Privacy Proofs for OpenDP: Clamping

Sílvia Casacuberta

June 2021

1 Algorithm Implementation

1.1 Code in Rust

The current OpenDP library contains the `make_clamp_vec` function implementing the clamping function. This is defined in lines 25-38 of the file `manipulation.rs` in the Git repository[1] (https://github.com/opendp/opendp/blob/58feb788ec78ce739caaf3cad8471c79fd5e7132/rust/opendp/src/trans/manipulation.rs#L25-L38).

```rust
pub fn make_clamp_vec<M, T>(lower: T, upper: T) -> Fallible<Transformation<VectorDomain<AllDomain<T>>, VectorDomain<IntervalDomain<T>>, M, M>>
    where M: Metric,
          T: 'static + Clone + PartialOrd,
          M::Distance: DistanceConstant + One {
    if lower > upper { return fallible!(MakeTransformation, "lower may not be greater than upper") }
    Ok(Transformation::new(
        VectorDomain::new_all(),
        VectorDomain::new(IntervalDomain::new(Bound::Included(lower.clone()), Bound::Included(upper.clone()))),
        Function::new(move |arg: &Vec<T>| arg.iter().map(|e| clamp(&lower, &upper, e)).collect()),
        M::default(),
        M::default(),
        // clamping has a c-stability of one, as well as a lipschitz constant of one
        StabilityRelation::new_from_constant(M::Distance::one())))
}
```

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".

(silvia) We could generalize the input domain below from float to a any general (unspecified) type that admits total ordering, and then add the corresponding precondition to assert that the type `T` for L and U has the trait `TotalOrd`. Note that partial ordering is not enough, and we might not be able to clamp in the domain. However, TotalOrd is still not implemented in the OpenDP library. For now, float is enough.

[1] As of June 16, 2021. Since then, the code has been updated to include a more general clampable domain, which is not yet finished.

Preconditions

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

(mike) The input metric can just be hardcoded to SymmetricDistance. This means there's no generic MI, so the precondition can be dropped (silvia) Same for output metric, no? Also, wouldn't it be safer not to hardcode it in case more metrics are added to the library in the future? (mike) After recent talks, I think we're thinking this should go in the preconditions. We can expand the proof in the future if we get more metrics. (silvia) Agreed

(mike) The type T can be any type that implements PartialOrd- for example floats or ints

(silvia) Fixed, but I think it should be "implements TotalOrd", no? (mike) Yes n I'm still seeing references to float throughout this document though. You should be able to just refer to T, where T is qualified by the trait bounds. (silvia) Agreed

(silvia) The domain/metric preconditions will be added once Prof. Vadhan agrees

- The type `T` must implement TotalOrd – for example, `floats` or `ints`.

```
1  def MakeClamp(L: T, U: T, metric):
2      if L > U: raise Exception('Invalid parameters')
3      input_domain = VectorDomain(AllDomain(T))
4      output_domain = VectorDomain(IntervalDomain(L, U))
5      input_metric = SymmetricDistance()
6      output_metric = SymmetricDistance()
7      def stability_relation(d_in: u32, d_out: u32) -> bool:
8          return d_out >= d_in*1
9
10     def function(data: Vec(T)) -> Vec(T):
11         def clamp(x: T) -> T:
12             return max(min(x, U), L)
13         return list(map(clamp, data))
14     return Transformation(input_domain, output_domain, function,
       input_metric, output_metric, stability_relation)
```

Conditions as specified in the pseudocode

(silvia) Redundant section with the pseudocode, but perhaps useful.

- Input domain: domain of all vectors of elements of type `T`.[2]
- Output domain: domain of all vectors of elements in `IntervalDomain(L, U)`, where `L` and `U` are of type `T`and stand for *lower bound* and *upper bound*, respectively.[3]
- Function: given v of type `Vec(T)`, it returns a list where each vector entry v_i has type `float` and is equal to `clamp`(v_i), as defined in line 11 of the pseudocode (Section 1.2).
- Input metric: symmetric distance.[4]

[2] In the future, this will be changed to the domain of all vectors of elements from an arbitrary data domain implementing total ordering (more concretely, for a generic type `T` that must have traits `static`, `Clone`, and `TotalOrd`). However, this is not yet implemented in the OpenDP library.

[3] The compiler infers `float` from the context, which is why it is omitted.

[4] As of June 24, SubstituteDistance is no longer a metric part of the OpenDP library.

- Output metric: same as input metric (which have to be consistent – this is checked by the preconditions).

- Stability relation: for $\texttt{d\_in}, \texttt{d\_out}$ of type `u32`, the relation returns `True` if and only if $\texttt{d\_in} \leq \texttt{d\_out}$. In particular, clamping has stability parameter $c = 1$.

