Privacy Proofs for OpenDP: Bounded Sum with Known n

Sílvia Casacuberta

Summer 2021

Contents

1 Algorithm Implementation 1
1.1 Code in Rust . . . 1
1.2 Pseudocode in Python . . . 2

2 Proof 3
2.1 Symmetric Distance . . . 3
2.2 First proof: using the path property (adjacent pairs approach) . . . 4
2.3 Second proof: direct method (all pairs approach) . . . 5
2.3.1 General inequality . . . 5

1 Algorithm Implementation

1.1 Code in Rust

The current OpenDP library contains the transformation `make_bounded_sum_n` implementing the bounded sum function with known n. This is defined in lines 53-68 of the file `sum.rs` in the Git repository[1] (https://github.com/opendp/opendp/blob/b936c74223b4e319698fa51837b5f8f40f3126d3/rust/opendp/src/trans/sum.rs#L53-L68).

```
pub fn make_bounded_sum_n<T>(
    lower: T, upper: T, length: usize
) -> Fallible<Transformation<SizedDomain<VectorDomain<IntervalDomain<T>>>, AllDomain<T>, SymmetricDistance, AbsoluteDistance<T>>>
    where T: DistanceConstant + Sub<Output=T>,
          for <'a> T: Sum<&'a T> {

    Ok(Transformation::new(
        SizedDomain::new(VectorDomain::new(IntervalDomain::new(
            Bound::Included(lower.clone()), Bound::Included(upper.clone()))?), length),
        AllDomain::new(),
        Function::new(|arg: &Vec<T>| arg.iter().sum()),
        SymmetricDistance::default(),
        AbsoluteDistance::default(),
        // d_out >= d_in * (M - m) / 2
        StabilityRelation::new_from_constant((upper - lower) / T::distance_cast(2)?)))
}
```

[1] As of July 1, 2021.

Update: after conversations with Mike about the possible overflow problems in Bounded Sum, the Rust code has been updated. The new version can be found at `https://github.com/opendp/opendp/blob/53e8d67b8dde4425930fb8bc397c126cd4f18370/rust/opendp/src/trans/sum.rs#L36-L55`. However, we also keep the code snippet above because this change is still a pull request and hence not yet part of the OpenDP library.[2] However, the proof below now refers to this most updated version.

```rust
pub fn make_bounded_sum_n<T>(
    lower: T, upper: T, length: usize
) -> Fallible<Transformation<SizedDomain<VectorDomain<IntervalDomain<T>>>, AllDomain<T>, SymmetricDistance, AbsoluteDistance<T>>>
    where T: DistanceConstant<IntDistance> + Sub<Output=T>, for <'a> T: Sum<&'a T> + ExactIntCast<usize> + CheckedMul,
          IntDistance: InfCast<T> {
    let length_ = T::exact_int_cast(length)?;
    if lower.checked_mul(&length_).is_none()
        || upper.checked_mul(&length_).is_none() {
        return fallible!(MakeTransformation, "Detected potential for overflow when computing function.")
    }
    Ok(Transformation::new(
        SizedDomain::new(VectorDomain::new(IntervalDomain::new(
            Bound::Included(lower.clone()), Bound::Included(upper.clone()))?), length),
        AllDomain::new(),
        Function::new(|arg: &Vec<T>| arg.iter().sum()),
        SymmetricDistance::default(),
        AbsoluteDistance::default(),
        // d_out >= d_in * (M - m) / 2
        StabilityRelation::new_from_constant((upper - lower) / T::exact_int_cast(2)?)))
}
```

1.2 Pseudocode in Python

We present a simplified Python-like pseudocode of the Rust implementation below. The necessary definitions for the pseudocode can be found at "List of definitions used in the pseudocode".

Preconditions

To ensure the correctness of the output, we require the following preconditions:

- **User-specified types:**
  - Variable `n` must be of type `usize`.
  - Type `T` must have traits `DistanceConstant(IntDistance)`, `TotalOrd`,[3] `CheckedMul`, `Sum(Output=T)`, `Sub(Output=T)`, and `ExactIntCast(usize)`.
  - `IntDistance` must have trait `InfCast(T)`. **Question:** Same question that Connor asked in `make_count` – this is not needed for the proof.

Postconditions

- Either a valid `Transformation` is returned or an error is returned.

[2] As of July 20.

