protecting against these leaks requires defining a notion of privacy and analyzing code to ensure that it meets the desired privacy level. However, most programmers are not privacy experts, and furthermore, analyzing large codebases is a tedious and error-prone process.

Many problems that place unneeded burden on programmers are dealt with via abstraction: for instance, object-oriented programming is a data abstraction that allows for efficient reuse of code. Therefore, it makes sense to make privacy abstract for the use case of privacy-naive programmers working with large code bases. For a programming language to adequately address this use case, it would need to not only abstract privacy but help the programmer select the right algorithm and parameters for the situation. Notable previous attempts build privacy-checking into either the static type system [6] or dynamic language constructs [4], preventing privacy by not compiling or throwing runtime errors whenever they detect that too much privacy is being leaked. In either case, a privacy-naive programmer would gain no insight into whether they made a suboptimal algorithm choice which is breaking their code. This problem is noted by the authors of DPComp [2], who comment that currently, "the practitioner is lost and incapable of deploying the right algorithm".

Together with my advisor, I plan to develop a programming language, `Jostle`, that will explore the fine-grained algorithmic choices automatically. In `Jostle`, the programmer represents their code as a network of subroutines for which the privacy and accuracy are known, some of which may solve the same problem yet may outperform one another depending on the data. `Jostle` will then explore the space of privacy and accuracy of these interchangeable subroutines during runtime to try to maximize privacy subject to each subroutine having certain accuracies. Figure 1 contains an algorithm with three subroutines. Subroutine B could be accomplished in 3 different ways: B_1, B_2, and B_3. This is where `Jostle` could make an important runtime decision that would be superior to using just one of the B_i separately: pick whichever B_i has better privacy for the particular data and input parameters. The performance differences of the B_i could be massive as demonstrated by DPComp [2]. For example, suppose that B_1 computes a private Support Vector Machine Classifier while B_2 computes a private Regularized Logistic Regression Classifier. Figure 2 shows slices of a privacy-accuracy-hyperparameter search space in which `Jostle` would optimize for a particular dataset. On the dataset, Logistic Regression tends to have better accuracy for a fixed privacy, but that may not always be the case. In general, the search space would consist of the amount of privacy, the amount of accuracy, and any input parameters to a subroutine.

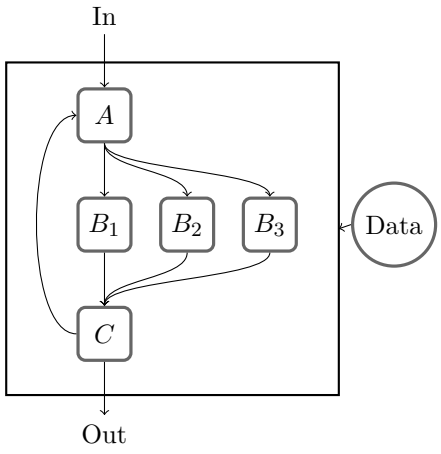

Figure 1: An example algorithm that combines 3 subroutines A, B, and C. Subroutine B can be accomplished in 3 different ways: B_1, B_2, and B_3.

For the scope of this proposal, I plan to implement a runtime solution for exploring algorithmic choice. Faceted execution [1] is a good starting place because facets represent two separate, independent executions of a program. The two classifiers example will look like:

```
label a
    restrict a: accuracy(data, C) > 0.8
    let classifier = <a? Logistic(C) : SVM(C)>
```

The `label` command creates a new label that will make the decision of which function to use based on its restrictions. The `restrict` command will add a restriction to a label, in this case forcing the classifier to have accuracy more than 0.8 subject to the data and to C. The programmer, who knows how to find the

