Simulation Workshop Summary

Jack Carlton
University of Kentucky
July 11th, 2025

Pattern Finding Goals Coming in

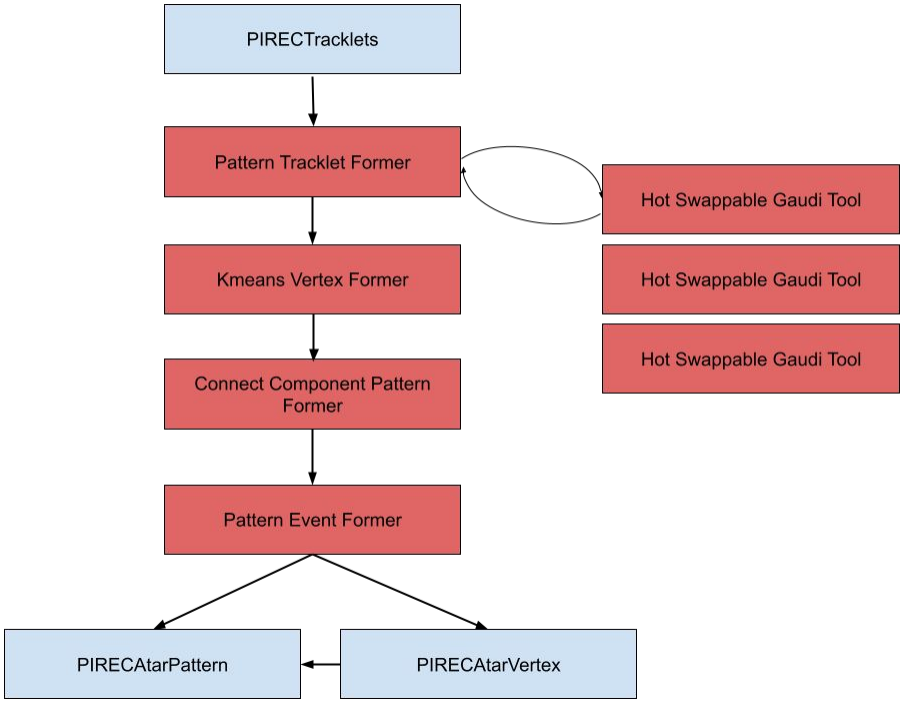
Proposed goals at the beginning:

- Characterize failure modes systematically
- Develop physics metrics to assess pattern finder's performance
- Refine vertex finding algorithm(s)
- Implement steps to handle latest tracklet finder developments, e.g. "strands"
- **Implement pattern finding framework in a more gaudi-like way**

This talk will focus on the structure of the new implementation

Pattern Finder Still Behaves Like a Pipeline

- Maintain the same “pipeline” behavior
- “Stages” are now Gaudi Tools
  - Have standard methods that run on Gaudi’s initialization and finalization sequence
  - Take Gaudi parameters for configuration
  - Allows for more natural configuration in .opts