More concisely, the stability relation can be stated as:[5] (silvia) Perhaps more concise: $\texttt{d\_in}, \texttt{d\_out} \in \texttt{AllDomain<u32>}$

$$\text{Relation}(\texttt{d\_in}, \texttt{d\_out}) = \begin{cases} \texttt{True} & \text{if } \texttt{d\_out} \geq \texttt{d\_in} \text{ for } \texttt{d\_in}, \texttt{d\_out} \text{ of type } \texttt{u32} \\ \texttt{False} & \text{otherwise} \end{cases} \tag{1}$$

2 Proof

2.1 Symmetric Distance

Theorem 1. *For every setting of the input parameters (L, U, metric) to `MakeClamp` such that the given preconditions hold, the transformation returned by `MakeClamp` has the following properties:*

1. (Appropriate output domain). *For every vector v in the input domain, $\texttt{function}(v)$ is in the output domain.*

2. (Stability guarantee). *For every input v, w of type `Vec(float)` and for every pair $(\texttt{d\_in}, \texttt{d\_out})$ (appropriately quantified), if v, w are d_{in}-close under the symmetric distance metric and $\text{Relation}(\texttt{d\_in}, \texttt{d\_out}) = \texttt{True}$, then $\texttt{function}(v), \texttt{function}(w)$ are d_{out}-close under the symmetric distance metric.*

Proof. **(Appropriate output domain).** In the case of `MakeClamp`, this corresponds to showing that for every vector v of type `Vec(T)`, $\texttt{function}(v)$ is an element of `VectorDomain (IntervalDomain(L, U))`. For that, we need to show two things: first, that the vector entries of $\texttt{function}(v)$ have type `T`. Second, that they belong to the interval $[L, U]$.

That $\texttt{function}(v)$ has type `Vec(T)` follows from the assumption that v is in the input domain, the precondition requirement that Thas type `T`, and the type signature of `function` in line 10 of the pseudocode (Section 1.2), which takes in an element of type `Vec(T)` and returns an element of type `Vec(T)`. If the Rust code compiles correctly, then the type correctness follows from the definition of the type signature enforced by Rust. Otherwise, the code raises an exception for incorrect input type. Secondly, we need to show that the vector entries belong to the interval $[L, U]$. This follows from the definition of `clamp` in line 11. According to line 11 in the pseudocode, there are 3 possible cases to consider:

1. $\texttt{x} > U$: then `clamp` returns U.

2. $\texttt{x} \in [L, U]$: then `clamp` returns `x`.

3. $\texttt{x} < L$: then `clamp` returns L.

[5] See page 14 of *A Programming Framework for OpenDP* for more examples. Note that in the Rust implementation the metrics $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ are *not* inputs to the relation, and they are instead fixed as attributes of the transformation.

In all three cases, the returned value of type `T`is contained in the interval $[L, U]$. Hence by the returned vector as defined in line 13 of the pseudocode, $\texttt{function}(v)$ is an element of the output domain `VectorDomain(IntervalDomain(L, U))`.

Lastly, the necessary condition that $L \leq U$ is checked in line 2 of the pseudocode, hence correctness is guaranteed if no exception is raised. Both L and U have type `T`by their precondition requirement. Both the definition of `IntervalDomain` and that of the `clamp` function (line 11 in the pseudocode, which uses the `min` and `max` functions) require that the type of L, U, and of each vector entry in v admits a total ordering. In the case of `T`, this holds by the preconditions. (mike) floats don't have a total ordering. It only holds for non-null floats!

(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 $\text{Relation}(\texttt{d\_in}, \texttt{d\_out}) = \texttt{True}$, by the `MakeClamp` stability relation (as defined in Equation (1)), we have that $\texttt{d\_in} \leq \texttt{d\_out}$. 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,[6] it follows that

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

We now consider $\text{MultiSet}(\texttt{function}(v))$ and $\text{MultiSet}(\texttt{function}(w))$. For each element $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$, where z has type `float`, if $z \in \text{MultiSet}(v) \Delta \text{MultiSet}(w)$, we will assume wlog that $z \in \text{MultiSet}(v) \setminus \text{MultiSet}(w)$. We consider the following cases:

1. $z > U$ or $z < L$: then, in the former case, $\texttt{clamp}(z) = U$. First consider the case when $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$ with the same multiplicity in both multisets. Then, $|h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)| = 0$ because we have both $h_{\texttt{function}(v)}(z) = 0$ and $h_{\texttt{function}(w)}(z) = 0$. Thus the sum

$$\sum_z |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)|$$

remains invariant, because the quantity $|h_v(z) - h_w(z)|$ is added to $|h_{\texttt{function}(v)}(U) - h_{\texttt{function}(w)}(U)|$, given that $\texttt{clamp}(z) = U$.

