# $2^{nd}$ round of `clus/` cleanups

## A focus on graphs

Brett Viren

May 30, 2025

# Overview

- Relevant Issues on GitHub.
- Introduction of new facade: `Ensemble`.
- Introduction of new concept: cluster flags.
- Transferring of cluster state.
- Cluster IDs
- Bee issues and wire-cell-python improvements
- Cluster cache issues
- New support for graphs and graph algorithms
- Factoring “clustering retile” to prep for patrec
- Comments on issues that are still open.

# Relevant issues on github

## wire-cell-toolkit

- [8/11] Cluster graph and other prep for future porting efforts. #414
  - ☒ Multi-grouping API for Cluster and MABC #416
  - ☒ Multi-graph storage on Cluster facade #419
  - ☒ Clustering Dijkstra's shortest path refactoring #417
  - ☒ Refactor Cluster::examine$_{\text{graph}}$() #418
  - ☒ Factor body of ClusteringRetile main loop into a stand-alone function #415
  - ☒ Normalize the enumeration of cluster IDs #421
  - ☒ Small blobs can have no "stepped" sample points. #425
  - ☒ WCT allows duplicate component instance type/names which violates the rules. #427
  - ☐ `Cluster` facade has grown state not related to caching and it needs control #420
  - ☐ Stand-alone PCA is needed for `clus` #423
  - ☐ Copy-pasted `Cluster` state transfer code needs to be replaced with `Cluster::from()` #424

## Tangential

- ☐ `HaiwangYu/wcp-porting-img` Ambigously named BlobSampler live/dead pairs in "pdhd" test.
- ☐ BlobSampler API has become broken which lays a trap for the user. #426

## Comments on open issues at end.

# `Ensemble` PC tree node facade and `IEnsembleVistor`

```
// old
void IClusteringMethod::clustering(Grouping& a, Grouping& b, cluster_set_t& c);

// new
void IEnsembleVisitor::visit(Facade::Ensemble& ensemble) const;
```

- New, generalized name and API.
  - Use for both "clustering" and "pattern recognition".
- The `cluster_set_t& c` replaced with "cluster flags" (described next).
- `Ensemble` is a **new facade** to a **new PC tree root node**
  - Children nodes carry `Grouping` facades.
  - Gives access to child `Grouping` by **name**.
- Visitors may create new `Grouping` (eg "shadow") without nuking others (eg "dead").

## MABC support changes

- `Ensemble` is merely a **temporary** used inside MABC.
- Load input based on `Grouping` name list (default: `["live", "dead"]`).
- Output saves a "loose" set of `Grouping` to `ITensorSet` to retain past behavior.
- In/out "datapaths" may be configured on per-grouping basis with reasonable defaults.

# Introduce new concept: “cluster flags”

## Currently just one usage to mark “the cluster is connected to a dead cluster”

```
// clustering_live_dead.cxx sets it initially.
cluster->set_flag(Flags::live_dead);
// merge_clusters() transfers it, via from() (more on this later).
fresh_cluster.from(*old_cluster);
// clustering_extend() tests it.
if (cluster->get_flag(Flags::live_dead)) { ... }
```

## Ways to use cluster flags

- Define a subset of clusters that hold a common flag.
  - Replace the special-purpose "`cluster_set_t`" with `Flags::live_dead`.
- Flags are actually `int` so can take value in addition to indicating a yes/no.

## Inventing new flags

New flags are defined by adding a `const string` variable in `namespace Flags` eg in `ClusteringFuncs.h`. Don’t use literal strings in code.

# New `Cluster::from(const Cluster&)` method

`Cluster` has gained "state" beyond mere cache (it is not really a "facade" any more). Worse, user should not care but now is forced to manage this state. At the least, we should minimize this management burden. Instead of copy-paste many lines, user must now call `from()` to transfer the extra state.

```
// Now, just one line:
fresh_cluster.from(*old_cluster);

// Defined as
void Cluster::from(const Cluster& other)
{
    this->default_scope_from(other);
    this->flags_from(other);
}
```

Do **not** extend this. See #424 for some todos.

# Cluster ident

- All the facades provide methods: `int ident()/set_ident(int)`
  - Do not make up special methods to do the same thing.
- Cluster ID numbering:
  - Past: clusters from merge/separate had ID of `-1`
  - Now, **merge**: new ID is same as first input.
  - Now, **separate**: new IDs all same as the single input.
- Cluster IDs can be forcibly (re)enumerated:
  - Past: **every** clustering function set IDs to "tree order" on exit
  - Now: `Grouping::enumerate(string order, start=1)` is available with `order`:

| | |
|---|---|
| none | no (re)enumeration, IDs are as made |
| "tree" | use order of children clusters |
| "size" | use order determined by `cluster_less` |

MABC option: `cluster_id_order` forward to `enumerate()`, default is "none".