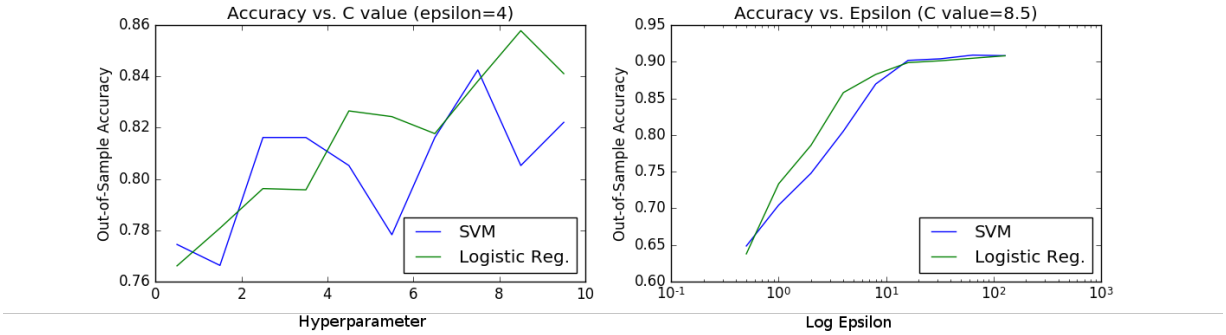

Figure 2: The performances of two models, Support Vector Machines and Logistic Regression, trained on the famous Iris dataset. Above are two cross sections of the 3D space of the hyperparameter C, the differential privacy level ϵ, and the out-of-sample accuracy in which `Jostle` would optimize.

accuracy of his models, will implement the `accuracy` function. In general, `accuracy` will be a probabilistic proposition that must be satisfied. Computing the proposition will require ideas from probabilistic programming and sampling [7] [3] because we will need to either find a closed form or sample during runtime from the distribution over all data; sampling is faster but less accurate. Doing this precisely without too much of a runtime overhead will be a challenge. Also, input parameters will need to be optimized so the subroutine both satisfies the proposition and minimizes privacy. It will be hard to choose an optimization strategy; convex optimization could be useful here, but if the algorithms are very complicated, then I will likely resort to heuristics or SMT solvers. I will enjoy working on these problems because I have experience in probability bounds as well as in optimization.

Broader Impacts `Jostle` would be the first example of dynamically exploring execution paths to enforce privacy. It is a novel application of facets and probabilistic assertions and will provide more evidence of their usefulness. It will be the first time these language constructs have been used together. In addition, `Jostle` can be applied to optimize the usage of resources other than privacy such as processor and battery time. The extent to which `Jostle` is able to select an optimal algorithm will have enormous applications to artificial intelligence and to robotics. Finally, since `Jostle` abstracts privacy and removes the difficulty of selecting the proper algorithm, it will allow more programmers to write truly private code than ever before. It is my hope that `Jostle` will improve the privacy problems that affect our lives every day.

References

[1] Thomas H. Austin, Jean Yang, Cormac Flanagan, and Armando Solar-Lezama. Faceted execution of policy-agnostic programs. In *Proceedings of the Eighth ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, PLAS '13, pages 15–26, New York, NY, USA, 2013. ACM.

[2] Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, Dan Zhang, and George Bissias. Exploring privacy-accuracy tradeoffs using dpcomp. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 2101–2104, New York, NY, USA, 2016. ACM.

[3] Martin Kucera, Petar Tsankov, Timon Gehr, Marco Guarnieri, and Martin Vechev. Spire: Synthesis of probabilistic privacy enforcement. *ACM CCS*, 2017.

[4] Frank McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. *Commun. ACM*, 53(9):89–97, September 2010.

[5] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. *CoRR*, abs/cs/0610105, 2006.

[6] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. *SIGPLAN Not.*, 45(9):157–168, September 2010.

[7] Adrian Sampson, Pavel Panchekha, Todd Mytkowicz, Kathryn S. McKinley, Dan Grossman, and Luis Ceze. Expressing and verifying probabilistic assertions. *SIGPLAN Not.*, 49(6):112–122, June 2014.