Suppose z has multiplicity $k_v \geq 0$ in $\text{MultiSet}(v)$ and multiplicity $k_w \geq 0$ in $\text{MultiSet}(w)$, where $k_v \neq k_w$. After considering z, the value $h_{\texttt{function}(v)}(U)$ becomes $h_{\texttt{function}(v)}(U) + k_v$, and $h_{\texttt{function}(w)}(U)$ becomes $h_{\texttt{function}(w)}(U) + k_w$. Hence the quantity $|h_{\texttt{function}(v)}(U) - h_{\texttt{function}(w)}(U)|$ increases by at most $|h_v(z) - h_w(z)|$, since, by the triangle inequality,

$$\begin{aligned} &|(h_{\texttt{function}(v)}(U) + k_v) - (h_{\texttt{function}(w)}(U) + k_w)| \leq \\ &\leq |h_{\texttt{function}(v)}(U) - h_{\texttt{function}(w)}(U)| + |k_v - k_w| = \\ &= |h_{\texttt{function}(v)}(U) - h_{\texttt{function}(w)}(U)| + |h_v(z) - h_w(z)|. \end{aligned}$$

[6] See *A Programming Framework for OpenDP*, footnote 1 in page 3. 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`.

The same argument applies whenever $z < L$.

(silvia) The first subcase discussed here, i.e., when $k_v = k_w$, is also proven by the triangle inequality expression above, but it seemed clean to separate the case where the total sum remains invariant.

2. $z \in (L, U)$: then, $\texttt{clamp}(z) = z$. Since $h_v(z) = h_{\texttt{function}(v)}(z)$ and $h_v(w) = h_{\texttt{function}(w)}(z)$, it follows that $|h_v(z) - h_w(z)| = |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)|$. Hence the histogram count, i.e., the quantity
$$\sum_z |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)|$$
remains invariant.

3. $z = U$ or $z = L$: then, in the former case, $\texttt{clamp}(z) = U$. If $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$ with the same multiplicity in both multisets, then the histogram count remains invariant under the addition of element z. Otherwise, if $z \in \text{MultiSet}(v) \setminus \text{MultiSet}(w)$, or if z is in their union but with different multiplicity, then element z can increase the quantity $|h_{\texttt{function}(v)}(U) - h_{\texttt{function}(w)}(U)|$ by at most $|h_v(z) - h_w(z)|$, following the same reasoning with the triangle inequality as in case 2.

   The same argument applies whenever $z = L$.

By aggregating the three cases above, we conclude that
$$\sum_z |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)| \leq \sum_z |h_v(z) - h_w(z)|.$$

By the initial assumptions, we recall that $\texttt{d\_in} \leq \texttt{d\_out}$, and that v, w are $\texttt{d\_in}$-close. Then,
$$\sum_z |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)| \leq \sum_z |h_v(z) - h_w(z)| \leq \texttt{d\_in} \leq \texttt{d\_out}.$$

Therefore,
$$|\text{MultiSet}(\texttt{function}(v)) \Delta \text{MultiSet}(\texttt{function}(w))| \leq \texttt{d\_out},$$
as we wanted to show. $\square$

(silvia) Maybe add domain of z below the sum?

3 Preliminaries

We recall that a transformation T is a *deterministic* mapping from *datasets* to *datasets*. In Rust, a Transformation is specified by the following attributes: input domain, output domain, function, input metric, output metric, and stability relation.

For $c \in \mathbb{R}$, we say that T is *c-stable* if for all datasets x, x' in the input domain,

$$d_{\mathcal{Y}}(T(x), T(x')) \leq c \cdot d_{\mathcal{X}}(x, x'). \tag{2}$$

The *metrics* $d_{\mathcal{X}}, d_{\mathcal{Y}}$ (i.e., functions that return the distance between each pair of points in a set) should not be mistaken with the actual distances `d_in`, `d_out`. In other words, the type of the evaluation of the $\mathcal{X}(\cdot)$ function is equal to the type of `d_in`, and the type of the evaluation of the $\mathcal{Y}(\cdot)$ function is equal to the type of `d_out`.

We refer to c as the *stability parameter*. In turn, the value c should not be mistaken with the stability relation, which is a relation property between `d_in`, `d_out`. For example, if the relation between `d_in`, `d_out` is specified as `d_in` $\leq$ `d_out`, then Relation(`d_in`, `d_out`) returns `true` if and only if `d_in` $\leq$ `d_out`. Moreover, the end user can empirically find the stability value c (i.e., the tightest bound) by querying multiple times with different (`d_in`, `d_out`) pairs in a binary search manner, and adaptively modifying them given the `true, false` answers returned by the stability relation. We will explore this relation further as well as the notion of d_{mid} when considering chaining.

We also remark that in the OpenDP library, the relation between `d_in`, `d_out` can (and will) be overestimated, and can only accept the worst-case or "global" privacy degradation over that transformation. In the case of measurements, the stability relation is replaced by the *privacy relation*, but we are not concerned with measurements in this document.