**Caveat: idents have no meaning, only rely on them for debugging.**

# Extension to MABC's Bee dumper

I wanted a richer set of Bee dumps: per visitor result and per grouping.

## Extend `BeePointsConfig` and MABC's configuration options parsing

```
struct BeePointsConfig {
    // ...
    std::string grouping;
    std::string visitor;
};
```

- `grouping` Name the grouping to consider, defaults to `"live"`
- `visitor` Name the `IEnsembleVisitor` results to dump, defaults to post-loop results.

Prior behavior shoudld be unchanged:

- Special case handling of `"img"` Bee type is preserved.

## Possible future work

MABC Bee dumper deserves some factoring and rationalization / reintegration with the stand-alone Bee-dumper component.

# Slightly off topic: `wire-cell-python` and `wcpy bee`

## Simpler, unified CLI brings a global `--help` and one-stop shopping

```
$ wire-cell-<name> <command>  # maps to:
$ wcpy <name> <command>
```

## Can now use `uv` to build/install/run

```
$ uv tool install git+https://github.com/wirecell/wire-cell-python
$ wcpy
```

## Added a `wirecell.bee` module and "`wcpy bee`" command

```
$ wcpy bee summary bee.zip
$ wcpy bee diff bee1.zip bee2.zip
```

Limit scope with: `--depth {point,shape,cluster,grouping,ensemble}`

## Future `wirecell.bee` feature additions

- "Rand index", "chamfer" and/or other point-cloud distance metrics for approximate, "geometrical" diffs.
- Add a "`wcpy bee upload`" command replacing WCT's `bee-upload` shell script.
- Implement the actual Bee server code for easier "self-hosting".

# Cluster cache management

## The general facade caching mechanism was being totally ignored.

Perhaps because it was not sufficient!
It now allows for **fine-grained** cache filling / invalidating.

## Cached `Cluster` results are now **all** in a new `struct ClusterCache`

Fine-grained cache management pattern:

```
// Top of some Cluster method
auto& data = cache().data;  // non-const reference
if (data.empty()) {         // exact test depends on data type
    data = ...;             // fine-grain cache fill
}
// ...use or return the "data"
```

## Do not add more `Cluster` state and cache.

But if you must, add members to `ClusterCache` and not directly to `Cluster`.

# Graph type

## Graphs made/used by Dijkstra's shortest path and "examine graph" are now of type:

```
using Graph = boost::adjacency_list<
    boost::vecS,        // vertices
    boost::vecS,        // edges
    boost::undirectedS, // edge direction (none)
    boost::property<boost::vertex_index_t, size_t>,
    boost::property<boost::edge_weight_t, float>
    >;
```

- The `vecS` **vertex descriptor** assures it is also an **index**.
  - Our code was conflating the two. It worked only by accident.
- The `vecS` container assures order stability (`setS` explicitly not guaranteed)
- Now as before, **no vertex removal** after initial graph construction.

## Dijkstra's shortest paths distances: `int` → `double`.

Use of `int` caused many "distance ties", `double` makes ties less likely.

- Tie-breaking strategy depends on use of `vecS` vs `setS` for vertices.
- Using `vecS` may allow more stable tie-breaking, but this is not yet confirmed.

# Graph construction

## Basic construction always defines number of vertices

```
auto graph = std::make_unique<Weighted::Graph>(cluster.npoints());
```

## Several ways to form edge connections via one or more functions

```
connect_graph_<flavor>(cluster, ...);
```

Connection “flavors”: “basic”, “ctpc”, “closely”, “relaxed” (overclustering protection).

## Construction and connections combined in graph “maker” functions

```
auto graph = make_graph_<flavor>(cluster, ...);
```

## User code gets same graphs as before.

- Dijkstra’s shortest paths: “closely + basic” or “closely + ctpc + basic”.
- Examine graph: “closely + relaxed”.

