The Complexity of Computing the Optimal Composition of Differential Privacy

Jack Murtagh* Salil Vadhan†

Center for Research on Computation & Society
John A. Paulson School of Engineering & Applied Sciences
Harvard University
{jmurtagh,salil}@seas.harvard.edu

July 14, 2015

Abstract

In the study of differential privacy, composition theorems (starting with the original paper of Dwork, McSherry, Nissim, and Smith (TCC'06)) bound the degradation of privacy when composing several differentially private algorithms. Kairouz, Oh, and Viswanath (ICML'15) showed how to compute the optimal bound for composing k arbitrary (ϵ, δ)-differentially private algorithms. We characterize the optimal composition for the more general case of k arbitrary $(\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k)$-differentially private algorithms where the privacy parameters may differ for each algorithm in the composition. We show that computing the optimal composition in general is #P-complete. Since computing optimal composition exactly is infeasible (unless FP=#P), we give an approximation algorithm that computes the composition to arbitrary accuracy in polynomial time. The algorithm is a modification of Dyer's dynamic programming approach to approximately counting solutions to knapsack problems (STOC'03).

1 Introduction

Differential privacy is a framework that allows statistical analysis of private databases while minimizing the risks to individuals in the databases. The idea is that an individual should be relatively unaffected whether he or she decides to join or opt out of a research dataset. More specifically, the probability distribution of outputs of a statistical analysis of a database should be nearly identical to the distribution of outputs on the same database with a single person's data removed. Here the probability space is over the coin flips of the randomized differentially private algorithm that handles the queries. To formalize this, we call two databases D_0, D_1 with n rows each *neighboring* if they are identical on at least $n-1$ rows, and define differential privacy as follows:

*Supported by NSF grant CNS-1237235 and a grant from the Sloan Foundation.
†Supported by NSF grant CNS-1237235, a grant from the Sloan Foundation, and a Simons Investigator Award.

Definition 1.1 (Differential Privacy [DMNS06, DKMMN06])**.** A randomized algorithm M is (ϵ, δ)-*differentially private* if for all pairs of neighboring databases D_0 and D_1 and all output sets $S \subseteq \text{Range}(M)$

$$\Pr[M(D_0) \in S] \leq e^{\epsilon} \Pr[M(D_1) \in S] + \delta$$

where the probabilities are over the coin flips of the algorithm M.

In the practice of differential privacy, we generally think of ϵ as a small, non-negligible, constant (e.g. $\epsilon = .1$). We view δ as a "security parameter" that is cryptographically small (e.g. $\delta = 2^{-30}$). One of the important properties of differential privacy is that if we run multiple distinct differentially private algorithms on the same database, the resulting composed algorithm is also differentially private, albeit with some degradation in the privacy parameters (ϵ, δ). In this paper, we are interested in quantifying the degradation of privacy under composition. We will denote the composition of k differentially private algorithms $M_1, M_2, \ldots, M_k$ as $(M_1, M_2, \ldots, M_k)$ where

$$(M_1, M_2, \ldots, M_k)(x) = (M_1(x), M_2(x), \ldots, M_k(x))$$

A handful of composition theorems already exist in the literature. The first basic result says:

Theorem 1.2 (Basic Composition [DKMMN06])**.** *For every $\epsilon \geq 0$, $\delta \in [0, 1]$, and (ϵ, δ)-differentially private algorithms $M_1, M_2, \ldots, M_k$, the composition $(M_1, M_2, \ldots, M_k)$ satisfies $(k\epsilon, k\delta)$-differential privacy.*

This tells us that under composition, the privacy parameters of the individual algorithms "sum up," so to speak. We care about understanding composition because in practice we rarely want to release only a single statistic about a dataset. Releasing many statistics may require running multiple differentially private algorithms on the same database. Composition is also a very useful tool in algorithm design. Often, new differentially private algorithms are created by combining several simpler algorithms. Composition theorems help us analyze the privacy properties of algorithms designed in this way.

Theorem 1.2 shows a linear degradation in global privacy as the number of algorithms in the composition (k) grows and it is of interest to improve on this bound. If we can prove that privacy degrades more slowly under composition, we can get more utility out of our algorithms under the same global privacy guarantees. Dwork, Rothblum, and Vadhan gave the following improvement on the basic summing composition above [DRV10].

Theorem 1.3 (Advanced Composition [DRV10])**.** *For every $\epsilon > 0, \delta, \delta' > 0$, $k \in \mathbb{N}$, and (ϵ, δ)-differentially private algorithms $M_1, M_2, \ldots, M_k$, the composition $(M_1, M_2, \ldots, M_k)$ satisfies $(\epsilon_g, k\delta + \delta')$-differential privacy for*

$$\epsilon_g = \sqrt{2k \ln(1/\delta')} \cdot \epsilon + k \cdot \epsilon \cdot (e^{\epsilon} - 1)$$

Theorem 1.3 shows that privacy under composition degrades by a function of $O(\sqrt{k \ln(1/\delta')})$ which is an improvement if $\delta' = 2^{-O(k)}$. It can be shown that a degradation function of $\Omega(\sqrt{k \ln(1/\delta)})$ is necessary even for the simplest differentially private algorithms, such as randomized response [War65].

Despite giving an asymptotically correct upper bound for the global privacy parameter, ϵ_g, Theorem 1.3 is not exact. We want an exact characterization because, beyond being theoretically

interesting, constant factors in composition theorems can make a substantial difference in the practice of differential privacy. Furthermore, Theorem 1.3 only applies to "homogeneous" composition where each individual algorithm has the same pair of privacy parameters, (ϵ, δ) . In practice we often want to analyze the more general case where some individual algorithms in the composition may offer more or less privacy than others. That is, given algorithms $M_1, M_2, \ldots, M_k$, we want to compute the best achievable privacy parameters for $(M_1, M_2, \ldots, M_k)$. Formally, we want to compute the function:

$$\text{OptComp}(M_1, M_2, \ldots, M_k, \delta_g) = \inf\{\epsilon_g \colon (M_1, M_2, \ldots, M_k) \text{ is } (\epsilon_g, \delta_g)\text{-DP}\}$$

It is convenient for us to view δ_g as given and then compute the best ϵ_g, but the dual formulation, viewing ϵ_g as given, is equivalent (by binary search). Actually, we want a function that depends only on the privacy parameters of the individual algorithms:

$$\text{OptComp}((\epsilon_1, \delta_1), (\epsilon_2, \delta_2), \ldots, (\epsilon_k, \delta_k), \delta_g) = \sup\{\text{OptComp}(M_1, M_2, \ldots, M_k, \delta_g) \colon M_i \text{ is } (\epsilon_i, \delta_i)\text{-DP } \forall i \in [k]\}$$

In other words we want OptComp to give us the minimum possible ϵ_g that maintains privacy for every sequence of algorithms with the given privacy parameters (ϵ_i, δ_i). A result from Kairouz, Oh, and Viswanath [KOV15] characterizes OptComp for the homogeneous case.

Theorem 1.4 (Optimal Homogeneous Composition [KOV15])**.** *For every $\epsilon \geq 0$ and $\delta \in [0, 1)$, $\text{OptComp}((\epsilon, \delta)_1, (\epsilon, \delta)_2, \ldots, (\epsilon, \delta)_k, \delta_g) = (k-2i)\epsilon$, where i is the largest integer in $\{0, 1, \ldots, \lfloor k/2 \rfloor\}$ such that*

$$\frac{\sum_{l=0}^{i-1} \binom{k}{l} \left(e^{(k-l)\epsilon} - e^{(k-2i+l)\epsilon}\right)}{(1+e^{\epsilon})^k} \leq 1 - \frac{1-\delta_g}{(1-\delta)^k}$$

With this theorem the authors exactly characterize the composition behavior of differentially private algorithms with a polynomial-time computable solution. The problem remains to find the optimal composition behavior for the more general heterogeneous case. Kairouz, Oh, and Viswanath also provide an upper bound for heterogeneous composition that generalizes the $O(\sqrt{k \ln(1/\delta')})$ degradation found in Theorem 1.3 for homogeneous composition but do not comment on how close it is to optimal.

1.1 Our Results

We begin by extending the results of Kairouz, Oh, and Viswanath [KOV15] to the general heterogeneous case.

Theorem 1.5 (Optimal Heterogeneous Composition)**.** *For all $\epsilon_1, \ldots, \epsilon_k \geq 0$ and $\delta_1, \ldots, \delta_k, \delta_g \in [0, 1)$, $\text{OptComp}((\epsilon_1, \delta_1), (\epsilon_2, \delta_2), \ldots, (\epsilon_k, \delta_k), \delta_g)$ equals the least value of ϵ_g such that*

$$\frac{1}{\prod_{i=1}^{k}(1+e^{\epsilon_i})} \sum_{S \subseteq \{1,\ldots,k\}} \max\left\{ e^{\sum_{i \in S} \epsilon_i} - e^{\epsilon_g} \cdot e^{\sum_{i \notin S} \epsilon_i}, 0 \right\} \leq 1 - \frac{1-\delta_g}{\prod_{i=1}^{k}(1-\delta_i)} \tag{1}$$

Theorem 1.5 exactly characterizes the optimal composition behavior for any arbitrary set of differentially private algorithms. It also shows that optimal composition can be computed in time

exponential in k by computing the sum over $S \subseteq \{1, \ldots, k\}$ by brute force. Of course in practice an exponential-time algorithm is not satisfactory for large k. Our next result shows that this exponential complexity is necessary:

Theorem 1.6. *Computing* OptComp *is #P-complete, even on instances where* $\delta_1 = \delta_2 = \ldots = \delta_k = 0$ *and* $\sum_{i \in [k]} \epsilon_i \leq \epsilon$ *for any desired constant* $\epsilon > 0$.

Recall that $\#P$ is the class of counting problems associated with decision problems in NP. So being $\#P$-complete means that there is no polynomial-time algorithm for OptComp unless there is a polynomial-time algorithm for counting the number of satisfying assignments of boolean formulas (or equivalently for counting the number of solutions of all NP problems). So there is almost certainly no efficient algorithm for OptComp and therefore no analytic solution. Despite the intractability of exact computation, we show that OptComp can be approximated efficiently.

Theorem 1.7. *There is a polynomial-time algorithm that given* $\epsilon_1, \ldots, \epsilon_k \geq 0, \delta_1, \ldots \delta_k, \delta_g \in [0, 1)$, *and* $\eta > 0$, *outputs* ϵ^* *where*

$$\text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), \delta_g) \leq \epsilon^* \leq \text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), e^{-\eta/2} \cdot \delta_g) + \eta$$

The algorithm runs in $O\left(\log\left(\frac{k}{\eta}\sum_{i=1}^k \epsilon_i\right)\frac{k^2}{\eta}\sum_{i=1}^k \epsilon_i\right)$ *time assuming constant-time arithmetic operations.*

Note that we incur a relative error of η in approximating δ_g *and* an additive error of η in approximating ϵ_g. Since we always take ϵ_g to be non-negligible or even constant, we get a very good approximation when η is polynomially small or even a constant. Thus, it is acceptable that the running time is polynomial in $1/\eta$.