[3] For now, the OpenDP library only implements `PartialOrd`, but `TotalOrd` will soon be implemented. Then, `TotalOrd` will be redundant, since the trait `TotalOrd` is part of the trait `DistanceConstant`.

```
1 def MakeBoundedSumN(L: T, U: T, n: usize):
2     input_domain = SizedDomain(VectorDomain(IntervalDomain(L, U)), n)
3     output_domain = AllDomain(T)
4     input_metric = SymmetricDistance()
5     output_metric = AbsoluteDistance(T)
6 
7     n_ = exact_int_cast(n, T)
8     if checked_mul(L, n_).is_none or checked_mul(U, n_).is_none:
9         raise Exception('Potential overflow')
10 
11     def Relation(d_in: u32, d_out: u32) -> bool:
12         return d_out >= d_in*(U-L)/exact_int_cast(2, T)
13 
14     def function(data: Vec(T)) -> T:
15         return data.iter().sum()
16     return Transformation(input_domain, output_domain, function,
    input_metric, output_metric, stability_relation = Relation)
```

2 Proof

2.1 Symmetric Distance

Theorem 1. *For every setting of the input parameters* `(L, U, n)` *to* `MakeBoundedSumN`*, the transformation returned by* `MakeBoundedSumN` *has the following properties:*

1. (Appropriate output domain). *For every element v in* `input_domain`*,* `function`(v) *is in* `output_domain`*.*

2. (Domain-metric compatibility). *The domain* `input_domain` *matches one of the possible domains listed in the definition of* `input_metric`*, and likewise* `output_domain` *matches one of the possible domains listed in the definition of* `output_metric`*.*

3. (Stability guarantee). *For every pair of elements v, w in* `input_domain` *and for every pair* (`d_in`, `d_out`)*, where* `d_in` *is of the associated type for* `input_metric` *and* `d_out` *is the associated type for* `output_metric`*, if v, w are d_{in}-close under* `input_metric` *and* `Relation`(`d_in`, `d_out`) = `True`*, then* `function`(v)*,* `function`(v) *are d_{out}-close under* `output_metric`*.*

Proof. **(Appropriate output domain).** In the case of `MakeBoundedSumN`, this corresponds to showing that for every vector v in `SizedDomain(VectorDomain(IntervalDomain (L, U)), n)`, where `L` and `U` have type `T`, `function`(v) belongs to `AllDomain(T)`. The output correctness follows from the type signature of `function` as defined in line 14 and from the overflow check done through the `checked_mul` function in line 9. The latter ensures that `function(v)` is contained within the interval `[get_min_value(T), get_max_value(T)]`, and hence prevents any overflow from occurring in line 15. Otherwise, an exception for potential overflow will be raised, as described in line 9. The former automatically enforces that `function(v)` has type `T`. Since the Rust code successfully compiles, by the type signature the appropriate output domain property must hold. Otherwise, the code will raise an exception for incorrect input type.

Question: Should I say something about the `exact_int_cast` check?

(Domain-metric compatibility). For `MakeBoundedSumN`, this corresponds to showing that `SizedDomain(VectorDomain(IntervalDomain (L, U)), n)` is compatible with symmetric distance, and that `AllDomain(T)` is compatible with absolute distance. The latter follows directly from the list of compatible domains in the definition of absolute distance, as described in "List of definitions used in the pseudocode". The former follows from the compatibility of symmetric distance and `VectorDomain(D)` as stated in the definition of symmetric distance along with the fact that `SizedDomain(VectorDomain(D))` is a subdomain of `VectorDomain(D)`. By Theorem 2.1 in "List of definitions used in the pseudocode", this implies that `SizedDomain(VectorDomain(D))` is compatible with symmetric distance as well.

(silvia) Flag: this is an example of the subdomain issues that we have been discussing during the week of July 19. Hence this paragraph might need some phrasing updates when the compatibility pairing constructor and the subdomain trait are implemented.

(Stability guarantee). Throughout the stability guarantee proof, we can assume that $\texttt{function}(v)$ and $\texttt{function}(w)$ are in the correct output domain, by the *appropriate output domain property* shown above.

Since by assumption $\texttt{Relation}(\texttt{d\_in}, \texttt{d\_out}) = \texttt{True}$, by the `MakeBoundedSumN` stability relation (as defined in line 11 in the pseudocode), we have that $\texttt{d\_out} \geq \texttt{d\_in} \cdot (\texttt{U} - \texttt{L})/2$. Moreover, v, w are assumed to be `d_in`-close. By the definition of the symmetric difference metric, this is equivalent to stating that $d_{Sym}(v, w) = |\text{MultiSet}(v) \Delta \text{MultiSet}(w)| \leq \texttt{d\_in}$.