4 Algorithm Implementation

4.1 Code in Rust

The current OpenDP library contains the `make_clamp_vec` function implementing the clamping function. This is defined in lines 25-38 of the file `manipulation.rs` in the Git repository[7] (`https://github.com/opendp/opendp/blob/main/rust/opendp/src/trans/manipulation.rs#L25-L38`). (mike) This is pretty dated now. Would be good to get an updated version with the AbsoluteDistance clamping. I'm happy to explain how this works.

4.2 Function Definition

In clamping, the input data domain $\mathcal{X}$ is $\mathbb{R}$ and the output data domain $\mathcal{Y}$ is $[L, U]$, where $L, U \in \mathcal{X} = \mathbb{R}$. In the OpenDP library, domain $\mathbb{R}$ corresponds to `AllDomain<T>`, where `T: Float`.

[7] As of June 16, 2021.

```rust
pub fn make_clamp_vec<M, T>(lower: T, upper: T) -> Fallible<Transformation<VectorDomain<AllDomain<T>>, VectorDomain<IntervalDomain<T>>, M, M>>
    where M: Metric,
          T: 'static + Clone + PartialOrd,
          M::Distance: DistanceConstant + One {
    if lower > upper { return fallible!(MakeTransformation, "lower may not be greater than upper") }
    Ok(Transformation::new(
        VectorDomain::new_all(),
        VectorDomain::new(IntervalDomain::new(Bound::Included(lower.clone()), Bound::Included(upper.clone()))),
        Function::new(move |arg: &Vec<T>| arg.iter().map(|e| clamp(&lower, &upper, e)).collect()),
        M::default(),
        M::default(),
        // clamping has a c-stability of one, as well as a lipschitz constant of one
        StabilityRelation::new_from_constant(M::Distance::one())))
}
```

The function $\texttt{clamp}_{L,U} : \mathcal{X} \to \mathcal{Y}$ is defined as:

$$\texttt{clamp}_{L,U}(z) = \begin{cases} U & \text{if } z > U \\ z & \text{if } z \in [L, U] \\ L & \text{if } z < L \end{cases}. \tag{3}$$

Then, we can define the transformation $T : \text{MultiSets}(\mathcal{X}) \to \text{MultiSets}(\mathcal{Y})$, which we will show to be 1-stable, under the symmetric difference metric or under the Hamming distance metric can be defined by:

$$T(x) = \{\text{clamp}_{L,U}(z) : z \in x\}.$$

Note that while the clamp function might admit other metrics, the OpenDP library currently only implements symmetric difference and Hamming distance for database metrics, which is why we restrict ourselves to them in this document.

4.3 Pseudocode in Python

We present a simplified Python-like pseudocode of the Rust implementation below.

```python
class Transformation:
    input_domain
    output_domain
    function
    input_metric
    output_metric
    stability_relation

def MakeClamp(L: float, U: float, metric):
    if L > U: raise Exception('Invalid parameters')
    input_domain = vector(float)
    output_domain = IntervalDomain(L, U, float)
    input_metric = metric
    output_metric = metric
    assert isinstance(metric, SymmetricDifference or HammingDistance)
    stability_relation = lambda(d_in, d_out) : d_in <= d_out

    def function(data):
        def clamp(x): return max(min(x, U), L)
        return map(clamp, data)
```

```
    return Transformation(input_domain, output_domain, function,
   input_metric, output_metric, stability_relation)

# Example: suppose we are interested in the
# number of entries in a vector of integers or floats:
Clamping = MakeClamp(l, u, SymmetricDifference)

# We could also use a different metric:
Clamping = MakeClamp(l, u, HammingDistance)
```

In the Rust code, the stability parameter c (which in the case of clamping is equal to 1) gets wrapped up inside of the stability relation property, and the end user could test it empirically.

(silvia) We could generalize the input domain above from float to a any general (unspecified) type that admits total ordering, and then add the line `assert_has_trait(L, TotalOrdering` to the pseudocode, and similarly for U).

Conditions as specified in the pseudocode

- Input domain: vectors of elements of an arbitrary data domain which implement PartialOrd (= partial ordering).[8] That is, such data domain must admit a binary relation $\leq$ such that for all a, b, c in this data domain, it satisfies:

  1. Reflexivity ($a \leq a$),
  2. Antisymmetry (if $a \leq b$ and $b \leq a$ then $a = b$,
  3. Transitivity (if $a \leq b$ and $b \leq c$ then $a \leq c$).

  In OpenDP, such data domain consists primarily of ints and floats, which is why we specify floats above (more concretely, we allow f32/64).

- Output domain: vectors of elements of the input domain specified above but with the additional restriction of being contained within the interval $[L, U]$, where L, U are elements of $\mathcal{X}$.