In addition to the results listed above, our proof of Theorem 1.5 also provides a somewhat simpler proof of the Kairouz-Oh-Viswanath homogeneous composition theorem (Theorem 1.4 [KOV15]). The proof in [KOV15] introduces a view of differential privacy through the lens of hypothesis testing and uses geometric arguments. Our proof relies only on elementary techniques commonly found in the differential privacy literature.

Practical Application. The theoretical results presented here were motivated by our work on an applied project called "Privacy Tools for Sharing Research Data"[1]. We are building a system that will allow researchers with sensitive datasets to make differentially private statistics about their data available through data repositories using the Dataverse[2] platform [Cro11, Kin07]. Part of this system is a tool that helps both data depositors and data analysts distribute a global privacy budget across many statistics. Users select which statistics they would like to compute and are given estimates of how accurately each statistic can be computed. They can also redistribute their privacy budget according to which statistics they think are most valuable in their dataset. We implemented the approximation algorithm from Theorem 1.7 and integrated it with this tool to ensure that users get the most utility out of their privacy budget.

2 Technical Preliminaries

A useful notation for thinking about differential privacy is defined below.

[1] privacytools.seas.harvard.edu
[2] dataverse.org

Definition 2.1. For two discrete random variables Y and Z taking values in the same output space S, the *δ-approximate max-divergence* of Y and Z is defined as:

$$D_\infty^\delta(Y\|Z) \equiv \max_S \left[\ln \frac{\Pr[Y \in S] - \delta}{\Pr[Z \in S]}\right]$$

Notice that an algorithm M is (ϵ, δ) differentially private if and only if for all pairs of neighboring databases, D_0, D_1, we have $D_\infty^\delta(M(D_0)\|M(D_1)) \leq \epsilon$. The standard fact that differential privacy is closed under "post processing" [DMNS06, DR13] now can be formulated as:

Fact 2.2. *If $f\colon S \to R$ is any randomized function, then*

$$D_\infty^\delta(f(Y)\|f(Z)) \leq D_\infty^\delta(Y\|Z)$$

Adaptive Composition. The composition results in our paper actually hold for a more general model of composition than the one described in the introduction. The model is called k-fold adaptive composition and was formalized in [DRV10]. We generalize their formulation to the heterogeneous setting where privacy parameters may differ across different algorithms in the composition.

The idea is that instead of running k differentially private algorithms chosen all at once on a single database, we can imagine an adversary adaptively engaging in a "composition game." The game takes as input a bit $b \in \{0,1\}$ and privacy parameters $(\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k)$. A randomized adversary A, tries to learn b through k rounds of interaction as follows: on the ith round of the game, A chooses an (ϵ_i, δ_i)-differentially private algorithm M_i and two neighboring databases $D_{(i,0)}, D_{(i,1)}$. A then receives an output $y_i = M_i(D_{(i,b)})$ where the internal randomness of M_i is independent of the internal randomness of $M_1, \ldots, M_{i-1}$. The choices of $M_i, D_{(i,0)}$, and $D_{(i,1)}$ may depend on $y_0, \ldots, y_{i-1}$ as well as the adversary's own randomness.

The outcome of this game is called the *view of the adversary*, V^b which is defined to be $(y_1, \ldots, y_k)$ along with A's coin tosses. The algorithms M_i and databases $D_{(i,0)}, D_{(i,1)}$ from each round can be reconstructed from V^b. Now we can formally define privacy guarantees under k-fold adaptive composition.

Definition 2.3. We say that the sequences of privacy parameters $\epsilon_1, \ldots, \epsilon_k \geq 0, \delta_1, \ldots, \delta_k \in [0,1)$ satisfy (ϵ_g, δ_g)-differential privacy *under adaptive composition* if for every adversary A we have $D_\infty^{\delta_g}(V^0\|V^1) \leq \epsilon_g$, where V^b represents the view of A in composition game b with privacy parameter inputs $(\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k)$.

Computing real-valued functions. Many of the computations we discuss involve irrational numbers and we need to be explicit about how we model such computations on finite, discrete machines. Namely when we talk about computing a function $f : \{0,1\}^* \to \mathbb{R}$, what we really mean is computing f to any desired number q bits of precision. More precisely, given x, q, the task is to compute a number $y \in \mathbb{Q}$ such that $|f(x) - y| \leq \frac{1}{2^q}$. We measure the complexity of algorithms for this task as a function of $|x| + q$.

3 Characterization of OptComp

Following [KOV15], we show that to analyze the composition of arbitrary (ϵ_i, δ_i)-DP algorithms, it suffices to analyze the composition of the following simple variant of randomized response [War65].

Definition 3.1 ([KOV15])**.** Define a randomized algorithm $\tilde{M}_{(\epsilon,\delta)}\colon \{0,1\} \to \{0,1,2,3\}$ as follows, setting $\alpha = 1-\delta$:

$$\begin{array}{ll} \Pr[\tilde{M}_{(\epsilon,\delta)}(0)=0] = \delta & \Pr[\tilde{M}_{(\epsilon,\delta)}(1)=0] = 0 \\ \Pr[\tilde{M}_{(\epsilon,\delta)}(0)=1] = \alpha \cdot \frac{e^\epsilon}{1+e^\epsilon} & \Pr[\tilde{M}_{(\epsilon,\delta)}(1)=1] = \alpha \cdot \frac{1}{1+e^\epsilon} \\ \Pr[\tilde{M}_{(\epsilon,\delta)}(0)=2] = \alpha \cdot \frac{1}{1+e^\epsilon} & \Pr[\tilde{M}_{(\epsilon,\delta)}(1)=2] = \alpha \cdot \frac{e^\epsilon}{1+e^\epsilon} \\ \Pr[\tilde{M}_{(\epsilon,\delta)}(0)=3] = 0 & \Pr[\tilde{M}_{(\epsilon,\delta)}(1)=3] = \delta \end{array}$$

Note that $\tilde{M}_{(\epsilon,\delta)}$ is in fact (ϵ,δ)-DP. Kairouz, Oh, and Viswanath showed that $\tilde{M}_{(\epsilon,\delta)}$ can be used to simulate the output of every (ϵ,δ)-DP algorithm on adjacent databases.

Lemma 3.2 ([KOV15])**.** *For every (ϵ,δ)-DP algorithm M and neighboring databases D_0, D_1, there exists a randomized algorithm T such that $T(\tilde{M}_{(\epsilon,\delta)}(b))$ is identically distributed to $M(D_b)$ for $b=0$ and $b=1$.*

For the sake of completeness, we provide a self-contained proof of this lemma, which does not use the hypothesis testing and geometric arguments in [KOV15]. Specifically, we give an explicit construction of the simulator, T in two steps. First we introduce a slight generalization of $\tilde{M}_{(\epsilon,\delta)}$ called $\tilde{M}_{(\epsilon,\delta_0,\delta_1)}$ and an algorithm T' that can use $\tilde{M}_{(\epsilon,\delta_0,\delta_1)}$ to simulate every differentially private algorithm on adjacent databases for some $\delta_0, \delta_1 \le \delta$. Then we show how to simulate $\tilde{M}_{(\epsilon,\delta_0,\delta_1)}$ using $\tilde{M}_{(\epsilon,\delta)}$ with an algorithm called T''. The construction will look like:

$$\tilde{M}_{(\epsilon,\delta)}(b) \xrightarrow{T''} \tilde{M}_{(\epsilon,\delta_0,\delta_1)}(b) \xrightarrow{T'} M(D_b)$$

Then the T needed for Lemma 3.2 will be $T = T' \circ T''$. Before introducing $\tilde{M}_{(\epsilon,\delta_0,\delta_1)}$ and T' we define some additional notation.

Given an (ϵ,δ)-DP algorithm M with output space R and neighboring databases D_0, D_1, let P_0, P_1 be the probability mass functions of $M(D_0)$ and $M(D_1)$, respectively. The definition of differential privacy tells us that for all sets $S \subseteq R$:

$$\begin{aligned} P_0(S) - e^\epsilon P_1(S) &\le \delta \\ P_1(S) - e^\epsilon P_0(S) &\le \delta \end{aligned}$$

The left-hand side of the first inequality is maximized by $S = S_0$ for

$$S_0 = \{r \in R\colon P_0(r) > e^\epsilon P_1(r)\} \tag{2}$$

and the left-hand side of the second inequality is maximized by

$$S_1 = \{r \in R\colon P_1(r) > e^\epsilon P_0(r)\} \tag{3}$$

Define δ_0, δ_1 as

$$\delta_0 = P_0(S_0) - e^\epsilon P_1(S_0) \le \delta \tag{4}$$

$$\delta_1 = P_1(S_1) - e^\epsilon P_0(S_1) \le \delta \tag{5}$$

We will show how to simulate M using the following algorithm.

Definition 3.3. Define $\tilde{M}_{(\epsilon,\delta_0,\delta_1)}\colon \{0,1\} \to \{0,1,2,3\}$ as follows, with δ_0, δ_1 as defined in Equations 4 and 5 for some (ϵ,δ)-DP algorithm and setting $\alpha_0 = 1-\delta_0, \alpha_1 = 1-\delta_1$:

$$\begin{array}{ll}
\Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(0)=0] = \delta_0 & \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(1)=0] = 0 \\
\Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(0)=1] = \frac{e^{2\epsilon}\alpha_0 - e^{\epsilon}\alpha_1}{e^{2\epsilon}-1} & \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(1)=1] = \frac{e^{\epsilon}\alpha_0 - \alpha_1}{e^{2\epsilon}-1} \\
\Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(0)=2] = \frac{e^{\epsilon}\alpha_1 - \alpha_0}{e^{2\epsilon}-1} & \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(1)=2] = \frac{e^{2\epsilon}\alpha_1 - e^{\epsilon}\alpha_0}{e^{2\epsilon}-1} \\
\Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(0)=3] = 0 & \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(1)=3] = \delta_1
\end{array}$$

Notice that if $\delta_0 = \delta_1 = \delta$ then $\tilde{M}_{(\epsilon,\delta_0,\delta_1)} = \tilde{M}_{(\epsilon,\delta)}$. We need to show that $\tilde{M}_{(\epsilon,\delta_0,\delta_1)}$ is composed of a valid probability distribution. Since $\alpha_b = 1-\delta_b$,

$$\sum_{x\in\{0,1,2,3\}} \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(b)=x] = 1 \text{ for } b = 0,1$$

To see that all of the terms are non-negative we need to show that the recurring terms $e^{\epsilon}\alpha_1 - \alpha_0$ and $e^{\epsilon}\alpha_0 - \alpha_1$ are non-negative and the rest follows by inspection.

