ACTS Vertexing and Deep Learning Vertex Finding

BASTIAN SCHLAG
CERN / JGU MAINZ
ACTS – A Common Tracking Software

ACTS Overview Talk by Xiaocong Ai (Recording Session 6)

• Detector-independent track- and vertex reconstruction toolkit

  ➢ based on ATLAS track reconstruction software: extremely well tested
  ➢ but: ~ 15 years old, not designed for MT

• Modern C++ (17) concepts

• Thread-safe design

  ➢ Const correctness

  ➢ Stateless algorithms (state provided by caller if necessary)

• Minimization of virtual inheritance (compile-time polymorphism)

• Minimal external dependencies (Eigen library in Acts Core)

• Unit- and integration-tests based development

• Our GitHub repository: here
The ACTS Vertexing Suite

Several ATLAS (primary) vertexing algorithms reimplemented in ACTS + numerically validated w.r.t. ATLAS physics performance

**Vertex (Seed) Finder:**
- Iterative Vertex Finder
- Adaptive Multi-Vertex Finder
- Z-Scan Vertex Finder
- Gaussian Track Density Vertex Finder

**Vertex Fitter:**
- Full-Billoir Vertex Fitter
- Adaptive Multi-Vertex Fitter

**Utilities:**
- Track linearizer, impact point estimator, annealing tool, ...
  - Modern C++
  - Thread-safe, modular design
  - Very well documented
  - Very easy to set up & use (< 40 lines of code, example: backup)
ACTS Vertexing Performance: AMVF

- Comparisons against ATLAS implementation on $<\mu> = 60$ ttbar events
- Perfect agreement on reconstructed objects

**Difference in number of reconstructed vertices**

![Graph showing the difference in number of reconstructed vertices between ACTS and ATLAS](image1)

**Difference in number of tracks associated to vertex**

![Graph showing the difference in number of tracks associated to vertex](image2)
ACTS Vertexing Performance: AMVF

**Vertex position resolution:**
- compared against ATLAS AMVF on $<\mu> = 60$ ttbar events
- very good agreement on micrometer level

**CPU performance:**
- no algorithmic changes (yet)
- significant speed-up w.r.t. ATLAS
Current Vertexing Developments in ACTS

Generalization of track linearization using the ACTS::Propagator:

- No assumption of helical track parameters anymore
- Vertex fitter more robust in all detector regions
- Harmonize primary and secondary vertexing with common math kernels
- Fully integrated time propagation in ACTS: Vertex fitting with time information possible

Gaussian Grid Track Density Vertex Seed Finder:

- Model track as 2-dim Gaussian density grid in $d_0$-$z_0$-plane
- Interested only in density distribution along beam axis:
  - calculate only track contribution along beam axis (red)
- Superimpose all tracks and find maximum along beam axis

Extremely fast in iterative approaches:

- Cache density contribution vector of each track
- Remove it from beam axis density vector if track is removed
Gaussian Grid Track Density Vertex Finder - Results

**Vertex z-position resolution**

- agreement on micrometer level
- depended on bin width
- not optimized yet (trade-off: resolution vs. speed)

**Speed-up vs. number of tracks**

- very significant speed-up w.r.t. to ATLAS algorithm
- greater speed-ups for higher track multiplicities
- especially suitable for high pile-up events
Current Deep Learning Developments in ACTS

- Goal: Construct vertex finding algorithm based on NN (track-to-vertex association)
- Idea: Learn track (+covariance) vector representation $h$ in embedding space $\mathcal{M}$
  
  $\Rightarrow$ "similar" tracks (originating from same vertex) are close in $\mathcal{M}$

Neural network architecture for track embedding:

**Input $x_i$:** Perigee parameters + flattened covariance entries ($5 + 15$)

**Hidden layers:** e.g. 15 layers, 50 neurons each

**Output $h_i$:** Track representation in e.g. 10-dim $\mathcal{M}$

**Concatenation:** Concatenate hidden layers for better gradient flow
Current Deep Learning Developments in ACTS

• But how to find/learn a good representation? → Siamese Neural Network

Siamese Neural Network

- Track embedding network
  - one instance of embedding network used twice
  - shared weights
  - working in parallel

- Score
  - metric $d(h_i, h_j)$
    - e.g. Euclidean
  - similarity score $\in [0,1]$

Train network with track pairs $(x_i, x_j)$

- 50% "positive pairs", i.e. tracks from same vertex
  → desired score: 1.