- Function: returns vector mapping where each vector element v equals `clamp(v)` as specified in Equation (3).

- Input metric: symmetric difference or Hamming distance.

  - Symmetric difference: For any two datasets v, u, we say that v, u are d-close in symmetric difference if $d_{Sym}(u, v) = |\text{MultiSets}(v)\Delta\text{MultiSets}(u)| \leq d$. The symmetric difference between $\text{MultiSets}(u), \text{MultiSets}(v)$ is the set of elements, or rows, that appear in either $\text{MultiSets}(u)$ or $\text{MultiSets}(v)$ but *not* in their intersection.
  - Hamming distance: For any two vectors v, u, we say that v, u are d-close in hamming distance if $d_{Ham}(v, u) \leq d$. The Hamming distance between two vectors v, u is the number of places where u and v differ. Thus the Hamming distance between two vectors is the number of bits we must change to transform

[8] In the future, OpenDP should be implementing Ord (ordering) and not PartialOrd, but we work with the current implementation. The difference is that PartialOrd does not guarantee that every pair of elements in the data domain is comparable.

one vector into the other. The desired Hamming notion for the OpenDP library has not yet been decided, and so for now we do not include it in the proof.

- Output metric: same as input metric (which have to be consistent).
- Stability relation: for the two 32-integer bits d_{in}, d_{out}, we return `True` if $d_{in} \leq d_{out}$. In particular, clamping has stability parameter $c = 1$.

More formally, the stability relation can be stated as:[9]

$$R((d_{\mathcal{X}}, \texttt{d\_in}), (d_{\mathcal{Y}}, \texttt{d\_out})) = \begin{cases} \texttt{True} & \text{if } d_{\mathcal{X}} = d_{\mathcal{Y}} = d_{Sym}\ \texttt{d\_in}, \texttt{d\_out} \in \mathbb{Z}, \texttt{d\_out} \geq \texttt{d\_in} \\ \texttt{True} & \text{if } d_{\mathcal{X}} = d_{\mathcal{Y}} = d_{Ham},\ \texttt{d\_in}, \texttt{d\_out} \in \mathbb{Z}, \texttt{d\_out} \geq \texttt{d\_in} \\ \texttt{False} & \text{otherwise} \end{cases}. \tag{4}$$

Importantly, such relations are sound but *not necessarily complete*. A transformation is considered *valid* if its stability relation is sound.

5 Proof

5.1 Symmetric Difference

Theorem 2 (Appropriate output domain)**.** *For every vector v in the input domain, $\texttt{clamp}(v)$ is in the output domain. That is, $\texttt{clamp}_{L,U}(v) \in \texttt{bounded\_float}(L, U)$.*

Proof. This follows directly from the definition of the `clamp` function in Equation (3) and line 19 in the pseudocode of Section 1.2. (silvia) Detail more? □

The necessary condition that $L \leq U$ is already checked in the pseudocode, hence correctness is guaranteed.

Next we show that the stability relation as defined in Equation (1) yields a valid transformation.

Theorem 3 (Stability guarantee)**.** *For every setting of the input parameters to `MakeClamp`, the transformation produced has the following property: For every v, w in the input domain and for every pair (d_{in}, d_{out}) (appropriately quantified), if v, w are d_{in}-close and* Relation(*`d_in`*, *`d_out`*) = *`True`*, *then $\texttt{clamp}_{L,U}(v)$, $\texttt{clamp}_{L,U}(w)$ are d_{out}-close.*

Proof. Since by assumption Relation(`d_in`, `d_out`) = `True`, by the clamping stability relation (Equation (1)) we have that `d_in` $\leq$ `d_out`. Moreover, v, w are assumed to be `d_in`-close. By the definition of the symmetric difference metric, this is equivalent to stating that $d_{Sym}(v, w) = |\text{MultiSet}(v) \Delta \text{MultiSet}(w)| \leq d_{in}$.

For any data domain $\mathcal{X}$, a dataset $x \in \text{MultiSets}(\mathcal{X})$ can be specified by its histogram $h_x : \mathcal{X} \to \mathbb{N}$, where $h_x(z)$ denotes the number of occurrences of z in $\mathcal{X}$.[10] (silvia) I

[9] See page 14 of *A Programming Framework for OpenDP* for more examples. Moreover, in the actual Rust implementation the metrics $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ are *not* inputs to the relation, and they are instead fixed as attributes of the transformation.

[10] *A Programming Framework for OpenDP*, footnote 1 in page 3. Note that there is a bijection between multisets and histograms, which is why the proof can be carried out with either notion.

do not fully agree with the notation used in the Programming Framework (PF) for the histograms. Is it not the number of occurrences of z in $x \in \text{MultiSet}(\mathcal{X})$?