Lemma 3.4. *For every (ϵ,δ)-DP algorithm, M with output space R and neighboring databases D_0 and D_1, $e^{\epsilon}\alpha_1 - \alpha_0$ and $e^{\epsilon}\alpha_0 - \alpha_1$ are non-negative where $\alpha_0 = 1-\delta_0, \alpha_1 = 1-\delta_1$ and δ_0, δ_1 are defined in Equations 4 and 5.*

Proof.

$$\begin{aligned}
\alpha_1 &= 1 - P_1(S_1) + e^{\epsilon}P_0(S_1) \\
&\le P_1(S_0) + e^{\epsilon}\cdot(1 - P_0(S_0)) \\
&\le e^{2\epsilon}P_1(S_0) + e^{\epsilon}\cdot(1 - P_0(S_0)) \\
&= e^{\epsilon}\alpha_0
\end{aligned}$$

The other inequality follows by symmetry. □

Now we show how to use $\tilde{M}_{(\epsilon,\delta_0,\delta_1)}$ to simulate any (ϵ,δ) differentially private algorithm.

Lemma 3.5. *For every (ϵ,δ)-DP algorithm M with output space R, and every pair of neighboring databases, D_0, D_1, there exists $\delta_0, \delta_1 \le \delta$ and a randomized algorithm $T'\colon \{0,1,2,3\} \to R$ such that $T'(\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(b))$ is identically distributed to $M(D_b)$ for $b=0$ and $b=1$.*

Proof. Fix neighboring databases, D_0, D_1 and let P_0, P_1 be the probability mass functions of M on D_0, D_1, respectively. We will use S_0, S_1, δ_0, and δ_1 as defined above in Equations 2, 3, 4, and 5. Fix $r \in R$. $T'\colon \{0,1,2,3\} \to R$ is defined in the table below.

x	$\Pr[T'(x)=r], r \in S_0$	$\Pr[T'(x)=r], r \in S_1$	$\Pr[T'(x)=r], r \in R \setminus S_0 \setminus S_1$
0	$\frac{1}{\delta_0}(P_0(r) - e^{\epsilon}P_1(r))$	0	0
1	$\frac{(e^{2\epsilon}-1)P_1(r)}{e^{\epsilon}\alpha_0 - \alpha_1}$	0	$\frac{e^{\epsilon}P_0(r) - P_1(r)}{e^{\epsilon}\alpha_0 - \alpha_1}$
2	0	$\frac{(e^{2\epsilon}-1)P_0(r)}{e^{\epsilon}\alpha_1 - \alpha_0}$	$\frac{e^{\epsilon}P_1(r) - P_0(r)}{e^{\epsilon}\alpha_1 - \alpha_0}$
3	0	$\frac{1}{\delta_1}(P_1(r) - e^{\epsilon}P_0(r))$	0

We need to show that $T'(x)$ is a valid probability distribution for each x. All of the terms are non-negative because $e^\epsilon\alpha_1 - \alpha_0$ and $e^\epsilon\alpha_0 - \alpha_1$ are non-negative by Lemma 3.4.

The sums of $\Pr[T'(0) = r]$ and $\Pr[T'(3) = r]$ are immediate from the definitions of δ_0 and δ_1, respectively:

$$\sum_{r\in R} \Pr[T'(0) = r] = \frac{1}{\delta_0} \sum_{r\in S_0} (P_0(r) - e^\epsilon P_1(r)) + 0 + 0 = 1$$

A symmetrical argument works for $\Pr[T'(3) = r]$. We now analyze the sum for $\Pr[T'(1) = r]$. The sum for $\Pr[T'(2) = r]$ follows by symmetry. We use the following identities:

$$\alpha_0 = 1 - \sum_{r\in S_0} (P_0(r) - e^\epsilon P_1(r)) = \sum_{r\in S_0} e^\epsilon P_1(r) + \sum_{r\in S_1} P_0(r) + \sum_{r\in R\setminus S_0\setminus S_1} P_0(r)$$
$$\alpha_1 = 1 - \sum_{r\in S_1} (P_1(r) - e^\epsilon P_0(r)) = \sum_{r\in S_0} P_1(r) + \sum_{r\in S_1} e^\epsilon P_0(r) + \sum_{r\in R\setminus S_0\setminus S_1} P_1(r)$$

Thus:

$$e^\epsilon\alpha_0 - \alpha_1 = \sum_{r\in S_0} (e^{2\epsilon} - 1)P_1(r) + \sum_{r\in R\setminus S_0\setminus S_1} (e^\epsilon P_0(r) - P_1(r))$$

This implies $\sum_{r\in R} \Pr[T'(1) = r] = 1$. Now we just need to show that $T'(\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(b))$ is identically distributed to $M(D_b)$. We will show this for $b = 0$ and the $b = 1$ case follows by symmetry. Fix $r \in R$. By the definition of $\tilde{M}_{(\epsilon,\delta_0,\delta_1)}$:

$$\Pr[T'(\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(0)) = r] = \delta_0\cdot\Pr[T'(0) = r] + \left(\frac{e^{2\epsilon}\alpha_0 - e^\epsilon\alpha_1}{e^{2\epsilon} - 1}\right)\cdot\Pr[T'(1) = r] + \left(\frac{e^\epsilon\alpha_1 - \alpha_0}{e^{2\epsilon} - 1}\right)\cdot\Pr[T'(2) = r]$$

From here we break the calculation into the three possible cases:

Case 1: $r \in S_0$

$$\begin{aligned}\Pr[T'(\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(0)) = r] &= \delta_0 \cdot \left(\frac{1}{\delta_0}(P_0(r) - e^\epsilon P_1(r))\right) + \frac{e^{2\epsilon}\alpha_0 - e^\epsilon\alpha_1}{e^{2\epsilon} - 1} \cdot \frac{(e^{2\epsilon} - 1)P_1(r)}{e^\epsilon\alpha_0 - \alpha_1} \\ &= P_0(r) - e^\epsilon P_1(r) + e^\epsilon P_1(r) = P_0(r)\end{aligned}$$

Case 2: $r \in S_1$

$$\Pr[T'(\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(0)) = r] = \frac{e^\epsilon\alpha_1 - \alpha_0}{e^{2\epsilon} - 1} \cdot \frac{(e^{2\epsilon} - 1)P_0(r)}{e^\epsilon\alpha_1 - \alpha_0} = P_0(r)$$

Case 3: $r \in R \setminus S_0 \setminus S_1$

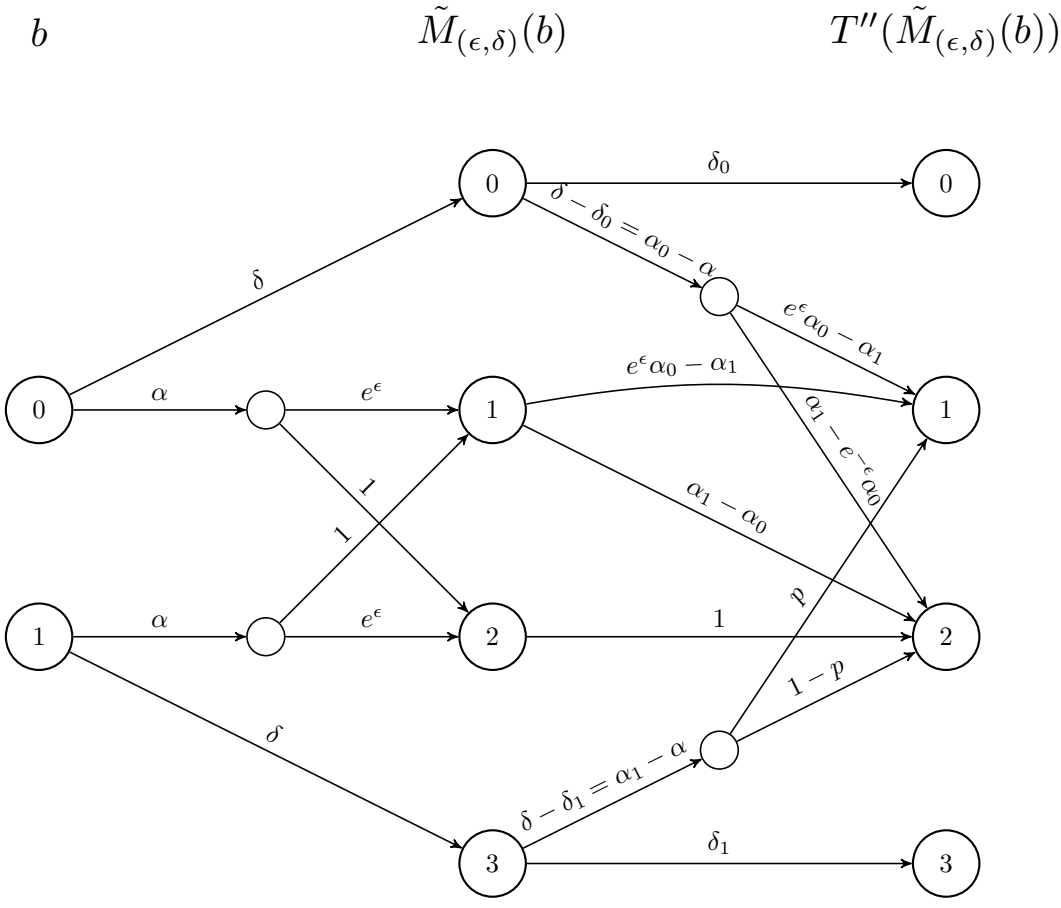
$$\begin{aligned}\Pr[T'(\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(0)) = r] &= \frac{e^{2\epsilon}\alpha_0 - e^\epsilon\alpha_1}{e^{2\epsilon} - 1} \cdot \frac{e^\epsilon P_0(r) - P_1(r)}{e^\epsilon\alpha_0 - \alpha_1} + \frac{e^\epsilon\alpha_1 - \alpha_0}{e^{2\epsilon} - 1} \cdot \frac{e^\epsilon P_1(r) - P_0(r)}{e^\epsilon\alpha_1 - \alpha_0} \\ &= \frac{e^{2\epsilon}P_0(r) - e^\epsilon P_1(r) + e^\epsilon P_1(r) - P_0(r)}{e^{2\epsilon} - 1} = P_0(r)\end{aligned}$$

□

We have shown how a generalization of $\tilde{M}_{(\epsilon,\delta)}$ called $\tilde{M}_{(\epsilon,\delta_0,\delta_1)}$ can be used to simulate the output of every differentially private algorithm. In the next lemma we show how to simulate $\tilde{M}_{(\epsilon,\delta_0,\delta_1)}$ using $\tilde{M}_{(\epsilon,\delta)}$, which implies that $\tilde{M}_{(\epsilon,\delta)}$ can be used to simulate the output of every differentially private algorithm by composing the simulator introduced in Lemma 3.5 with the one introduced below.