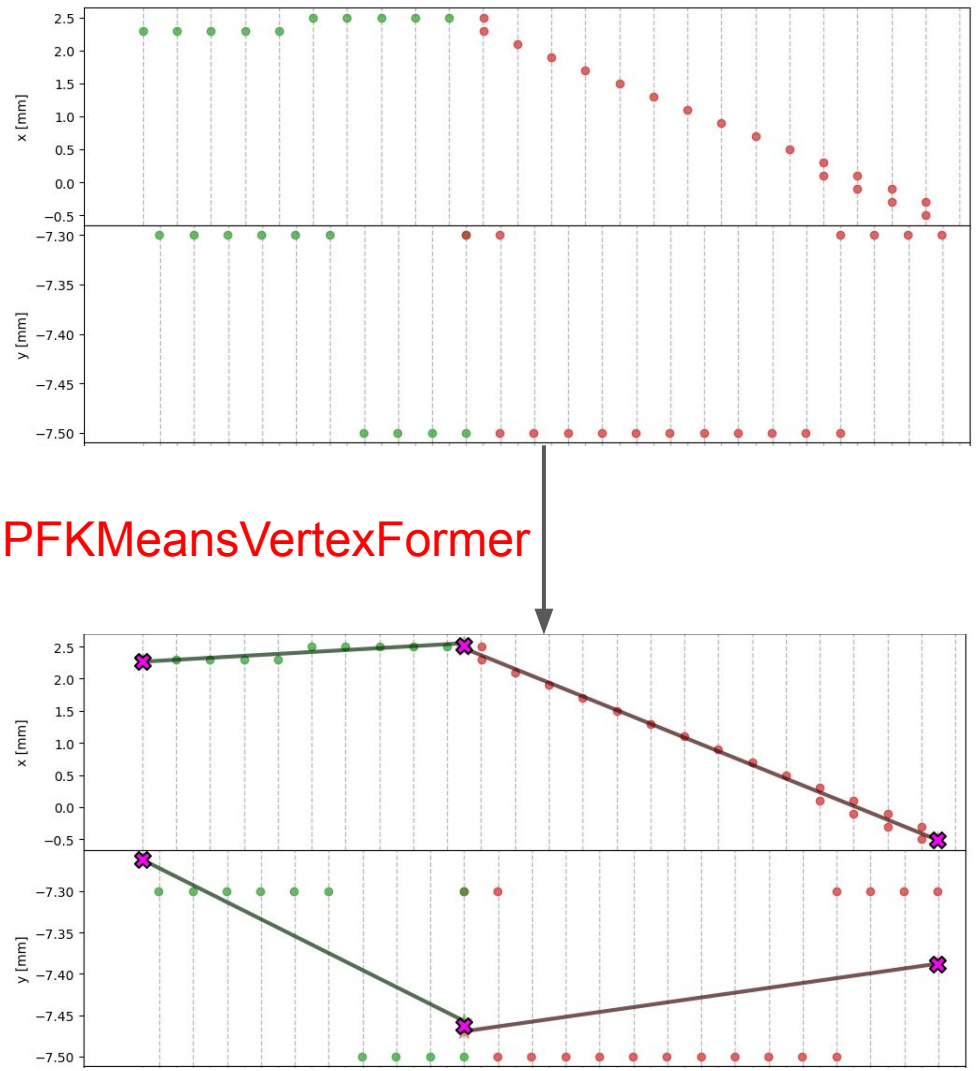
Gaudi Tools

- Gaudi's documentation is a bit dated but the "recipe" for making a tool is the same
- Created four tool interfaces
  - IPFTrackletFormer ≡ set<PIRECTracklet> → set<PFTracklet>
  - IPFVertexFormer ≡ set<PFTracklet> → set<PFVertex>
  - IPFPatternFormer ≡ set<PFVertex> → set<PFPattern>
  - IPFEventFormer ≡ set<PFPattern> → set<PFEvent>
- Implementations of each tool are created by deriving from the interface classes and implementing their virtual methods

```cpp
#pragma once
#include "GaudiKernel/IAlgTool.h"
#include "PFTracklet.h"
#include "PFVertex.h"
#include <unordered_set>
#include <memory>

// Interface for vertex formation tools
class IPFVertexFormer : virtual public IAlgTool {
public:
    DeclareInterfaceID(IPFVertexFormer, 1, 0);

    virtual ~IPFVertexFormer() = default;

    virtual std::unordered_set<std::shared_ptr<PFVertex>> form(
        const std::unordered_set<std::shared_ptr<PFTracklet>>& tracklets) const = 0;
};
```

Default Implementations

- PFDefaultTrackletFormer
  - Creates thin wrapper class PFTracklet around PIRECTracklets; in particular sets endpoints for KMeans to use
- PFKMeansVertexFormer
  - “Meat” of the pattern finder
  - Uses KMeans to create centroids that cluster endpoints in order to “match” tracklets
- PFDefaultPatternFormer
  - Uses DFS to find all connected vertices
- PFDefaultEventFormer
  - Create event object to access set of patterns



Control in Gaudi .opts file

- Tool implementations can changed in the options file
  - If not specified, the defaults discussed with be used
- Tools can also take their parameters in the options file
  - Again, if not specified the defaults will be used

```
//--------------------------------------------------------------
// PatternFinder tool configuration
//--------------------------------------------------------------
PatternFinder.TrackletFormer = "PFDefaultTrackletFormer";
PatternFinder.VertexFormer   = "PFKMeansVertexFormer";
PatternFinder.PatternFormer  = "PFDefaultPatternFormer";
PatternFinder.EventFormer    = "PFDefaultEventFormer";

//--------------------------------------------------------------
// Configure the tool properties
//--------------------------------------------------------------
// Then configure it via ToolSvc
PatternFinder.PFKMeansVertexFormer.Sigma = 1.5;
PatternFinder.PFKMeansVertexFormer.NIters = 20;
```

Next Steps

- Develop new algorithms to improve pattern finder performance
  - Can use Pattern Finding Playground python as a workspace to more rapidly develop and test algorithms performance
- Add python tested new algorithms in C++ Framework
  - Simple as making a new class, allows for seamless merging
- (If needed) Convert internal containers to available objects in the root tree
- (If needed) “Normalize” Pattern finder tools so they all derive from one interface
  - Would allow use of a “`ToolHandlerArray`” to define custom sequences in the pattern finder with an arbitrary number of “stages”
  - Requires inputless tools. Tools would grab data from Gaudi storage (similar to Josh’s proposal).

Auxiliary Slides

What is Each Container Class?