Then, it follows that

$$|\text{MultiSets}(v)\Delta\text{MultiSets}(w)| = \|h_v - h_w\|_1 = \sum_z |h_v(z) - h_w(z)|.$$

(silvia) The above notation for the symmetric difference is also not ideal because while we want to mean that we regard the input vector as a multiset (i.e., with no ordering), $\text{MultiSet}(v)$ would really mean a multiset of vector records, according to the PF. So we should agree on whether we write $|\text{MultiSet}(v)\Delta\text{MultiSet}(w)|$ or $|v\Delta w|$.

That is, the symmetric distance between v and w is equal to the ℓ_1 distance between h_v and h_w (histogram notation).[11] The second equality above applies the definition of ℓ_1 distance. By putting the above observation together with the stability relation, we obtain that

$$\sum_z |h_v(z) - h_w(z)| \leq d_{in} \leq d_{out}.$$

We now consider $\text{MultiSet}(\texttt{clamp}_{L,U}(v))$ and $\text{MultiSet}(\texttt{clamp}_{L,U}(w))$. For each element $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$, if $z \in \text{MultiSet}(v)\Delta\text{MultiSet}(w)$, we will assume wlog that $z \in \text{MultiSet}(v) \setminus \text{MultiSet}(w)$. We consider the following cases:

1. $z > U$ or $z < L$: then, in the former case, $\texttt{clamp}_{L,U}(z) = U$. First consider the case when $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$ with the same multiplicity in both multisets. Then, $|h_{\texttt{clamp}_{L,U}(v)}(z) - h_{\texttt{clamp}_{L,U}(w)}(z)| = 0$ because we have both $h_{\texttt{clamp}_{L,U}(v)}(z) = 0$ and $h_{\texttt{clamp}_{L,U}(w)}(z) = 0$. Thus the sum

$$\sum_z |h_{\texttt{clamp}_{L,U}(v)}(z) - h_{\texttt{clamp}_{L,U}(w)}(z)|$$

remains invariant, because the quantity $|h_v(z) - h_w(z)|$ is added to $|h_{\texttt{clamp}_{L,U}(v)}(U) - h_{\texttt{clamp}_{L,U}(w)}(U)|$, given that $\texttt{clamp}_{L,U(v)}(z) = \texttt{clamp}_{L,U(w)}(z) = U$.

(grace) Good explanation, especially w word invariant.

Suppose z has multiplicity $k_v \geq 0$ in $\text{MultiSet}(v)$ and multiplicity $k_w \geq 0$ in $\text{MultiSet}(w)$, where $k_v \neq k_w$. Then after considering z, the value $h_{\texttt{clamp}_{L,U}(v)}(U)$ becomes $h_{\texttt{clamp}_{L,U}(v)}(U) + k_v$, and $h_{\texttt{clamp}_{L,U}(w)}(U)$ becomes $h_{\texttt{clamp}_{L,U}(v)}(U) + k_w$. Hence the quantity $|h_{\texttt{clamp}_{L,U}(v)}(U) - h_{\texttt{clamp}_{L,U}(w)}(U)|$ increases by *at most* $|h_v(z) - h_w(z)|$, since, by the triangle inequality,

$$\begin{aligned}|(h_{\texttt{clamp}_{L,U}(v)}(U) + k_v) - (h_{\texttt{clamp}_{L,U}(w)}(U) + k_w)| \leq \\ \leq |h_{\texttt{clamp}_{L,U}(v)}(U) - h_{\texttt{clamp}_{L,U}(w)}(U)| + |k_v - k_w| = \\ = |h_{\texttt{clamp}_{L,U}(v)}(U) - h_{\texttt{clamp}_{L,U}(w)}(U)| + |h_v(z) - h_w(z)|.\end{aligned}$$

The same argument applies whenever $z < L$.

(silvia) The first subcase discussed here, i.e., when $k_v = k_w$, is also proven by the triangle inequality expression above, but it seemed clean to separate the case where the total sum remains invariant.

[11] This is because the symmetric difference between two databases X, X' is the set of elements, or rows, that appear in either X or X' but *not* in their intersection.

2. $z \in (L, U)$: then, $\texttt{clamp}_{L,U}(z) = z$. Since $h_v(z) = h_{\texttt{clamp}_{L,U}(v)}(z)$ and $h_v(w) = h_{\texttt{clamp}_{L,U}(w)}(z)$, it follows that $|h_v(z) - h_w(z)| = |h_{\texttt{clamp}_{L,U}(v)}(z) - h_{\texttt{clamp}_{L,U}(w)}(z)|$. Hence the histogram count, i.e., the quantity

$$\sum_z |h_{\texttt{clamp}_{L,U}(v)}(z) - h_{\texttt{clamp}_{L,U}(w)}(z)|$$

remains invariant.