Lemma 3.6. *For every $\epsilon \geq 0$ and $\delta_0, \delta_1, \delta \in [0,1)$ such that $e^\epsilon \cdot (1-\delta_0) \geq 1-\delta_1$ and $e^\epsilon \cdot (1-\delta_1) \geq 1-\delta_0$ and $\delta_0, \delta_1 \leq \delta$, there exists a randomized algorithm T'' such that $T''(\tilde{M}_{(\epsilon,\delta)}(b))$ is identically distributed to $\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(b)$ for both $b = 0, 1$.*

Proof. Assume without loss of generality that $\delta_0 \geq \delta_1$ and set $\alpha = 1-\delta, \alpha_0 = 1-\delta_0$, and $\alpha_1 = 1-\delta_1$. We will represent $T''(\tilde{M}_{(\epsilon,\delta)}(b))$ as a Markov Chain below. Here, the probability of transitioning from one state to another is proportional to the weight of an edge. That is, the true probability along an edge leaving some node a is the weight divided by the sum of the weights of all of the edges leaving a (this is just to avoid cluttering the diagram with the normalizing denominators).



Where

$$p = \left(\frac{\alpha_0 - \alpha}{\alpha_1 - \alpha}\right) \cdot \left(\frac{e^\epsilon \alpha_0 - \alpha_1}{\alpha_0(e^{2\epsilon} - 1)}\right)$$

All of the weights are non-negative because $\alpha_1 \geq \alpha_0 \geq \alpha$, $e^\epsilon \alpha_1 \geq \alpha_0$, and p is also at most 1, which we verify now:

$$\begin{aligned}(\alpha_0 - \alpha) \cdot (e^\epsilon \alpha_0 - \alpha_1) &\leq (\alpha_1 - \alpha) \cdot (e^{2\epsilon} \alpha_0 - \alpha_1) \\ &\leq (\alpha_1 - \alpha) \cdot (e^{2\epsilon} \alpha_0 - \alpha_0) \\ &= (\alpha_1 - \alpha) \cdot \alpha_0 \cdot (e^{2\epsilon} - 1)\end{aligned}$$

We need to show that $T''(\tilde{M}_{(\epsilon,\delta)}(b))$ is identically distributed to $\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(b)$ for $b = 0$ and $b = 1$, which will complete the proof. Notice that $\Pr[T''(\tilde{M}_{(\epsilon,\delta)}(0)) = 3] = 0 = \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(0) = 3]$ because there is no path from the $b = 0$ node to the $T'' = 3$ node. Similarly, $\Pr[T''(\tilde{M}_{(\epsilon,\delta)}(1)) = 0] = 0 = \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(1) = 0]$ We also have:

$$\begin{aligned}\Pr[T''(\tilde{M}_{(\epsilon,\delta)}(0)) = 0] &= \left(\frac{\delta}{\delta + \alpha}\right) \cdot \left(\frac{\delta_0}{\delta_0 + (\delta - \delta_0)}\right) \\ &= \frac{\delta}{1} \cdot \frac{\delta_0}{\delta} \\ &= \delta_0 \\ &= \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(0) = 0]\end{aligned}$$

Similarly,

$$\begin{aligned}\Pr[T''(\tilde{M}_{(\epsilon,\delta)}(3)) = 3] &= \left(\frac{\delta}{\delta + \alpha}\right) \cdot \left(\frac{\delta_1}{\delta_1 + (\delta - \delta_1)}\right) \\ &= \frac{\delta}{1} \cdot \frac{\delta_1}{\delta} \\ &= \delta_1 \\ &= \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(3) = 3]\end{aligned}$$

Next we check the probabilities with which T'' outputs 1 and 2 when $b = 0$.

$$\begin{aligned}\Pr[T''(\tilde{M}_{(\epsilon,\delta)}(0)) = 1] &= \delta \cdot \left(\frac{\alpha_0 - \alpha}{\delta}\right) \cdot \left(\frac{e^\epsilon \alpha_0 - \alpha_1}{\alpha_0(e^\epsilon - e^{-\epsilon})}\right) + \alpha \cdot \left(\frac{e^\epsilon}{e^\epsilon + 1}\right) \cdot \left(\frac{e^\epsilon \alpha_0 - \alpha_1}{\alpha_0(e^\epsilon - 1)}\right) \\ &= (\alpha_0 - \alpha + \alpha) \cdot \left(\frac{e^{2\epsilon} \alpha_0 - e^\epsilon \alpha_1}{\alpha_0(e^{2\epsilon} - 1)}\right) \\ &= \frac{e^{2\epsilon} \alpha_0 - e^\epsilon \alpha_1}{e^{2\epsilon} - 1} \\ &= \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(0) = 1]\end{aligned}$$

It follows that $\Pr[T''(\tilde{M}_{(\epsilon,\delta)}(0)) = 2] = \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(0) = 2]$ because the probabilities sum to 1. Finally we show the probabilities with which T'' outputs 1 and 2 when $b = 1$.

$$\begin{aligned}\Pr[T''(\tilde{M}_{(\epsilon,\delta)}(1)) = 1] &= \delta \cdot \frac{\alpha_1 - \alpha}{\delta} \cdot \left(\frac{\alpha_0 - \alpha}{\alpha_1 - \alpha}\right) \cdot \left(\frac{e^\epsilon \alpha_0 - \alpha_1}{\alpha_0(e^{2\epsilon} - 1)}\right) + \alpha \cdot \left(\frac{1}{e^\epsilon + 1}\right) \cdot \left(\frac{e^\epsilon \alpha_0 - \alpha_1}{\alpha_0(e^\epsilon - 1)}\right) \\ &= (\alpha_0 - \alpha + \alpha) \cdot \left(\frac{e^\epsilon \alpha_0 - \alpha_1}{\alpha_0(e^{2\epsilon} - 1)}\right) \\ &= \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(1) = 1]\end{aligned}$$

Again because the probabilities sum to 1, it follows that $\Pr[T''(\tilde{M}_{(\epsilon,\delta)}(1)) = 2] = \Pr[\tilde{M}_{(\epsilon,\delta_0,\delta_1)}(1) = 2]$, which completes the proof. □

So $\tilde{M}_{(\epsilon,\delta)}$ can simulate any (ϵ, δ) differentially private algorithm. Since it is known that post-processing preserves differential privacy (Fact 2.2), it follows that to analyze the composition of arbitrary differentially private algorithms, it suffices to analyze the composition of $\tilde{M}_{(\epsilon_i,\delta_i)}$'s:

Lemma 3.7. *For all* $\epsilon_1, \ldots, \epsilon_k \geq 0, \delta_1, \ldots, \delta_k, \delta_g \in [0, 1)$,

$$\text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), \delta_g) = \text{OptComp}(\tilde{M}_{(\epsilon_1,\delta_1)}, \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}, \delta_g)$$

Proof. Since $\tilde{M}_{(\epsilon_1,\delta_1)}, \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}$ are $(\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k)$-differentially private, we have:

$$\begin{aligned}\text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), \delta_g) &= \sup\{\text{OptComp}(M_1, \ldots, M_k, \delta_g) \colon M_i \text{ is } (\epsilon_i, \delta_i)\text{-DP } \forall i \in [k]\} \\ &\geq \text{OptComp}(\tilde{M}_{(\epsilon_1,\delta_1)}, \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}, \delta_g)\end{aligned}$$

For the other direction, it suffices to show that for every $M_1, \ldots, M_k$ that are $(\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k)$-differentially private, we have

$$\text{OptComp}(M_1, \ldots, M_k, \delta_g) \leq \text{OptComp}(\tilde{M}_{(\epsilon_1,\delta_1)}, \ldots, \tilde{M}_{(\epsilon_k,\delta_k)})$$

That is,

$$\inf\{\epsilon_g \colon (M_1, \ldots, M_k) \text{ is } (\epsilon_g, \delta_g)\text{-DP}\} \leq \inf\{\epsilon_g \colon (\tilde{M}_{(\epsilon_1,\delta_1)}, \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}) \text{ is } (\epsilon_g, \delta_g)\text{-DP}\}$$

So suppose $(\tilde{M}_{(\epsilon_1,\delta_1)}, \ldots, \tilde{M}_{(\epsilon_k,\delta_k)})$ is (ϵ_g, δ_g)-DP. We will show that $(M_1, \ldots, M_k)$ is also (ϵ_g, δ_g)-DP. Taking the infimum over ϵ_g then completes the proof.

We know from Lemma 3.2 that for every pair of neighboring databases D_0, D_1, there must exist randomized algorithms $T_1, \ldots, T_k$ such that $T_i(\tilde{M}_{(\epsilon_i,\delta_i)}(b))$ is identically distributed to $M_i(D_b)$ for all $i \in \{1, \ldots, k\}$. By hypothesis we have

$$D_\infty^{\delta_g}\left((\tilde{M}_{(\epsilon_1,\delta_1)}(0), \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}(0)) \| (\tilde{M}_{(\epsilon_1,\delta_1)}(1), \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}(1))\right) \leq \epsilon_g$$

Thus by Fact 2.2 we have:

$$\begin{aligned}&D_\infty^{\delta_g}\left((M_1(D_0), \ldots, M_k(D_0)) \| (M_1(D_1), \ldots, M_k(D_1))\right) = \\ &D_\infty^{\delta_g}\left((T_1(\tilde{M}_{(\epsilon_1,\delta_1)}(0)), \ldots, T_k(\tilde{M}_{(\epsilon_k,\delta_k)}(0))) \| (T_1(\tilde{M}_{(\epsilon_1,\delta_1)}(1)), \ldots, T_k(\tilde{M}_{(\epsilon_k,\delta_k)}(1)))\right) \leq \epsilon_g\end{aligned}$$

□

Now we are ready to characterize OptComp for an arbitrary set of differentially private algorithms.