- All these classes are pattern finder “internal helper container” classes, not mean to be in the TTree
  - Can migrate these into TTree in the future
- PFTracklet
  - Wrapper around PIRECTracklet
- PFVertex
  - Wrapper around set of PFTracklets
- PFPattern
  - Wrapper around set of PFVertices
- PFEvent
  - Wrapper around set of PFPatterns

How to Add a New Implementation

1. Derive from interface
2. Write interface method implementation
3. Specify in Gaudi opts file the class name
    a. This name is grabbed from `DECLARE_COMPONENT`

- New adding new tool interfaces is a bit more involved
    - Must register the interface in PIAPatternFinder.hh using the ToolHandler

```cpp
#include "PFDefaultEventFormer.h"

DECLARE_COMPONENT(PFDefaultEventFormer)

PFDefaultEventFormer::PFDefaultEventFormer(const std::string& type, const std::string& name,
                                           const IInterface* parent)
    : base_class(type, name, parent) {
}

StatusCode PFDefaultEventFormer::initialize() {
    return StatusCode::SUCCESS;
}

StatusCode PFDefaultEventFormer::finalize() {
    return StatusCode::SUCCESS;
}

std::shared_ptr<PFEvent> PFDefaultEventFormer::form(
    const std::unordered_set<std::shared_ptr<PFPattern>>& patternSet) const {

    std::shared_ptr<PFEvent> event = std::make_shared<PFEvent>(0);
    event->setPatterns(patternSet);
    return event;
}
```

```
PatternFinder.EventFormer = "PFDefaultEventFormer";
```

Simple Example Implementation and How to specify in .opts

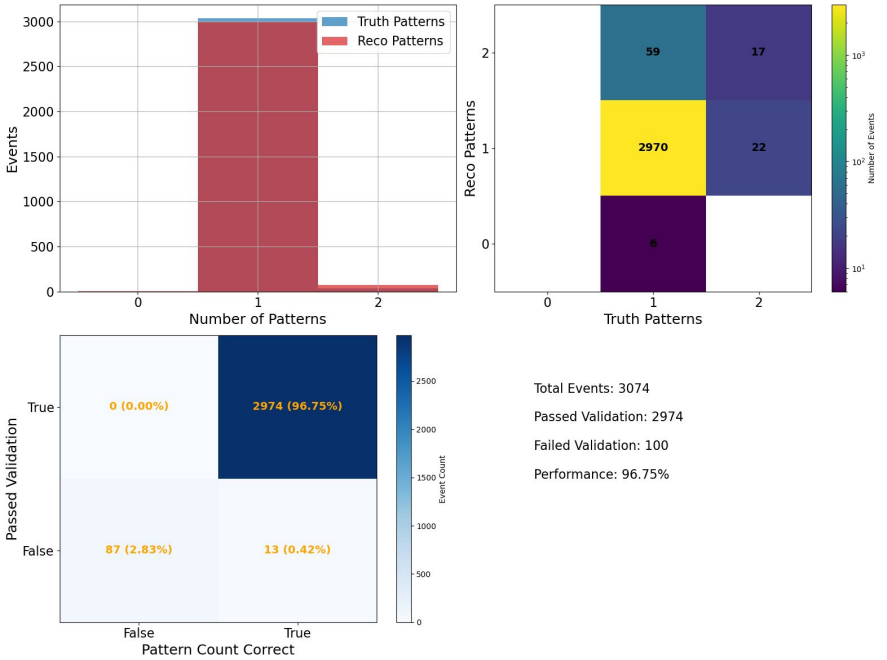
How to Create a Tool Type

- Use ToolHandle
  - Template around your custom interface
  - Give it a name an optional default implementation
- That algorithm with ownership can use and configure the tool
- For a hot swappable *list* of tools see ToolHandleArray

```
#ifndef PIAPatternFinder_H
#define PIAPatternFinder_H 1

#include <Gaudi/Algorithm.h>
#include <Gaudi/Property.h>
#include <GaudiKernel/ToolHandle.h>

// Tool interfaces
#include "IPFTrackletFormer.h"
#include "IPFVertexFormer.h"
#include "IPFPatternFormer.h"
#include "IPFEventFormer.h"

#include <memory>

// Forward declarations
class PIRECEvent;
class PIMCGeoHelper;

class PIAPatternFinder : public Gaudi::Algorithm {
public:
    PIAPatternFinder(const std::string& name, ISvcLocator* pSvcLocator);
    ~PIAPatternFinder() override;

    StatusCode initialize() override;
    StatusCode execute(const EventContext& ctx) const override;
    StatusCode finalize() override;

private:
    // --- Event and geometry
    PIRECEvent* fRecEvent;
    PIMCGeoHelper* fGeoHelper;

    // --- Tool handles
    ToolHandle<IPFTrackletFormer> m_trackletFormer{this, "TrackletFormer", "PFDefaultTrackletFormer"};
    ToolHandle<IPFVertexFormer> m_vertexFormer{this, "VertexFormer", "PFKMeansVertexFormer"};
    ToolHandle<IPFPatternFormer> m_patternFormer{this, "PatternFormer", "PFDefaultPatternFormer"};
    ToolHandle<IPFEventFormer> m_eventFormer{this, "EventFormer", "PFDefaultEventFormer"};
};

#endif // PIAPatternFinder_H
```

Pattern Finding Playground

- Recently got a new semi-major update
  - More general pipeline, can insert stages arbitrarily to build up an “event”
- Comes with event tools and plotting displays that take in events
- Carries diagnostic data at each stage for additional plotting



Example Performance Plot available in Playground