Further, applying the histogram notation,[4] it follows that

$$d_{Sym}(v, w) = \|h_v - h_w\|_1 = \sum_z |h_v(z) - h_w(z)| \leq \texttt{d\_in}.$$

We want to show that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq d_{Sym}(v, w) \cdot \frac{\texttt{U-L}}{2}.$$

This would imply that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq d_{Sym}(v, w) \cdot \frac{\texttt{U-L}}{2} \leq \texttt{d\_in} \cdot \frac{\texttt{U-L}}{2}, \tag{1}$$

and by the stability relation this will imply that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq \texttt{d\_out}, \tag{2}$$

as we want to see. □

2.2 First proof: using the path property (adjacent pairs approach)

To show that $d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq d_{Sym}(v, w) \cdot \frac{\texttt{U-L}}{2}$, we will use the three lemmas described in the section "The path property of symmetric distance on sized domains" from Section 4.2 in the document "List of definitions used in the proofs". With these three lemmas, which are applicable to `MakeBoundedSumN` because `input_domain` is a a sized domain and `input_metric` is symmetric distance, it suffices to show the following: For all vectors $x, y \in \texttt{input\_domain}$ such that $d_{Sym}(x, y) = 2$, it follows that

$$d_{Abs}(\texttt{function}(x), \texttt{function}(y)) \leq \texttt{U} - \texttt{L}.$$

[4] Note that there is a bijection between multisets and histograms, which is why the proof can be carried out with either notion. For further details, please consult https://www.overleaf.com/project/60d214e390b337703d200982.

By Lemma 4.3 from "List of definitions used in the proofs", we know that vectors x, y only differ on one element, given that, by assumption, $d_{Sym}(x, y) = 2$. Wlog, let this different element be the k-th element of x and y, where $x_k = \alpha$, $y_k = \beta$ with $\alpha \neq \beta$.[5] Then,

$$d_{Abs}(\texttt{function}(x), \texttt{function}(y)) = |\texttt{function}(x) - \texttt{function}(y)| =$$

$$= \Big|\sum_{i=0}^{\texttt{n}-1} x_i - \sum_{i=0}^{\texttt{n}-1} y_i\Big| = \Big|\sum_{i=0}^{\texttt{n}-1}(x_i - y_i)\Big| = |\alpha - \beta| \leq |\texttt{U-L}| = \texttt{U-L},$$

since $\texttt{U} \geq \texttt{L}$. Therefore, applying Lemma 4.4 from "List of definitions used in the proofs", it follows that `function` is (U-L)/2-stable. By definition, this implies that for any $v, w \in$ `input_domain`,

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq d_{Sym}(v, w) \cdot (\texttt{U-L})/2.$$

Lastly, by Equations 1 and 2 this implies that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq \texttt{d\_out},$$

as we want to prove.

(silvia) Flag: this will be updated to the more general notion of path property (through shortest path metric on a graph), but this matches the current version of the proofs document.

2.3 Second proof: direct method (all pairs approach)

2.3.1 General inequality

The general statement that we will need to prove is the following. For any elements $a_1, \ldots, a_n \in$ `[L, U]` and $b_1, \ldots, b_n$,

$$\Big|\sum_i a_i b_i\Big| \leq \frac{a_{\max} - a_{\min}}{2} \cdot \Big(\sum_i |b_i|\Big).$$

Note that this corresponds to the tightest possible `[L, U]` interval.

Let u denote the vector formed by all the elements of v and w *without multiplicities* (i.e., u contains exactly once each of the elements in $\mathrm{MultiSet}(v) \cup \mathrm{MultiSet}(w)$, in any order). Let u_i denote the i-th element of u, and similarly for v and w, and let m denote $\texttt{len}(u)$. Then, by definition,

$$d_{Sym}(v, w) = \sum_z \Big|h_v(z) - h_w(z)\Big| = \sum_i \Big|h_v(u_i) - h_w(u_i)\Big|;$$

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) = \Big|\texttt{function}(v) - \texttt{function}(w)\Big| = \Big|\sum_i v_i - \sum_i w_i\Big| =$$

$$= \Big|\sum_i u_i \cdot h_v(u_i) - \sum_i u_i \cdot h_w(u_i)\Big| = \Big|\sum_i u_i \cdot (h_v(u_i) - h_w(u_i))\Big|.$$