Proof of Theorem 1.5. Given $(\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k)$ and δ_g, let $\tilde{M}^k(b)$ denote the composition $(\tilde{M}_{(\epsilon_1,\delta_1)}(b), \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}(b))$ and let $\tilde{P}_b^k(x)$ be the probability mass function of $\tilde{M}^k(b)$, for $b = 0$ and $b = 1$. By Lemma 3.7, OptComp$((\epsilon_1, \delta_1), \ldots, (\epsilon_\mathrm{k}, \delta_\mathrm{k}), \delta_\mathrm{g})$ is the smallest value of ϵ_g such that:

$$\delta_g \geq \max_{Q \subseteq \{0,1,2,3\}^k} \{\tilde{P}_0^k(Q) - e^{\epsilon_g} \cdot \tilde{P}_1^k(Q)\}$$

Given ϵ_g, the set $S \subseteq \{0,1,2,3\}^k$ that maximizes the right-hand side is

$$S = S(\epsilon_g) = \left\{x \in \{0,1,2,3\}^k \mid \tilde{P}_0^k(x) \geq e^{\epsilon_g} \cdot \tilde{P}_1^k(x)\right\}$$

We can further split $S(\epsilon_g)$ into $S(\epsilon_g) = S_0(\epsilon_g) \cup S_1(\epsilon_g)$ with

$$\begin{aligned} S_0(\epsilon_g) &= \left\{x \in \{0,1,2,3\}^k \mid \tilde{P}_1^k(x) = 0\right\} \\ S_1(\epsilon_g) &= \left\{x \in \{0,1,2,3\}^k \mid \tilde{P}_0^k(x) \geq e^{\epsilon_g} \cdot \tilde{P}_1^k(x), \text{ and } \tilde{P}_1^k(x) > 0\right\} \end{aligned}$$

Note that $S_0(\epsilon_g) \cap S_1(\epsilon_g) = \emptyset$. We have $\tilde{P}_1^k(S_0(\epsilon_g)) = 0$ and $\tilde{P}_0^k(S_0(\epsilon_g)) = 1 - \Pr[\tilde{M}^k(0) \in \{1,2,3\}^k] = 1 - \prod_{i=1}^k (1-\delta_i)$. So

$$\begin{aligned} \tilde{P}_0^k(S(\epsilon_g)) - e^{\epsilon_g}\tilde{P}_1^k(S(\epsilon_g)) &= \tilde{P}_0^k(S_0(\epsilon_g)) - e^{\epsilon_g}\tilde{P}_1^k(S_0(\epsilon_g)) + \tilde{P}_0^k(S_1(\epsilon_g)) - e^{\epsilon_g}\tilde{P}_1^k(S_1(\epsilon_g)) \\ &= 1 - \prod_{i=1}^k (1-\delta_i)^k + \tilde{P}_0^k(S_1(\epsilon_g)) - e^{\epsilon_g}\tilde{P}_1^k(S_1(\epsilon_g)) \end{aligned}$$

Now we just need to analyze $\tilde{P}_0^k(S_1(\epsilon_g)) - e^{\epsilon_g}\tilde{P}_1^k(S_1(\epsilon_g))$. Notice that $S_1(\epsilon_g) \subseteq \{1,2\}^k$ because for all $x \in S_1(\epsilon_g)$, we have $\tilde{P}_0(x) > \tilde{P}_1(x) > 0$. So we can write:

$$\begin{aligned} &\tilde{P}_0^k(S_1(\epsilon_g)) - e^{\epsilon_g} \cdot \tilde{P}_1^k(S_1(\epsilon_g)) \\ &\quad = \sum_{y \in \{1,2\}^k} \max\left\{ \prod_{i:\, y_i=1} \frac{(1-\delta_i)e^{\epsilon_i}}{1+e^{\epsilon_i}} \cdot \prod_{i:\, y_i=2} \frac{(1-\delta_i)}{1+e^{\epsilon_i}} - e^{\epsilon_g} \prod_{i:\, y_i=1} \frac{(1-\delta_i)}{1+e^{\epsilon_i}} \cdot \prod_{i:\, y_i=2} \frac{(1-\delta_i)e^{\epsilon_i}}{1+e^{\epsilon_i}}, 0 \right\} \\ &\quad = \prod_{i=1}^k \frac{1-\delta_i}{1+e^{\epsilon_i}} \sum_{y \in \{0,1\}^k} \max\left\{ \frac{e^{\sum_{i=1}^k \epsilon_i}}{e^{\sum_{i=1}^k y_i\epsilon_i}} - e^{\epsilon_g} \cdot e^{\sum_{i=1}^k y_i\epsilon_i}, 0 \right\} \end{aligned}$$

Putting everything together yields:

$$\begin{aligned} \delta_g &\geq \tilde{P}_0^k(S_0(\epsilon_g)) - e^{\epsilon_g}\tilde{P}_1^k(S_0(\epsilon_g)) + \tilde{P}_0^k(S_1(\epsilon_g)) - e^{\epsilon_g}\tilde{P}_1^k(S_1(\epsilon_g)) \\ &= 1 - \prod_{i=1}^k (1-\delta_i) + \frac{\prod_{i=1}^k (1-\delta_i)}{\prod_{i=1}^k (1+e^{\epsilon_i})} \sum_{S \subseteq \{1,\ldots,k\}} \max\left\{ e^{\sum_{i \in S} \epsilon_i} - e^{\epsilon_g} \cdot e^{\sum_{i \notin S} \epsilon_i}, 0 \right\} \end{aligned}$$

□

We have characterized the optimal composition for an arbitrary set of differentially private algorithms $(M_1, \ldots, M_k)$ under the assumption that the algorithms are chosen in advance and all run on the same database. Next we show that OptComp under this restrictive model of composition is actually equivalent under the more general k-fold adaptive composition discussed in Section 2.

Theorem 3.8. *The privacy parameters $\epsilon_1, \ldots, \epsilon_k \geq 0, \delta_1, \ldots, \delta_k \in [0, 1)$, satisfy (ϵ_g, δ_g)-differential privacy under adaptive composition if and only if* $\text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), \delta_g) \leq \epsilon_g$

Proof. First suppose the privacy parameters $\epsilon_1, \ldots, \epsilon_k, \delta_1, \ldots, \delta_k$ satisfy (ϵ_g, δ_g)-differential privacy under adaptive composition. Then $\text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), \delta_g) \leq \epsilon_g$ because adaptive composition is more general than the composition defining OptComp.

Conversely, suppose $\text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), \delta_g) \leq \epsilon_g$. In particular, this means $\text{OptComp}(\tilde{M}_{(\epsilon_1,\delta_1)}, \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}, \delta_g) \leq \epsilon_g$. To complete the proof, we must show that the privacy parameters $\epsilon_1, \ldots, \epsilon_k, \delta_1, \ldots, \delta_k$ satisfy (ϵ_g, δ_g)-differential privacy under adaptive composition.

Fix an adversary A. On each round i, A uses its coin tosses r and the previous outputs $y_1, \ldots, y_{i-1}$ to select an (ϵ_i, δ_i)-differentially private algorithm $M_i = M_i^{r, y_1, \ldots, y_{i-1}}$ and neighboring databases $D_0 = D_0^{r, y_1, \ldots, y_{i-1}}, D_1 = D_1^{r, y_1, \ldots, y_{i-1}}$. Let V^b be the view of A with the given privacy parameters under composition game b for $b = 0$ and $b = 1$.

Lemma 3.2 tells us that there exists an algorithm $T_i = T_i^{r, y_1, \ldots, y_{i-1}}$ such that $T_i(\tilde{M}_{(\epsilon_i,\delta_i)}(b))$ is identically distributed to $M_i(D_b)$ for both $b = 0, 1$ for all $i \in [k]$. Define $\hat{T}(z_1, \ldots, z_k)$ for $z_1, \ldots, z_k \in \{0, 1, 2, 3\}$ as follows:

1. Randomly choose coins r for A
2. For $i = 1, \ldots, k$, let $y_i \leftarrow T_i^{r, y_1, \ldots, y_{i-1}}(z_i)$
3. Output $(r, y_1, \ldots, y_k)$

Notice that $\hat{T}(\tilde{M}_{(\epsilon_1,\delta_1)}(b), \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}(b))$ is identically distributed to V^b for both $b = 0, 1$. By hypothesis we have

$$D_\infty^{\delta_g}\left((\tilde{M}_{(\epsilon_1,\delta_1)}(0), \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}(0)) \| (\tilde{M}_{(\epsilon_1,\delta_1)}(1), \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}(1))\right) \leq \epsilon_g$$

Thus by Fact 2.2 we have:

$$D_\infty^{\delta_g}\left(V^0 \| V^1\right) = D_\infty^{\delta_g}\left(\hat{T}(\tilde{M}_{(\epsilon_1,\delta_1)}(0), \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}(0)) \| \hat{T}(\tilde{M}_{(\epsilon_1,\delta_1)}(1), \ldots, \tilde{M}_{(\epsilon_k,\delta_k)}(1))\right) \leq \epsilon_g$$

□

4 Hardness of OptComp

$\#P$ is the class of all counting problems associated with decision problems in NP. It is a set of functions that count the number of solutions to some NP problem. More formally:

Definition 4.1. A function $f\colon \{0,1\}^* \to \mathbb{N}$ is in the class $\#P$ if there exists a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ and a polynomial time algorithm M such that for every $x \in \{0,1\}^*$:

$$f(x) = \left|\left\{y \in \{0,1\}^{p(|x|)}\colon M(x, y) = 1\right\}\right|$$

Definition 4.2. A function g is called *#P-hard* if every function $f \in \#P$ can be computed in polynomial time given oracle access to g. That is, evaluations of g can be done in one time step.

If a function is #P-hard, then there is no polynomial-time algorithm for computing it unless there is a polynomial-time algorithm for counting the number of solutions of all NP problems.

Definition 4.3. A function f is called *#P-easy* if there is some function $g \in \#P$ such that f can be computed in polynomial time given oracle access to g.

If a function is both #P-hard and #P-easy, we say it is #P-complete. Proving that computing OptComp is #P-complete can be broken into two steps: showing that it is #P-easy and showing that it is #P-hard.

Lemma 4.4. *Computing OptComp is #P-easy.*

Proof. For convenience we will view $(\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k)$ and ϵ_g as given arguments to OptComp and compute δ_g. Recall that the two versions of OptComp, viewing ϵ_g as given and computing δ_g and vice versa, are equivalent up to a polynomial factor (just run binary search over values of δ_g computing polynomially many bits of precision). So the formulation we choose for the proof will not affect whether OptComp is in #P or not. Recall that in our model of computing real valued functions, we will take another input q and we will output δ_g satisfying the following, to q bits of precision:

$$\frac{1}{\prod_{i=1}^{k}(1+e^{\epsilon_i})} \sum_{S \subseteq \{1,\ldots,k\}} \max\left\{ e^{\sum_{i \in S} \epsilon_i} - e^{\epsilon_g} \cdot e^{\sum_{i \notin S} \epsilon_i}, 0 \right\} = 1 - \frac{1-\delta_g}{\prod_{i=1}^{k}(1-\delta_i)}$$

in polynomial time using a #P oracle. Notice that the only part of the expression above that cannot be computed in polynomial time is the summation over subsets of $\{1, \ldots, k\}$. If we knew the sum, computing δ_g would be easy given our inputs. We show how to compute the sum in polynomial time using a #P oracle and it follows that computing δ_g is #P-easy .

