Wire-Cell Toolkit Randomness

Brett Viren

January 17, 2025

Topics

Wire-Cell Toolkit randomness

- The `IRandom` interface class
- The `Random` implementation class
- Random consumers
- Reseeding issues
- Proposed adaption to *art*'s random
- Multi-threading issues
- Class of bugs leading to non-deterministic results

The IRandom Interface Class

```
class IRandom : public IComponent<IRandom> {
    using int_func = std::function<int()>;

    /// Sample a binomial distribution
    virtual int binomial(int max, double prob) = 0;
    virtual int_func make_binomial(int max, double prob);
    // ...etc for other distributions
};
```

- Direct `<distribution>()` methods return a random sample.
- The `make_<distribution>()` methods allow returning a closure over some distribution object to avoid the overhead of calling its constructor for each sample.

The Random implementation class

```
class Random : public IRandom, public IConfigurable {
    // IConfigurable implementation...
    // IRandom implementation....
};
```

Configuration parameters accepted:

`seeds` one or more integers, passed to C++ random engine.

`generator` name of a generator (C++'s default or `"twister"` uses `std::mt19937`).

Each `make_<distribution>()` method returns a callable closure over an instance of C++'s type: `std::<distribution>_distribution`.

Getting and using an IRandom

Like all WCT interfaces, one gets an instance by **type/name**.

- The type/name is almost always provided via configuration.
- The IRandom type/name is usually provided by parameter "rng".

Drifter example with a default type/name of "Random":

```
void Gen::Drifter::configure(const WireCell::Configuration& cfg)
{
    m_rng_tn = get(cfg, "rng", m_rng_tn);
    m_rng = Factory::find_tn<IRandom>(m_rng_tn);
    // ...
}
```

Drifter example usage of IRandom:

```
dQ = sign * m_rng->binomial((int) std::abs(Qi), absorbprob);
```

Issues for adding re-seeding to WCT

Re-seeding is very application-specific

Re-seeding strategies are very application specific. WCT does not even have the idea of a “event” so any “per-event reseeding” has no meaning.

No “re-configure” phase in WCT’s life-cycle

Re-seeding implies a novel “re-configure” phase would be needed.

- Can add one, in principle, but opens a can-of-worms.

A new `IRandom` implementation for *art* in LArSoft's `larwirecell`

```
// Would be placed, eg, in larwirecell/Components/Random.{h,cxx}
namespace wcls {
  class Random : public IArtEventVisitor, public IRandom {
    // ... IArtEventVisitor methods ...
    virtual void visit(art::Event& event);
    // ... IRandom methods ...
  };
}
```

An example of a "two-faced" component with faces:

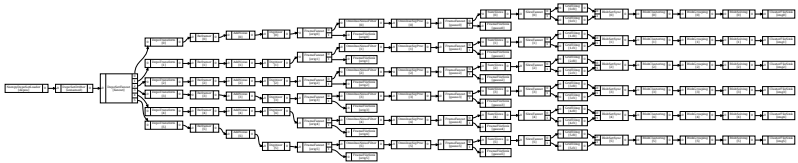
WCT implements `IRandom`

- Called by WCT-side random consumers.
- Hold/manage/use an *art* pRNG in the standard *art* way.

art implements `IArtEventVisitor`

- Implement `visit()` to do per-event "*art* things".
  - *eg* re-seed the held *art* pRNG based "event number".

Multi-threading issues - CPU



sim+sp+img for 6-APA PDSP

`IRandom` users: `Drifter` (1), `DepoTransorm` (6) and `AddNoise` (6)

- Sharing a common `IRandom` + `TbbFlow` = irreproducible results due to MT.
- **Expect** MT-reproducibility if we construct a unique `IRandom` for each consumer.
  - Assuming the underlying pRNG instances do not share state.
  - Needs explicit validation, so far we have not tried this.

We can invent a Jsonnet function to “bolt on” unique `IRandom`’s to existing graph.

Multi-threading issues - GPU

Our general understanding is that GPU will induce non-reproducibility at some level.
This impacts these Wire-Cell Toolkit features:

- DNNROI signal processing
  - Other DL algs in development
- FFT acceleration
- Signal Processing Next Generation
- Future simulation acceleration

Unacceptable sources of randomness - pointer order

A common idiom (not just a WCT issue):

```
std::map<Type*, Value> mymap = ...;
for (auto& [t,v] : mymap) {
    do_something_where_order_matters(t,v);
}
```

The order of the loop depends on **pointer values** which are not deterministic.

- Reproducibility, itself, is not so reproducible in this case.
  - *eg* stable ordering until OS, compiler or compiler flags change.

In Wire-Cell Toolkit, this idiom is considered a **bug that must be fixed**.

- Recently, found `Pgraph` had it! Fixed now.
- No promise that there are no more of these bugs lurking.

I am curious how large this problem is outside of Wire-Cell Toolkit!

Summary

All WCT consumers of randoms use `IRandom` interface

- Single implementation based on C++'s `<random>`.
- Provides for seeding but not re-seeding.

Can add a new `IRandom` implementation in `larwirecell`.

- Can use *art* pRNG internally.
- Can re-seed before and/or after each "event".

Configure unique `IRandom` for each consumer for MT-reproduciblity.