- 50% "negative pairs", i.e. tracks from neighboring vertices
  → desired score: 0.
Current Deep Learning Developments in ACTS

Training and first results on ACTS smeared truth tracks

• ~ 90% score accuracy ($<\mu>$ = 60 events)
• construct adjacency matrix with track pair scores
  \[ \text{cluster matrix blocks to vertices} \]
• but: non-zero interconnection between blocks

Most straight-forward approach:

• cut all values below threshold, e.g. cut = 0.5
• cluster matrix blocks to vertices
  \[ \text{able to reconstruct vertices} \]

but:
What about stronger interconnection/more difficult cases?
Current Deep Learning Developments in ACTS

Training and first results on ACTS smeared truth tracks

- ~ 90% score accuracy ($\langle \mu \rangle = 60$ events)
- construct adjacency matrix with track pair scores
  
  ➡️ cluster matrix blocks to vertices
- but: non-zero interconnection between blocks

Idea/outlook: Consider problem as graph problem

- Graph nodes: tracks
- Graph edges: similarity scores
  
  ➡️ use graph clustering algorithm to cluster to vertices

Alternative approach:

- let Graph Neural Network predict edge labels, i.e. similarity scores
Summary & Outlook

• Detector-independent track- and vertex reconstruction software ACTS

• MT-capable vertexing suite, modern C++
  • ACTS Adaptive-Multi Vertex Finder most likely to be used in ATLAS Run-3
  • Significant CPU speed-ups with precisely matching physics performance (w.r.t. ATLAS)

• Various vertexing R&D ongoing (classical + machine learning)

• Graph clustering / GNN approaches in early stage but promising
  • more work to be done, only ACTS smeared truth tracks so far
  • to be extended to full ACTS reco tracks with ATLAS ITk description

Thank you!

bastian.schlag@cern.ch
Employing the ACTS Vertexing

**Input to ACTS Vertexing:**

ACTS::BoundParameters

or

Arbitrary user-defined track type:

```cpp
// Dummy user-defined InputTrack type
struct InputTrack {
    InputTrack(const BoundParameters& params) : m_parameters(params) {}
    // Store additional information here */
    // Method to return BoundParameters
    const BoundParameters& parameters() const { return m_parameters; }
    private:
    BoundParameters m_parameters;
};
```

Only requirement for user-defined track:

std::function which returns ACTS::BoundParameters

Set up and run the ACTS vertexing in < 40 lines of code

```cpp
// Set up B-Field and Propagator
ConstantBField bField(0.0, 0.0, 1_T);
EigenStepper<ConstantBField> stepper(bField);
auto propagator = std::make_shared<Propagator>(stepper);

// Linearizer for user defined InputTrack type
Linearizer::Config linearizerCfg(bField, propagator);
Linearizer linearizer(linearizerCfg);

using BilloirFitter = FullBilloirVertexFitter<InputTrack, Linearizer>;
BilloirFitter::Config vertexFitterCfg;
BilloirFitter bFitter(vertexFitterCfg, extractParameters);

using IPEstimator = ImpactPointEstimator<InputTrack, Propagator>;
IPEstimator::Config ipEstimatorCfg(bField, propagator);
IPEstimator ipEstimator(ipEstimatorCfg);

using ZScanSeedFinder = ZScanVertexFinder<BilloirFitter>;
ZScanSeedFinder::Config seedFinderCfg(ipEstimator);
ZScanSeedFinder sFinder(seedFinderCfg, extractParameters);

using VertexFinder = IterativeVertexFinder<BilloirFitter, ZScanSeedFinder>;
VertexFinder::Config cfg(bFitter, linearizer, std::move(sFinder), ipEstimator);
VertexFinder finder(cfg, extractParameters);
VertexFindingOptions<InputTrack> vertexOptions(geoContext, magFieldContext);

std::vector<const InputTrack*> tracksPtr = getTracks();
auto finderResult = finder.find(tracksPtr, vertexOptions);
if (finderResult.ok()) {
    std::vector<Vertex<InputTrack>> allVertices = *finderResult;
}
```
Adaptive Multi-Vertex Finder (AMVF)

**Gaussian Track Density Seed Finder:**
- model each track as 2-dim Gaussian distribution in \(d_0-z_0\)-plane around \((d_0, z_0)\)
- find z value of highest track density along z-axis

**Adaptive Multi-Vertex Fit:**
- weighted adaptive Kalman filter using deterministic annealing scheme
- subject to beamspot and seed constraint
- simultaneous refit of all vertices connected through a chain of vertices and tracks, with weights:

\[
\omega_i(\chi^2_i, T) = \frac{e^{-\frac{1}{2}\chi^2_i/T}}{\sum_j e^{-\frac{1}{2}\chi^2_j/T} + e^{-\frac{1}{2}\chi^2_0/T}}
\]
Iterative Vertex Finder (IVF)

- **Input:** all tracks
- **Seed Finder:** vertex candidate
- **Select compatible tracks**
- **Call vertex fitter**
- **Remove all used tracks from track collection**

**ZScanVertexFinder:**
- Find mode value of all $z_0$ track parameters
- Vertex candidate at position $(z_0, 0, 0)$

**Iterative fitting-after-finding approach:**
- Iteratively find vertex and fit with compatible tracks
- Single track always associated to at most one vertex
- Tracks removed from pool after fitting