Define $f\colon 2^{[k]} \to \mathbb{R}$ as $f(S) = \max\left\{ e^{\sum_{i \in S} \epsilon_i} - e^{\epsilon_g} \cdot e^{\sum_{i \notin S} \epsilon_i}, 0 \right\}$. f is computable in polynomial time (to any desired precision). Let $\hat{f}$ be a function computable in polynomial time where $\left|\hat{f}(S) - f(S)\right| < \frac{1}{2^{q+k}}$ for all S. Set $m = 10^q$. Now define the function $g\colon 2^{[k]} \times \mathbb{N} \to \{0,1\}$ as follows:

$$g(S,n) = \begin{cases} 1 & \text{if } m \cdot \hat{f}(S) \geq n \\ 0 & \text{otherwise} \end{cases}$$

We can now phrase a decision problem in NP: Does there exist a pair (S,n) such that $g(S,n) = 1$? This is in NP because given a witness (S,n), we can compute $m \cdot \hat{f}(S)$ and compare the output to n, thereby verifying the solution, in polynomial time. Since this is an NP problem, a #P oracle can count the number of solutions to it in one time step. Notice that for every set S, the number of solutions (pairs of the form (S,n) satisfying $g(S,n) = 1$) is exactly $m \cdot \hat{f}(S)$ because g will output 1 for $g(S,1), g(S,2), \ldots, g(S, m \cdot \hat{f}(S))$. So over all possible sets S, the number of solutions as counted by the #P oracle equals $m \cdot \sum_{S \subseteq [k]} \hat{f}(S)$. Dividing this by m gives us the sum up to an additive error of $\frac{2^k}{2^{q+k}} = \frac{1}{2^q}$, which can be used to compute δ_g to q bits of precision in polynomial time. This only required one call to a #P oracle. So computing OptComp is #P-easy. ∎

Next we show that computing OptComp is also $\#P$-hard through a series of reductions. We start with a multiplicative version of the partition problem that is known to be $\#P$-complete by Ehrgott [Ehr00]. The problems in the chain of reductions are defined below.

Definition 4.5. #INT-PARTITION is the following problem: given a set $Z = \{z_1, z_2, \ldots, z_k\}$ of positive integers, count the number of partitions $P \subseteq [k]$ such that

$$\prod_{i \in P} z_i - \prod_{i \notin P} z_i = 0$$

All of the remaining problems in our chain of reductions take inputs $\{w_1, \ldots, w_k\}$ where $1 \leq w_i \leq e$ is the Dth root of a positive integer for all $i \in [k]$ and some positive integer D. All of the reductions we present hold for every positive integer D, including $D = 1$ when the inputs are integers. The reason we choose D to be large enough such that our inputs are in the range $[1, e]$ is because in the final reduction to OptComp, ϵ_i values in the proof are set to $\ln(w_i)$. We want to show that our reductions hold for reasonable values of ϵ's in a differential privacy setting so throughout the proofs we use w_i's $\in [1, e]$ to correspond to ϵ_i's $\in [0, 1]$ in the final reduction. It is important to note though that the reductions still hold for any choice of positive integer D and thus any range of ϵ's ≥ 0.

Definition 4.6. #PARTITION is the following problem: given a number $D \in \mathbb{N}$ and a set $W = \{w_1, w_2, \ldots, w_k\}$ of real numbers where for all $i \in [k]$, $1 \leq w_i \leq e$ is the Dth root of a positive integer, count the number of partitions $P \subseteq [k]$ such that

$$\prod_{i \in P} w_i - \prod_{i \notin P} w_i = 0$$

Definition 4.7. #T-PARTITION is the following problem: given a number $D \in \mathbb{N}$ and a set $W = \{w_1, w_2, \ldots, w_k\}$ of real numbers where for all $i \in [k]$, $1 \leq w_i \leq e$ is the Dth root of a positive integer and a *positive* real number T, count the number of partitions $P \subseteq [k]$ such that

$$\prod_{i \in P} w_i - \prod_{i \notin P} w_i = T$$

Definition 4.8. #SUM-PARTITION: given a number $D \in \mathbb{N}$ and a set $W = \{w_1, w_2, \ldots, w_k\}$ of real numbers where for all $i \in [k]$, $1 \leq w_i \leq e$ is the Dth root of a positive integer and a real number $r > 1$, find

$$\sum_{P \subseteq [k]} \max \left\{ \prod_{i \in P} w_i - r \cdot \prod_{i \notin P} w_i, 0 \right\}$$

We prove that computing OptComp is $\#P$-hard by the following series of reductions:

#INT-PARTITION $\leq$ #PARTITION $\leq$ #T-PARTITION $\leq$ #SUM-PARTITION $\leq$ OptComp

Since #INT-PARTITION is known to be $\#P$-complete [Ehr00], the chain of reductions will prove that OptComp is $\#P$-hard.

Lemma 4.9. *For every constant $c > 1$,* #PARTITION *is #P-hard, even on instances where* $\prod_i w_i \leq c$.

Proof. Given an instance of #INT-PARTITION, $\{z_1, \ldots, z_k\}$, we show how to find the solution in polynomial time using a #PARTITION oracle. Set $D = \lceil \log_c(\prod_i z_i) \rceil$ and $w_i = \sqrt[D]{z_i}\ \forall i \in [k]$. Note that $\prod_i w_i = \left(\prod_i z_i\right)^{1/D} \leq c$. Let $P \subseteq [k]$:

$$\prod_{i \in P} w_i = \prod_{i \notin P} w_i \iff \left(\prod_{i \in P} w_i\right)^D = \left(\prod_{i \notin P} w_i\right)^D$$
$$\iff \prod_{i \in P} z_i = \prod_{i \notin P} z_i$$

There is a one-to-one correspondence between solutions to the #PARTITION problem and solutions to the given #INT-PARTITION instance. We can solve #INT-PARTITION in polynomial time with a #PARTITION oracle. Therefore #PARTITION is #P-hard. □

Lemma 4.10. *For every constant $c > 1$,* #T-PARTITION *is #P-hard, even on instances where* $\prod_i w_i \leq c$.

Proof. Let $c > 1$ be a constant. We will reduce from #PARTITION, so consider an instance of the #PARTITION problem, $W = \{w_1, w_2, \ldots, w_k\}$. We may assume $\prod_i w_i \leq \sqrt{c}$ since $\sqrt{c}$ is also a constant greater than 1.

Set $W' = W \cup \{w_{k+1}\}$, where $w_{k+1} = \prod_{i=1}^{k} w_i$. Notice that $\prod_{i=1}^{k+1} w_i \leq (\sqrt{c})^2 = c$. Set $T = \sqrt{w_{k+1}}\,(w_{k+1} - 1)$. Now we can use a #T-PARTITION oracle to count the number of partitions $Q \subseteq \{1, \ldots, k+1\}$ such that

$$\prod_{i \in Q} w_i - \prod_{i \notin Q} w_i = T$$

Let $P = Q \cap \{1, \ldots, k\}$. We will argue that $\prod_{i \in Q} w_i - \prod_{i \notin Q} w_i = T$ if and only if $\prod_{i \in P} w_i = \prod_{i \notin P} w_i$, which completes the proof. There are two cases to consider: $w_{k+1} \in Q$ and $w_{k+1} \notin Q$.

Case 1: $w_{k+1} \in Q$. In this case, we have:

$$w_{k+1} \cdot \left(\prod_{i \in P} w_i\right) - \prod_{i \notin P} w_i = \prod_{i \in Q} w_i - \prod_{i \notin Q} w_i = T = \sqrt{w_{k+1}}\,(w_{k+1} - 1)$$

$$\iff \left(\prod_{i \in [k]} w_i\right)\left(\prod_{i \in P} w_i\right)^2 - \prod_{i \in [k]} w_i = \sqrt{\prod_{i \in [k]} w_i}\left(\prod_{i \in [k]} w_i - 1\right)\left(\prod_{i \in P} w_i\right) \quad \text{multiplied both sides by } \prod_{i \in P} w_i$$

$$\iff \left(\prod_{i \in P} w_i - \sqrt{\prod_{i \in [k]} w_i}\right)\left(\prod_{i \in [k]} w_i \prod_{i \in P} w_i + \sqrt{\prod_{i \in [k]} w_i}\right) = 0 \quad \text{factored quadratic in } \prod_{i \in P} w_i$$

$$\iff \prod_{i \in P} w_i = \sqrt{\prod_{i \in [k]} w_i}$$

$$\iff \prod_{i \notin P} w_i = \prod_{i \in P} w_i$$

So there is a one-to-one correspondence between solutions to the #T-PARTITION instance W' where $w_{k+1} \in Q$ and solutions to the original #PARTITION instance W.

Case 2: $w_{k+1} \notin Q$. Solutions now look like:

$$\prod_{i\in P} w_i - \prod_{i\in[k]} w_i \prod_{i\notin P} w_i = \sqrt{\prod_{i\in[k]} w_i}\left(\prod_{i\in[k]} w_i - 1\right)$$

One way this can be true is if $w_i = 1$ for all $i \in [k]$. We can check ahead of time if our input set W contains all ones. If it does, then there are $2^k - 2$ partitions that yield equal products (all except $P = [k]$ and $P = \emptyset$) so we can just output $2^k - 2$ as the solution and not even use our oracle. The only other way to satisfy the above expression is for $\prod_{i\in P} w_i > \prod_{i\in[k]} w_i$ which cannot happen because $P \subseteq [k]$. So there are no solutions in the case that $w_{k+1} \notin Q$.

Therefore the output of the #T-PARTITION oracle on W' is the solution to the #PARTITION problem. So #T-PARTITION is #P-hard. □

Lemma 4.11. *For every constant $c > 1$, #SUM-PARTITION is #P-hard even on instances where $\prod_i w_i \leq c$.*

Proof. We will use a #SUM-PARTITION oracle to solve #T-PARTITION given a set of Dth roots of positive integers $W = \{w_1, \ldots, w_k\}$ and a positive real number T. Notice that for every $z > 0$:

$$\prod_{i\in P} w_i - \prod_{i\notin P} w_i = z \implies \prod_{i\in P} w_i - \frac{\prod_{i\in[k]} w_i}{\prod_{i\in P} w_i} = z$$

$$\implies \exists\, j \in \mathbb{Z}^+ \text{such that } \sqrt[D]{j} - \frac{\prod_{i\in[k]} w_i}{\sqrt[D]{j}} = z$$

Above, j must be a positive integer, which tells us that the gap in products from every partition must take a particular form. This means that for a given D and W, #T-PARTITION can only be non-zero on a discrete set of possible values of $T = z$. Given z, we can find a $z' > z$ such that the above has no solutions in the interval (z, z'). Specifically, solve the above quadratic for $\sqrt[D]{j}$ (where j may or may not be an integer), let $j' = \lfloor j + 1 \rfloor > j$, and $z' = \sqrt[D]{j'} - \frac{\prod_i w_i}{\sqrt[D]{j'}}$. We use this property twice in the proof.