3. $z = U$ or $z = L$: then, in the former case, $\texttt{clamp}_{L,U}(z) = U$. If $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$ with the same multiplicity in both multisets, then the histogram count remains invariant under the addition of element z. Otherwise, if $z \in \text{MultiSet}(v) \setminus \text{MultiSet}(w)$, or if z is in their union but with different multiplicity, then element z can increase the quantity $|h_{\texttt{clamp}_{L,U}(v)}(U) - h_{\texttt{clamp}_{L,U}(w)}(U)|$ by at most $|h_v(z) - h_w(z)|$, following the same reasoning with the triangle inequality as in case 2.

   The same argument applies whenever $z = L$.

By aggregating the three cases above, we conclude that

$$\sum_z |h_{\texttt{clamp}(v)}(z) - h_{\texttt{clamp}(w)}(z)| \leq \sum_z |h_v(z) - h_w(z)|.$$

By the initial assumptions, we recall that $\texttt{d\_in} \leq \texttt{d\_out}$, and that v, w are $\texttt{d\_in}$-close. Then,

$$\sum_z |h_{\texttt{clamp}(v)}(z) - h_{\texttt{clamp}(w)}(z)| \leq \sum_z |h_v(z) - h_w(z)| \leq \texttt{d\_in} \leq \texttt{d\_out}.$$

Therefore,

$$|\text{MultiSet}(\texttt{clamp}_{L,U}(v)) \Delta \text{MultiSet}(\texttt{clamp}_{L,U}(w))| \leq \texttt{d\_out},$$

as we wanted to show.

□

(silvia) Maybe add domain of z below the sum (grace) Great, very rigorous

5.2 Hamming Distance

The same proof as for symmetric difference holds. However, we await to write it out formally until the precise notion of Hamming distance (i.e., whether ordering wants to be preserved in the multiset or not) is encoded in the library.

6 Past Resolved Confusions

6.1 The Flipping of d_in, d_out

That a transformation T is c-stable is a property of the function, not of the relation. The stability relation is a claim about the transformation, namely:

Definition 1. Given a transformation T, a stability relation R is *valid* for T with respect to "metrics"[12] $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ if and only if $\forall x, x'$ in the input domain $\mathcal{X}$ and $\forall \texttt{d\_in}, \texttt{d\_out}$ such that $\text{Relation}(\texttt{d\_in}, \texttt{d\_out}) = \texttt{True}$, if x, x' are `d_in`-close with respect to $d_{\mathcal{X}}$, then $T(x), T(x')$ are `d_out`-close with respect to $d_{\mathcal{Y}}$.

So the way to think about this is as follows: first, we imagine the transformation without the relation (i.e., the function and the corresponding domains and metrics). Then we find the stability parameter between $d_{\mathcal{X}}, d_{\mathcal{Y}}$; i.e., the parameter c such that $d_{\mathcal{Y}}(T(x), T(x')) \leq c \cdot d_{\mathcal{X}}(x, x')\ \forall x, x'$. Then, because the stability relation has to be sound (but not complete), we establish the stability relation to be $\text{Relation}(\texttt{d\_in}, \texttt{d\_out}) = \texttt{True}$ if $\texttt{d\_in} \leq c \cdot \texttt{d\_out}$. Note that the inequality sign flips. This is because the implication

$$\text{If } d_{\mathcal{X}}(x, x') \leq \texttt{d\_in}, \text{ then } d_{\mathcal{Y}}(T(x), T(x')) \leq \texttt{d\_out}$$

holds if (and not if and only if)

$$\texttt{d\_out} \geq c \cdot \texttt{d\_in},$$

given that, in this case,

$$d_{\mathcal{Y}}(T(x), T(x')) \leq c \cdot d_{\mathcal{X}}(x, x') \leq c \cdot \texttt{d\_in} \leq \texttt{d\_out}.$$

Only after this has been proven do we add the stability relation to the code.

6.2 Sets, Vectors, and Metrics

In the OpenDP Programming Framework, datasets are represented as a multiset of records. Therefore, by the definition of a set, there is no ordering. While such multisets are represented as domains of vectors in the OpenDP library, this does *not* imply that these vectors are adding any notion of order to the set. For Hamming distance, we need to decide whether we stick with the usual definition, or whether we allow permutations ("semi-Hamming"). In the latter, between the symmetric difference and "semi-Hamming", only the data domain would change, and the two metrics would yield distances always a factor of 2 apart.

6.3 Forward and Backward Maps

Mike explanation on the stability relation in Rust: the internals of StabilityRelation actually contain up to three closures: the relation itself, as well as an optional forward map, and an optional backward map. The maps translate a distance in one space to another. For StabilityRelations constructed from `new_from_constant`, the forward map is automatically constructed and is essentially $|\texttt{d\_in}| c * \texttt{d\_in}$.[13]

When the composed relation is checked, the integrity of the relation is upheld by the mandatory relation closure. The maps that may come bundled inside StabilityRelation are used to construct hinters: if a forward map exists, the hint is $|\texttt{d\_in}, \texttt{d\_out}|\ \text{forward\_map}(\texttt{d\_in})$, which of course simplifies to $d_{mid} = c * \texttt{d\_in}$ if chaining with a c-stable transformation. Not all forward maps are as simple as $c * \texttt{d\_in}$.

