MABC and Clustering cleanups

Brett Viren

May 6, 2025

Relevant issues on github

wire-cell-toolkit

- [5/6] 412 MultiAlgBlobClustering (MABC) cleanup (“meta” issue)
  - ☒ 408 IDetectorVolumes is not a dumping ground
  - ☒ 379 MABC should configure its clustering funcs in configure() not operator().
  - ☒ 375 MABC hard-wires part of the input/output dataset paths
  - ☒ 413 Clustering facades are in the wrong namespace
  - ☒ 411 Interface between MABC and clustering functions should be based on actual Interfaces
  - ☐ 377 More flexible clustering function interface to handle variable number of kinds of Grouping
    - Some discussion on this one is likely useful.

wcp-porting-img

- ☐ 7 Ambigously named BlobSampler live/dead pairs in “pdhd” test.
  - Found along the way, blocking full cleanup of this config.

Summary of non-changes

No changes to algorithms and “identical” results expected.

Validation strategies

Started with these from `wcp-porting-img`

- `qlport/`
- `fgval/`
- `pdhd/`

Copied their Jsonnet (almost) as-is into:

`wire-cell-toolkit/clus/test/test-porting/{name}/*.jsonnet`

Built BATS test to run one or all of them:

```
$ bats [-f {name}] clus/test/test-porting.bats
```

Collect logs and compiled JSON and do before/after `diff`.

False-positive differences led to major time sinks

- **Easy: cosmetic differences**
  - Log timestamps, component instance names, TICK/MEM measurements.
  - Requires judicious `grep` and `sed` to normalize cosmetic differences.
- **Medium: nondeterministic PGrapher**
  - Cherry-picked fix from `master` to `apply-pointcloud`.
- **Hard: nondeterministic ClusteringConnect1**
  - The number of clusters found can change by a small amount.
    - ★ No visible change seen when viewign results in **Bee**.
    - ★ Simply rerunning the same code causes no change.
    - ★ Unrelated code modification may **sometimes** lead to a changed result.
  - Xin thinks this is floating point precision (which seems reasonable to me).
    - ★ Differences thought to be related to 64bit C++ vs 80bit CPU FP representations.
  - I don't have a fix nor even a small test to reproduce it.

Split `IDetectorVolumes` and `Clus::IPCTransformSet`

New interface: `IPCTransformSet`

- Defined in `clus/` as I feel it is not general enough for `iface/`.
- Provides `forward()`, `backward()` and `filter()` methods.
- Single implementation: `PCTransformSet` with a single "T0Correction" transform.
- `Grouping` is still given an `IDetectorVolumes` but for its internal use only.
- Any component that needs `IDetectorVolumes` or `IPCTransformSet` gets type/name and does a `Factory::find_tn()`, as usual.

Summary of changes - Clustering “functions”

- The “clustering classes” are all now proper `IClusteringMethod` components
  - WCT (not MABC does creation / configuration.
  - MABC now only takes list of the usual component `type:name`.
- `IClusteringMethod` implementations provide:

```
void clustering(Grouping& a, Grouping& b, cluster_set_t& c) const;
```

- The `clustering_*.cxx` houses all code for each component.
  - Shared functions still in `ClusteringFuncs.h`.
- MABC now takes a list of `IClusteringMethod` type names

Configuration change for clustering:

Then

MABC acted creator, configurator and manager of clustering functions.

- MABC had giant, hard-wired if/then/else factory pattern.
- Changes in many places needed for new clustering “functions”.
- Clustering config was done (and redone) at execute time.

Now

Create and configure an `IClusteringMethod` as you normally do any component.

- MABC gets usual `"type:name"` list and uses `Factory::find_tn()` to get interface.
- Clustering configuration is done once at configuration time.

Exploit clustering structural patterns: C++

Many `IClusteringMethod`'s need one or more of three configuration items.
To normalize this, they inherit from one or more `Need*` mixin classes:

- `NeedDV` to get `IDetectorVolumes` as `m_dv`
- `NeedPCTS` to get `IPCTransformSet` as `m_pcts`
- `NeedScope` to get a `Scope` as `m_scope`.
  - default scope is `"3d"` PC with `["x", "y", "z"]` coords.

Exploit clustering structural patterns: configuration

New file: `cfg/pgrapher/common/clus.jsonnet`.

```
local clus = import "pgrapher/common/clus.jsonnet";
local cm = clus.cluster_methods("all", dv, pcts, pc_name, coords);
local cm_objs = [                // pipeline of clustering "functions"
  cm.live_dead(),
  cm.regular("-one"),             // default values. "all-one"
  // Need unique name for 2nd instance, "all-two"
  cm.regular("-two", length_cut=30*wc.cm, flag_enable_extend=true),
];
local mabc = g.pnode({           // pass to MABC
   type: "MultiAlgBlobClustering",
   data: {
       clustering_methods = wc.tns(cm_objs);
       ...
   },
}, nin=1, nout=1, uses=cm_objs + [...]);
```

Note the `"all"` prefix to help assure unique type/names.

Bug in `wcp-porting-img/pdhd/clus.jsonnet`?

Issue #7: Ambigously named BlobSampler live/dead pairs in “pdhd” test.

- Both “live” and “dead” `BlobSamplers` for a given apa/face have same name.

```
name: "%s-%d"%[anode.name, face],
```

- Only one will be returned by code calling `Factory::find_tn()`.
  - Depends on order of configuration array but first? last? dunno.
- Breaking the ambiguity led to the job crashing.
  - I was unable to figure out how to fix or even how to “properly break”.
    - ★ *eg* by defining only live or only dead versions.

This needs fixing before applying the final configuration polish to the “pdhd” config.

Open issue - facade clean ups

The `Cluster` facade has become enormous

- Xin has some useful categories of potential cleanup.
  - I think removing any "scope" held by `Cluster` would go a long way.
  - Lots of effort and frankly it's getting hard to care.
- There are some bad side effect patterns that we might attack.
  - Eg: `dijkstra_shortest_paths()` creates the graph it operates on!
  - Need to assure the graph is made before `dijkstra_shortest_paths()` is called.

Best steps going forward with future porting:

- "No" new facade methods - of course, there may be some needs.
- Instead, make new **free functions** taking eg `Cluster` as arg.

Open issue - point tree building

We still have PC trees built slightly differently in ≈ 3 places.

- `PointTreeBuilding`
- `UbooneClusterSource`
- `ClusteringRetile`

Open issue - revamp the `IClusteringMethod` API

Issue #377 More flexible clustering function interface to handle variable number of kinds of Grouping. Right now we have:

```
void clustering(Grouping& live, Grouping& dead, cluster_set_t& cs) const;
```

Main problem with the current API:

- This hard-wires exactly two `Groupings` while different numbers are needed.
  - Eg, a “shadow” grouping is neuter “live” nor “dead”.

We’ve discussed a “grouping map” API:

```
using GroupingMap = map<string, Grouping::pointer>;
GroupingMap clustering(GroupingMap grmap, cluster_set_t& cs) const;
```

Pros and cons of `GroupingMap`

pros

- gives the desired flexibility

cons

- clustering functions need to check for existence of needed maps.
- need some way to connect “needed input” and “produced output” grouping names.
  - effectively, it hides an implicit data-flow graph

desiderata

- What sort of patterns, vis-a-vis multiple Groupings, will future porting require?

Misc - new BATS helper script

```
./test/scripts/bats-debug -f 'pdhd' clus/test/test-porting.bats
```

Simply sets some env. vars. to help run the BATS test in “developer mode”

- Keep the output directory (`tmp/`)
- Turn on BATS logging to terminal at “debug” level.