# Cluster graph algorithm caching

## New `Cluster` method and sub-object for graph algorithms

```
// The "basic" (non-ctpc) graph, or get previously made GA.
const Graphs::Weighted::GraphAlgorithms&
graph_algorithms(const std::string& flavor = "basic") const;
// The "ctpc" and "relax" graph flavors need detector info
const Graphs::Weighted::GraphAlgorithms&
graph_algorithms(const std::string& flavor,
                 IDetectorVolumes::pointer dv,
                 IPCTransformSet::pointer pcts) const;
```

## `GraphAlgorithms` hold graph and are cached by `Cluster`

- Graphs to use come in **basic** and **ctpc** flavors (as before - based on `use_cptc` option).
- The `GraphAlgorithms` object encapsulates the graph and derivative results
  - Maps from **source point index** to a `ShortestPaths` object
  - `ShortestPaths` yields and caches map from **target point index** to the **shortest path**.
- `Cluster` caches up to two `GraphAlgorithms` objects based on `"basic"` and `"ctpc"` names.

## Future extensions

- More graph algorithms in `GraphAlgorithms`

# Comment on `GraphAlgorithms` and "bare" graphs

## Right now, `Cluster` only has `GraphAlgorithms`

- GA's **do not expose their graphs**.
- User can not "get" nor modify the underlying graph.
  - User can only chose what graph "flavor" to construct.

Primarily, this restriction is for cache and conceptual simplicity.

## Future usage may require more intimate graph access (eg Steiner).

- We will figure out what best to do once construction/mutation patterns are understood.

# Comment on graph serialization - a non feature (for now?)

## For now, we have none

- Graphs are purely cache on `Cluster` and not saved to PC tree.
- Thanks Haiwang for pointing out the wisdom.

## Clear idea for providing it, if needed.

I have code to perform serialization with `PointCloud::Dataset`.

- It is laborious written for each C++ type of `boost::graph`.
- Also I have code for a **generic graph I/O** system
  - PC datasets, files (XML, JSON), and more.
  - Requires WCT gain a dependency on Boost's serialization library.

# Factor `ClusteringRetile` → `RetileCluster`

## Why do this?

- `ClusteringRetile` operates at `Grouping` level.
- Pat. rec. will need to "retile" individual clusters to make a temporary.

## The body of `ClusteringRetile`'s main loop moved into a `RetileCluster`

```
// New Interface:
std::unique_ptr<node_t> IPCTreeMutate::mutate(node_t& node) const;
```

- Uses PC tree **node**, not **facade** due to ownership issues.
- Named "mutate" as `separate()/merge()` requires changes (though they be temporary).
  - A const `IPCTreeTransform::transform(const node_t&)` is also now available.
- Return may be `nullptr`.

## Leaves `ClusteringRetile` very short

Handles creation of a "shadow" grouping in the `Ensemble` node.

- Loops over clusters, calls `IPCTreeMutate::mutate()`.

# Misc API/code cleanups

- `Cluster::get_pca()` only, method. It returns `struct PCA` with `.center/.axis/.value`.
- Try to make all class attributes start with `m_`.
- Try to make `this->` prefix on internal `Cluster` method calls to help mark targets for future refactoring.
- Match `ClusteringFuncs{.h,cxx}` file names.

# Open issues - PCA

Stand-alone PCA is needed for `clus` #423

- We have a PCA calculator in `WireCell::Array` for points already in an `Eigen:Matrix`.
- Any “point type generic” PCA calculator may be more trouble than it is worth.

Since I am unsure what “best” to do, I put this on hold until we hit the need. It should be easy to provide then.

# Open issues - Cluster complications

- `Cluster` facade has grown state not related to caching and it needs control #420
- Copy-pasted `Cluster` state transfer code needs to be replaced with `Cluster::from()` #424

There's probably no end to these. I think main thing now is to go through and replace the multi-line transfer of Cluster state (default scope, etc) to the single call to `cluster->from(&other_cluster)` in all the clustering functions.
I can do this but for now I'm just tired of the cleanup.

# Open (closed?) issues - ambiguous names

- wcp-porting-img's Ambigously named BlobSampler live/dead pairs in "pdhd" test.

At this point, I suggest we close this Issue in favor and copy WCT's

```
clus/test/test-porting/pdhd/clus.jsonnet
```

Note, this uncovered

- Small blobs can have no "stepped" sample points. #425

Fixing that involves adding a `cm.pointed()` clustering function at the front of MABC's pipeline.

- This is done in WCT's `pdhd/clus.jsonnet`.
- It will change the results but not perceptively in Bee.

# Open issues - BlobSampler API

- BlobSampler API has become broken which lays a trap for the user. #426

Not sure what to do about this one as it really breaks things and I'm tired of doing cleanup for other's breakage. See issue for details.