[12] In fact, $d_{\mathcal{X}}$ are not required to be actual metrics (in the usual mathematical definition). Nonetheless, certain metric properties such as the triangle inequality might be necessary for certain later DP properties; e.g., group privacy.

[13] Notation: when he writes $|\texttt{d\_in}|\ c * \texttt{d\_in}$ he means $\text{forward\_map}(\texttt{d\_in}) = c * \texttt{d\_in}$.

So then you have `StabilityRelation::new_from_constant(1u32))`, which makes a relation $|\texttt{d\_in}, \texttt{d\_out}|\, \texttt{d\_out} \geq 1 * \texttt{d\_in}$, that also contains a forward map $|\texttt{d\_in}|\, 1 * \texttt{d\_in}$, and a backward map $|\texttt{d\_out}|\, \texttt{d\_out}/1$. This behaves as any 1-stable transformation would: when tight, the transformation does not increase the distance between datasets.

The forward map translates a distance in the input space to a distance in the output space. So that closure takes one argument, a distance in the input space. Then it translates it to a distance in the output space by multiplying `d_in` by c. When chaining two relations, you can use the forward map on the first relation to get d_{mid}. If the first relation does not have the optional forward map, check if the second relation has a backward map, and if so use it to construct a hinted d_{mid} from `d_out`. It is notation for a closure; the stuff in the pipes are the arguments, and the stuff after the pipes is the function body.

7 Further Comments/Questions – (Resolved after 16/6 meeting)

- Unification of proofs: agreeing on the class attributes in the pseudocode and in the proof elements. Should the stability parameter be part of the pseudocode? Minor: float vs f32/f64 (see my pseudocode above). And include Example with actual metrics in the pseudocode?
- **Important.** Proof elements: (perhaps correct output domain), the stability relation is sound, and `d_in` plus stability relation being true implies `d_out` (DP guarantee).
  - Assuming vs proving the `d_in`, `d_out` stability relation (Excel column). Different relations are possible (because they are not required to be tight; completeness is not necessary), but for the transformation to be valid they have to be sound.
  - Rephrasing: what we have all done is: assume *input* $\leq$ `d_in` and `d_in` $\leq$ `d_out` (so assume relation). Then, we want to show that *output* $\leq$ `d_out`. To do so, our three proofs do: we show that *output* $\leq$ *input*, because then *output* $\leq$ *input* $\leq$ `d_in` $\leq$ `d_out`. But showing *output* $\leq$ *input* is precisely what justifies using the stability relation `d_in` $\leq$ `d_out`. Recursive argument? (Should not assume Excel columns, they should be proven)
  - Formal definition of `d_in`, `d_out`.
  - What the PF says on page 12 is not coherent with the above. Page 14 is though.
  - The flipping inequality of the `d_in`, `d_out`, and relation to $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$, respectively.
  - Relationship to ϵ and sensitivity.
  - Base Laplace proof *fixes* `d_in`, `d_out`.
- **Relevant.** I think we are all making mistakes with vectors vs datasets vs MultiSets when discussing d_{Sym}, d_{Ham}. Because Rust takes in a vector when we are talking about a dataset. Symmetric difference is for datasets; Hamming is for vectors. And for histogram: where does z belong to? (E.g., clamping: there is a difference between it being 0 and z just not being part of the considered domain). In this line of question, in the Excel sheet some transformations have both symmetric and Hamming, and some only have symmetric. Why? And those that have both symmetric

and Hamming, how can they have the same domain if set $\neq$ vector? Lastly, count is using a reasoning based on rows, so again important to be clear on what is a vector and what is a dataset (vectors do not have rows). I think there are also some confusions in the proofs with x vs $\mathcal{X}$.

- Independence of the stability relation from the end user input metric. This is how we all have it now in the pseudocode, but this is not coherent with PF page 14.
- Using histogram notation (Grace has the same question). Histogram notation for other metrics outside of symmetric difference? (E.g., Hamming).
- Connor and Grace show that their input/output relation holds for every element z, whereas I need to consider the whole sum (and would not be correct element-wise).
- MultiSet notation: writing v, w vs writing $\text{MultiSet}(v), \text{MultiSet}(w)$. And when we discuss the domain of $function(v)$.
- Stability relation in the Rust code (Slack discussions).
- Triangle inequality.
- Hint and sensitivity (Excel sheet).
- Repeating the proof for Hamming distance: is what we do enough or should we do it in full and self-contained?
- (Small) Include code snippet? (for unification purposes)
- Constructors.