

fn make_select_private_candidate

Michael Shoemate

This proof resides in “**contrib**” because it has not completed the vetting process.

Proves soundness of `make_select_private_candidate` in `mod.rs` at commit `f5bb719` (outdated¹).
`make_select_private_candidate` returns a `Measurement` that returns a release from `measurement` whose score is above `threshold`, or may fail and return nothing.

1 Hoare Triple

Precondition

Compiler-verified

- Generic DI (input domain) is a type with trait `Domain`.
- Generic MI (input metric) is a type with trait `Metric`.
- `MetricSpace` is implemented for (DI, MI). Therefore MI is a valid metric on DI.
- Argument `measurement` is a measurement whose output metric is `MaxDivergence`, and releases a tuple (f64, T0), where T0 is some arbitrary type.
- Argument `stop_probability` is of type f64.
- Argument `threshold` is of type f64.

User-verified

None

Pseudocode

```
1 def make_select_private_candidate(  
2     measurement: Measurement,  
3     stop_probability: float,  
4     threshold: float,  
5 ) -> Measurement:  
6     if not 0 <= stop_probability < 1: #  
7         raise "stop_probability must be in [0, 1]"  
8  
9     if not threshold.is_finite():  
10         raise "threshold must be finite"  
11  
12     scale = None  
13     if stop_probability > 0.0:
```

¹See new changes with `git diff f5bb719..0e34305 rust/src/combinators/select_private_candidate/mod.rs`

```

14     ln_cp = (1.0).neg_inf_sub(stop_probability).inf_ln()
15     scale = ln_cp.recip().neg().into_rational() #
16
17     def function(arg):
18         remaining_iterations = None
19         if scale is not None:
20             remaining_iterations = UBig.ONE + sample_geometric_exp_fast(scale) #
21
22         while True:
23             score, output = measurement(arg)
24
25             if score >= threshold:
26                 return score, output
27
28             if remaining_iterations is not None:
29                 remaining_iterations -= UBig.ONE
30                 if remaining_iterations == UBig.ZERO:
31                     return None
32
33     return Measurement(
34         input_domain=measurement.input_domain,
35         input_metric=measurement.input_metric,
36         output_measure=measurement.output_measure,
37         function=function,
38         privacy_map=lambda d_in: measurement.map(d_in).inf_mul(2),
39     )

```

Postcondition

Theorem 1.1. For every setting of the input parameters (`measurement`, `stop_probability`, `threshold`, `DI`, `MI`, `T0`) to `make_select_private_candidate` such that the given preconditions hold, `make_select_private_candidate` raises an exception (at compile time or run time) or returns a valid measurement. A valid measurement has the following property:

1. (Privacy guarantee). For every pair of elements x, x' in `input_domain` and for every pair (d_{in}, d_{out}) , where d_{in} has the associated type for `input_metric` and d_{out} has the associated type for `output_measure`, if x, x' are d_{in} -close under `input_metric`, `privacy_map(d_in)` does not raise an exception, and `privacy_map(d_in) ≤ d_out`, then `function(x), function(x')` are d_{out} -close under `output_measure`.

2 Proofs

This section shows that the pseudocode implements Theorem 3.1 in [1], where `stop_probability` is γ and `threshold` is τ .

Theorem 2.1. The function pseudocode is equivalent to Algorithm 1 in [1].

Proof. `scale` is $-1/\ln(1 - \gamma)$, with arithmetic conservatively rounded down. The rounding down corresponds to higher effective stop probability. A higher effective stop probability results in a lower effective privacy usage ϵ_0 . This means the advertised privacy loss is a conservative overestimate.

Since γ is restricted to $[0, 1)$ by 6, `scale` is non-negative. Therefore in 20, the precondition of `sample_geometric_exp_fast` is satisfied. In Algorithm 1, the mechanism is invoked once before potentially terminating, so one is added to the sample. `remaining_iterations`, a sample from the shifted geometric distribution, is then equivalent to a count of the number of coin flips made until sampling one heads, as is used in Algorithm 1.

We then only run as many iterations as has been sampled, as in Algorithm 1. In the case where T is infinity and γ is zero, then the algorithm only terminates when score exceeds threshold.

If the measurement releases a score of NaN, then this is effectively treated as a score below negative infinity, as NaN will never compare greater than the threshold.

Otherwise, the rest of the algorithm is evidently equivalent. \square

Theorem 2.2. `make_select_private_candidate` is consistent with Theorem 3.1 of [1].

Proof. Since there is no limit on iterations, ϵ_0 is zero. By 2.1, the function is equivalent to Algorithm 1, so we can claim parts a-e of Theorem 3.1. Since `measurement` is a valid measurement, then we know that it satisfies ϵ_1 -DP. The privacy map returns $2\epsilon_1 + \epsilon_0$ with arithmetic rounded conservatively up, which is consistent with part b in Theorem 3.1. \square

(Privacy guarantee.) By 2.2, assuming correctness of Theorem 3.1 part b in [1], then for every pair of elements $x, x' \in \text{input_domain}$ and every $d_{MI}(x, x') \leq \text{d_in}$ with $\text{d_in} \geq 0$, if x, x' are d_in -close then `function(x), function(x')` are `privacy_map(d_in)`-close under `output_measure` (the Max-Divergence).

References

- [1] Jingcheng Liu and Kunal Talwar. Private selection from private candidates, 2018.