Define $P^z \equiv \{P \subseteq [k] \mid \prod_{i\in P} w_i - \prod_{i\notin P} w_i \geq z\}$. As described above we can find the interval (T, T') of values above T with no solutions. Then, for every $c \in (T, T')$:

$$\left|\left\{P \subseteq [k] \mid \prod_{i\in P} w_i - \prod_{i\notin P} w_i = T\right\}\right| = \left|P^T \backslash P^c\right|$$

$$= \frac{1}{T}\left(\sum_{P\in P^T\backslash P^c}\left(\prod_{i\in P} w_i - \prod_{i\notin P} w_i\right)\right)$$

$$= \frac{1}{T}\left(\sum_{P\in P^T}\left(\prod_{i\in P} w_i - \prod_{i\notin P} w_i\right) - \sum_{P\in P^c}\left(\prod_{i\in P} w_i - \prod_{i\notin P} w_i\right)\right)$$

We now show how to find $\sum_{P \in P^z} \left(\prod_{i \in P} w_i - \prod_{i \notin P} w_i \right)$ for any $z > 0$ using the #SUM-PARTITION oracle. Once we have this procedure, we can run it for $z = T$ and $z = c$ and plug the outputs into the expression above to solve the #T-PARTITION problem. We want to set the input r to the #SUM-PARTITION oracle such that:

$$\prod_{i \in P} w_i - r \cdot \prod_{i \notin P} w_i \geq 0 \iff \prod_{i \in P} w_i - \prod_{i \notin P} w_i \geq z$$

Solving this expression for r gives:

$$r_z = \frac{4 \prod_{i \in [k]} w_i}{\left(\sqrt{z^2 + 4 \prod_{i \in [k]} w_i} - z \right)^2}$$

Below we check that this setting satisfies the requirement.

$$\begin{aligned}
\prod_{i \in P} w_i - \frac{4 \prod_{i \in [k]} w_i}{\left(\sqrt{z^2 + 4 \prod_{i \in [k]} w_i} - z \right)^2} \cdot \prod_{i \notin P} w_i \geq 0 &\iff 1 - \frac{4 \left(\prod_{i \notin P} w_i \right)^2}{\left(\sqrt{z^2 + 4 \prod_{i \in [k]} w_i} - z \right)^2} \geq 0 \\
&\iff \sqrt{z^2 + 4 \prod_{i \in [k]} w_i} \geq 2 \prod_{i \notin P} w_i + z \\
&\iff 4 \prod_{i \in [k]} w_i \geq 4 \left(\prod_{i \notin P} w_i \right)^2 + 4z \prod_{i \notin P} w_i \\
&\iff \prod_{i \in P} w_i - \prod_{i \notin P} w_i \geq z
\end{aligned}$$

So we have $P^z = \left\{ P \subseteq [k] \mid \prod_{i \in P} w_i - r_z \cdot \prod_{i \notin P} w_i \geq 0 \right\}$ but this does not necessarily mean that

$$\sum_{P \in P^z} \left(\prod_{i \in P} w_i - \prod_{i \notin P} w_i \right) = \sum_{P \in P^z} \left(\prod_{i \in P} w_i - r_z \cdot \prod_{i \notin P} w_i \right)$$

The sum on the left-hand side without the r_z coefficient is what we actually need to compute. To get this we again use the discreteness of potential solutions to find $z'' \neq z$ such that $P^z = P^{z''}$. We just pick z'' from the interval (z, z') of values above z that cannot possibly contain solutions to #T-PARTITION.

Running our #SUM-PARTITION oracle for r_z and $r_{z''}$ will output:

$$\sum_{P \in P^z} \left(\prod_{i \in P} w_i - r_z \cdot \prod_{i \notin P} w_i \right)$$
$$\sum_{P \in P^z} \left(\prod_{i \in P} w_i - r_{z''} \cdot \prod_{i \notin P} w_i \right)$$

This is just a system of two equations with two unknowns and it can be solved for $\sum_{P \in P^z} \prod_{i \in P} w_i$ and $\sum_{P \in P^z} \prod_{i \notin P} w_i$ separately. Then we can reconstruct $\sum_{P \in P^z} \left(\prod_{i \in P} w_i - \prod_{i \notin P} w_i \right)$. Running this procedure for $z = T$ and $z = c$ gives us all of the information we need to count the number of solutions to the #T-PARTITION instance we were given. We can solve #T-PARTITION in polynomial time with four calls to a #SUM-PARTITION oracle. Therefore #SUM-PARTITION is $\#P$-hard. □

Now we prove that computing OptComp is $\#P$-complete.

Proof of Theorem 1.6. We have already shown that computing OptComp is $\#P$-easy. Here we prove that it is also $\#P$-hard, thereby proving $\#P$-completeness.

Given an instance D, $W = \{w_1, \ldots, w_k\}, r$ of #SUM-PARTITION, where $\forall i \in [k]$, w_i is the Dth root of an integer and $\prod_i w_i \leq c$, set $\epsilon_i = \ln(w_i)\ \forall i \in [k]$, $\delta_1 = \delta_2 = \ldots \delta_k = 0$ and $\epsilon_g = \ln(r)$. Note that $\sum_i \epsilon_i = \ln\left(\prod_i w_i\right) \leq \ln(c)$. Since we can take c to be an arbitrary constant greater than 1, we can ensure that $\sum_i \epsilon_i \leq \epsilon$ for an arbitrary $\epsilon > 0$.

Again we will use the version of OptComp that takes ϵ_g as input and outputs δ_g. After using an OptComp oracle to find δ_g we know the optimal composition equation 1 from Theorem 1.5 is satisfied:

$$\frac{1}{\prod_{i=1}^{k} (1 + e^{\epsilon_i})} \sum_{S \subseteq \{1,\ldots,k\}} \max \left\{ e^{\sum_{i \in S} \epsilon_i} - e^{\epsilon_g} \cdot e^{\sum_{i \notin S} \epsilon_i}, 0 \right\} = 1 - \frac{1 - \delta_g}{\prod_{i=1}^{k} (1 - \delta_i)} = \delta_g$$

Thus we can compute:

$$\begin{aligned} \delta_g \cdot \prod_{i=1}^{k} (1 + e^{\epsilon_i}) &= \sum_{S \subseteq \{1,\ldots,k\}} \max \left\{ e^{\sum_{i \in S} \epsilon_i} - e^{\epsilon_g} \cdot e^{\sum_{i \notin S} \epsilon_i}, 0 \right\} \\ &= \sum_{S \subseteq \{1,\ldots,k\}} \max \left\{ \prod_{i \in S} w_i - r \cdot \prod_{i \notin S} w_i, 0 \right\} \end{aligned}$$

This last expression is exactly the solution to the instance of #SUM-PARTITION we were given. We solved #SUM-PARTITION in polynomial time with one call to an OptComp oracle. Therefore computing OptComp is $\#P$-hard. □

5 Approximation of OptComp

Although we cannot hope to efficiently compute the optimal composition for a general set of differentially private algorithms (assuming P≠NP or even FP≠ #P), we show in this section that we can approximate OptComp arbitrarily well in polynomial time.

Theorem 1.7 (restated). *There is a polynomial-time algorithm that given $\epsilon_1, \ldots, \epsilon_k \geq 0, \delta_1, \ldots \delta_k, \delta_g \in [0, 1)$, and $\eta > 0$, outputs ϵ^* where*

$$\text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), \delta_g) \leq \epsilon^* \leq \text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), e^{-\eta/2} \cdot \delta_g) + \eta$$

The algorithm runs in $O\left(\log\left(\frac{k}{\eta}\sum_{i=1}^k \epsilon_i\right)\frac{k^2}{\eta}\sum_{i=1}^k \epsilon_i\right)$ time assuming constant-time arithmetic operations.

We prove this theorem using the following three lemmas:

Lemma 5.1. *Given non-negative integers $a_1, \ldots, a_k$, B and weights $w_1, \ldots, w_k \in \mathbb{R}$, one can compute*

$$\sum_{\substack{S \subseteq [k] \text{ s.t.} \\ \sum_{i \in S} a_i \leq B}} \prod_{i \in S} w_i$$

in time $O(Bk)$.

Notice that the constraint in Lemma 5.1 is the same one that characterizes knapsack problems. Indeed, the algorithm we give for computing $\sum_{S \subseteq [k]} \prod_{i \in S} w_i$ is a slight modification of the known pseudo-polynomial time algorithm for counting knapsack solutions, which uses dynamic programming. Next we show that we can use this algorithm to approximate OptComp.

Lemma 5.2. *Given $\epsilon_1, \ldots, \epsilon_k, \epsilon^* \geq 0, \delta_1, \ldots \delta_k, \delta_g \in [0, 1)$, if $\epsilon_i = a_i \epsilon_0$ $\forall i \in \{1, \ldots, k\}$ for non-negative integers a_i and some $\epsilon_0 > 0$, then there is an algorithm that determines whether or not $\text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), \delta_g) \leq \epsilon^*$ that runs in time $O\left(\frac{k}{\epsilon_0}\sum_{i=1}^k \epsilon_i\right)$.*

In other words, if the ϵ values we are given are all integer multiples of some ϵ_0, we can determine whether or not the composition of those privacy parameters is (ϵ^*, δ_g)-DP in pseudo-polynomial time for every $\epsilon^* \geq 0$. This means that given any inputs to OptComp, if we discretize and polynomially bound the ϵ_i's, then we can check if the parameters satisfy any global privacy guarantee in polynomial time. Once we have this, we only need to run binary search over values of ϵ^* to find the optimal one. In other words, we can solve OptComp exactly for a slightly different set of ϵ_i's. The next lemma tells us that the output of OptComp on this different set of ϵ_i's can be used as a good approximation to OptComp on the original ϵ_i's.

Lemma 5.3. *For all $\epsilon_1, \ldots, \epsilon_k, c \geq 0$ and $\delta_1, \ldots, \delta_k, \delta_g \in [0, 1)$:*

$$\text{OptComp}((\epsilon_1 + c, \delta_1), \ldots, (\epsilon_k + c, \delta_k), \delta_g) \leq \text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), e^{-kc/2} \cdot \delta_g) + kc$$

Next we prove the three lemmas and then show that Theorem 1.7 follows.