Because by assumption $v, w \in$ `input_domain = SizedDomain(VectorDomain(IntervalDomain (L, U)), n)`, we know that $\texttt{len}(v) = \texttt{len}(w) = \texttt{n}$. Therefore,

[5] The first element of a vector is indexed by 0.

$$\sum_i (h_v(u_i) - h_w(u_i)) = \mathtt{n} - \mathtt{n} = 0. \tag{3}$$

We now separate the positive values from the negative ones by defining vectors x, y, λ and μ as follows. Let

$$h_v(u_{k_1}) - h_w(u_{k_1}) \leq \ldots \leq 0 \leq h_v(u_{k_m}) - h_w(u_{k_m})$$

be the sequence of the $\{h_v(u_i) - h_w(u_i)\}$ in increasing order. Let s be the smallest value such that $h_v(u_{k_s}) - h_w(u_{k_s})$ is greater or equal to 0 (we set $t = m$ if all the values are negative). Then, we define the vector entries of x, y, λ, μ as

$$x_j = h_v(u_{k_j}) - h_w(u_{k_j}),$$

$$\lambda_j = u_j,$$

for $s \leq j \leq m$, and

$$y_j = h_v(u_{k_j}) - h_w(u_{k_j}),$$

$$\mu_j = u_j$$

for $0 \leq j < s$.[6] That is, x contains all of the positive values and y all of the negative ones.

Let r denote the length of vectors x and λ as constructed above, and by construction s denotes the length of vectors y and μ above (where $r + s = m$). Hence we obtain the values $x_1, \ldots, x_r \geq 0$ and $y_1, \ldots, y_s \leq$ for some $r, s \in \mathbb{Z}$, such that

$$\sum_i x_i + \sum_j y_j = 0 \quad \text{and so} \quad \sum_i x_i = \sum_j |y_j|,$$

by Equation 3. Then,

$$d_{Abs}(\mathtt{function}(v), \mathtt{function}(w)) = \left| \sum_i u_i \cdot (h_v(u_i) - h_w(u_i)) \right| =$$

$$= |\lambda_1 x_1 + \cdots + \lambda_r x_r + \mu_1 y_1 + \cdots \mu_s y_s| = \left| \overline{\lambda} \sum_i x_i + \overline{\mu} \sum_j y_j \right| =$$

$$= \frac{|\overline{\lambda} - \overline{\mu}|}{2} \left(\sum_i x_i + \sum_j |y_j| \right) = |\overline{\lambda} - \overline{\mu}| \sum_i x_i,$$

where

$$\overline{\lambda} = \frac{\sum \lambda_i x_i}{\sum x_i}, \quad \overline{\mu} = \frac{\sum \mu_j y_j}{\sum y_j} = \frac{\sum \mu_j |y_j|}{\sum |y_j|};$$

i.e., they correspond to the weighted arithmetic mean.

By definition of the `input_domain`, the entries of v and w are contained within the interval `[L, U]`, and hence $\mathtt{U} \geq \max\{\lambda_i, \mu_j\}$ and $\mathtt{L} \leq \min\{\lambda_i, \mu_j\}$. Then,

$$\frac{\mathtt{U\text{-}L}}{2} \left(\sum_i x_i + \sum_j |y_j| \right) = \frac{\mathtt{U\text{-}L}}{2} \cdot 2 \sum_i x_i = (\mathtt{U\text{-}L}) \sum_i x_i.$$

[6] It is not necessary that the entries of x_j and y_j are ordered; only that they only contain positive and negative values, respectively, and that the λ and μ values match their corresponding indices.

Since $|\overline{\lambda} - \overline{\mu}| \leq \texttt{U-L}$, it follows that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) = \frac{\overline{\lambda} - \overline{\mu}}{2}\Big(\sum_i x_i + \sum_j |y_j|\Big) = \frac{\overline{\lambda} - \overline{\mu}}{2}\Big(\sum_i x_i - \sum_j y_j\Big) =$$

$$\frac{\overline{\lambda} - \overline{\mu}}{2}\Big(\sum_i |h_v(u_i) - h_w(u_i)|\Big) = \frac{\overline{\lambda} - \overline{\mu}}{2} \cdot d_{Sym}(v, w) \leq \frac{\texttt{U-L}}{2} \cdot d_{Sym}(v, w).$$

Hence,

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq \frac{\texttt{U-L}}{2} \cdot d_{Sym}(v, w),$$

as we wanted to show.