Proof of Lemma 5.1. We modify Dyer's algorithm for approximately counting solutions to knapsack problems [Dye03]. The algorithm uses dynamic programming. Given non-negative integers $a_1, \ldots, a_k$, B and weights $w_1, \ldots, w_k \in \mathbb{R}$, define

$$F(r,s) = \sum_{\substack{S \subseteq [r] \text{ s.t.} \\ \sum_{i \in S} a_i \leq s}} \prod_{i \in S} w_i$$

We want to compute $F(k, B)$. We can find this by tabulating $F(r,s)$ for $(0 \leq r \leq k,\ 0 \leq s \leq B)$ using the recursion:

$$F(r,s) = \begin{cases} 1 & \text{if } r = 0 \\ F(r-1,s) + w_r F(r-1, s-a_r) & \text{if } r > 0 \text{ and } a_r \leq s \\ F(r-1,s) & \text{if } r > 0 \text{ and } \ a_r > s \end{cases}$$

Each cell $F(r,s)$ in the table can be computed in constant time given earlier cells $F(r',s')$ where $r' < r$. Thus filling the entire table takes time $O(Bk)$. □

Proof of Lemma 5.2. Given $\epsilon_1, \ldots, \epsilon_k, \epsilon^* \geq 0$ such that $\epsilon_i = a_i \epsilon_0\ \forall i \in \{1, \ldots, k\}$ for non-negative integers a_i and some $\epsilon_0 > 0$, and $\delta_1, \ldots \delta_k, \delta_g \in [0,1)$, Theorem 1.5 tells us that answering whether or not

$$\text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), \delta_g) \leq \epsilon^*$$

is equivalent to answering whether or not the following inequality holds:

$$\frac{1}{\prod_{i=1}^k (1 + e^{\epsilon_i})} \sum_{S \subseteq \{1,\ldots,k\}} \max \left\{ e^{\sum_{i \in S} \epsilon_i} - e^{\epsilon^*} \cdot e^{\sum_{i \notin S} \epsilon_i}, 0 \right\} \leq 1 - \frac{1 - \delta_g}{\prod_{i=1}^k (1 - \delta_i)}$$

The right-hand side and the coefficient on the sum are easy to compute given the inputs so in order to check the inequality, we will show how to compute the sum. Define

$$\begin{aligned} K &= \left\{ T \subseteq [k] \mid \sum_{i \notin T} \epsilon_i \geq \epsilon^* + \sum_{i \in T} \epsilon_i \right\} \\ &= \left\{ T \subseteq [k] \mid \sum_{i \in T} \epsilon_i \leq \left(\sum_{i=1}^k \epsilon_i - \epsilon^* \right) / 2 \right\} \\ &= \left\{ T \subseteq [k] \mid \sum_{i \in T} a_i \leq B \right\} \text{ for } B = \left\lfloor \left(\sum_{i=1}^k \epsilon_i - \epsilon^* \right) / 2\epsilon_0 \right\rfloor \end{aligned}$$

and observe that by setting $T = S^{\mathsf{c}}$, we have

$$\sum_{S \subseteq \{1,\ldots,k\}} \max \left\{ e^{\sum_{i \in S} \epsilon_i} - e^{\epsilon^*} \cdot e^{\sum_{i \notin S} \epsilon_i}, 0 \right\} = \sum_{T \in K} \left(\left(e^{\sum_{i=1}^k \epsilon_i} \cdot \prod_{i \in T} e^{-\epsilon_i} \right) - \left(e^{\epsilon^*} \cdot \prod_{i \in T} e^{\epsilon_i} \right) \right)$$

We just need to compute this last expression and we can do it for each term separately since K is a set of knapsack solutions. Specifically, setting $w_i = e^{-\epsilon_i}\ \forall i \in [k]$, Lemma 5.1 tells us that we can compute $\sum_{T\subseteq[k]} \prod_{i\in T} w_i$ subject to $\sum_{i\in T} a_i \leq B$, which is equivalent to $\sum_{T\in K} \prod_{i\in T} e^{-\epsilon_i}$.

To compute $\sum_{T\in K} \prod_{i\in T} e^{\epsilon_i}$, we instead set $w_i = e^{\epsilon_i}$ and run the same procedure. Since we used the algorithm from Lemma 5.1, the running time is $O(Bk) = O\left(\frac{k}{\epsilon_0}\sum_{i=1}^k \epsilon_i\right)$ □

Proof of Lemma 5.3. Let $\text{OptComp}((\epsilon_1,\delta_1),\ldots,(\epsilon_k,\delta_k), e^{-kc/2}\cdot\delta_g) = \epsilon_g$. From Equation 1 in Theorem 1.5 we know:

$$\frac{1}{\prod_{i=1}^k (1+e^{\epsilon_i})} \sum_{S\subseteq\{1,\ldots,k\}} \max\left\{ e^{\sum_{i\in S}\epsilon_i} - e^{\epsilon_g}\cdot e^{\sum_{i\notin S}\epsilon_i}, 0\right\} \leq 1 - \frac{1-e^{-kc/2}\cdot\delta_g}{\prod_{i=1}^k(1-\delta_i)}$$

Multiplying both sides by $e^{kc/2}$ gives:

$$\begin{aligned}\frac{e^{kc/2}}{\prod_{i=1}^k (1+e^{\epsilon_i})} \sum_{S\subseteq\{1,\ldots,k\}} \max\left\{ e^{\sum_{i\in S}\epsilon_i} - e^{\epsilon_g}\cdot e^{\sum_{i\notin S}\epsilon_i}, 0\right\} &\leq e^{kc/2}\cdot\left(1 - \frac{1-e^{-kc/2}\cdot\delta_g}{\prod_{i=1}^k(1-\delta_i)}\right)\\ &\leq 1 - \frac{1-\delta_g}{\prod_{i=1}^k(1-\delta_i)}\end{aligned}$$

The above inequality together with Theorem 1.5 means that showing the following will complete the proof:

$$\sum_{S\subseteq\{1,\ldots,k\}} \max\left\{ e^{\sum_{i\in S}(\epsilon_i+c)} - e^{\epsilon_g+kc}\cdot e^{\sum_{i\notin S}(\epsilon_i+c)}, 0\right\} \leq \frac{e^{kc/2}\cdot\prod_{i=1}^k(1+e^{\epsilon_i+c})}{\prod_{i=1}^k(1+e^{\epsilon_i})} \sum_{S\subseteq\{1,\ldots,k\}} \max\left\{ e^{\sum_{i\in S}\epsilon_i} - e^{\epsilon_g}\cdot e^{\sum_{i\notin S}\epsilon_i}, 0\right\}$$

Since $(1+e^{\epsilon_i+c})/(1+e^{\epsilon_i}) \geq e^{c/2}$ for every $\epsilon_i > 0$, it suffices to show:

$$\sum_{S\subseteq\{1,\ldots,k\}} \max\left\{ e^{\sum_{i\in S}(\epsilon_i+c)} - e^{\epsilon_g+kc}\cdot e^{\sum_{i\notin S}(\epsilon_i+c)}, 0\right\} \leq \sum_{S\subseteq\{1,\ldots,k\}} e^{kc}\cdot\max\left\{ e^{\sum_{i\in S}\epsilon_i} - e^{\epsilon_g}\cdot e^{\sum_{i\notin S}\epsilon_i}, 0\right\}$$

This inequality holds term by term. If a right-hand term is zero $\left(\sum_{i\in S}\epsilon_i \leq \epsilon_g + \sum_{i\notin S}\epsilon_i\right)$, then so is the corresponding left-hand term $\left(\sum_{i\in S}(\epsilon_i+c) \leq \epsilon_g + kc + \sum_{i\notin S}(\epsilon_i+c)\right)$. For the nonzero terms, the factor of e^{kc} ensures that the right-hand terms are larger than the left-hand terms. □

Proof of Theorem 1.7. Lemma 5.2 tells us that we can determine whether a set of privacy parameters satisfies some global differential privacy guarantee if the ϵ values are discretized. Notice that then we can solve OptComp exactly for a discretized set of ϵ values by running binary search over values of ϵ^* until we find the minimum ϵ^* that satisfies (ϵ^*, δ_g)-DP.

Given $\epsilon_1,\ldots,\epsilon_k,\epsilon^*$, and an additive error parameter $\eta > 0$, set $a_i = \left\lfloor \frac{k}{\eta}\epsilon_i \right\rfloor, \epsilon_i' = \frac{\eta}{k}\cdot a_i\ \forall i \in [k]$. With these settings, the a_i's are non-negative integers and the ϵ_i' values are all integer multiples of $\epsilon_0 = \eta/k$. Lemma 5.2 tells us that we can determine if the new privacy parameters with ϵ'

values satisfy (ϵ^*, δ_g)-DP in time $O\left(\frac{k^2}{\eta}\sum_{i=1}^{k}\epsilon_i\right)$. Running binary search over values of ϵ^* will then compute $\text{OptComp}((\epsilon'_1, \delta_1), \ldots, (\epsilon'_k, \delta_k), \delta_g) = \epsilon'_g$ exactly in time $O\left(\log\left(\frac{k}{\eta}\sum_{i=1}^{k}\epsilon_i\right)\frac{k^2}{\eta}\sum_{i=1}^{k}\epsilon_i\right)$

Notice that $\epsilon_i - \eta/k \leq \epsilon'_i \leq \epsilon_i \ \forall i \in [k]$. Lemma 5.3 says that the outputted ϵ'_g is at most $\text{OptComp}((\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k), e^{-\eta/2} \cdot \delta_g) + \eta$ as desired. □

References

[Cro11] Mercè Crosas. The Dataverse Network®: an open-source application for sharing, discovering and preserving data. *D-lib Magazine*, 17.1, 2, 2011.

[DKMMN06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: privacy via distributed noise generation. *Advances in Cryptology-EUROCRYPT*, pages 486-503, 2006.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis: *Third Theory of Cryptography Conference (TCC'06)*, pages 265-284, 2006.

[DR13] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9.3-4, pages 211-407, 2013.

[DRV10] Cynthia Dwork, Guy N. Rothblum, and Salil Vadhan. Boosting and differential privacy. *In Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS'10), 2010 51st Annual IEEE Symposium.*, IEEE, 2010.

[Dye03] Martin Dyer. Approximate counting by dynamic programming. *Proceedings of the 35th annual ACM Symposium on Theory of Computing (STOC'13)*, ACM, pages 693-699, 2003.

[Ehr00] Matthias Ehrgott. Approximation algorithms for combinatorial multicriteria optimization problems. *International Transactions in Operational Research*, Wiley Online Library, pages 5-31, 2000.

[Kin07] Gary King. An introduction to the Dataverse Network as an infrastructure for data sharing. *Sociological Methods & Research*, 36.2, pages 173-199, 2007.

[KOV15] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The Composition Theorem for Differential Privacy. *Proceedings of the 32nd International Conference on Machine Learning, (ICML'15)*, 37, pages 1376-1385, 2015.

[War65] Stanley L. Warner. Randomized Response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60.309, pages 63-69, 1965.