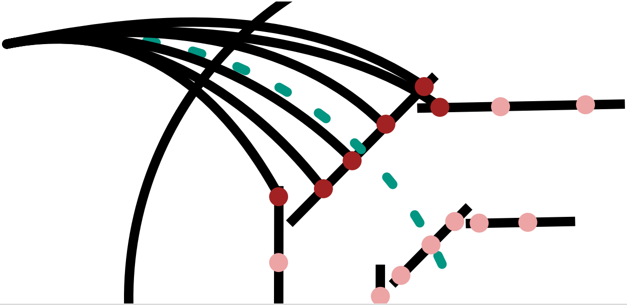
Karlsruher Institut für Technologie

Combinatorial Kalman Filter

Tracking Meeting.

Nils Braun | 28.04.2017

IEKP - KIT



Motivation

CKF

A Combinatorial Kalman Filter uses the principles of the Kalman Filter for track finding. Starting with a seed, it adds hits with some kind of Monte Carlo Tree Search algorithm.
First implementation: extrapolate from CDC to VXD (SVD).

- Reduction of fakes
- Reduction of `SpacePoint` combinations
- Increased finding efficiency

Primary

All MC tracks	10325							
MC track has VXD hits	Yes						No	
	10241						84	
VXD part was found	Yes			No			No	
	8340			1901			84	
MC track has CDC hits	Yes		No	Yes		No	Yes	
	7955		385	1622		279	84	
CDC part was found	Yes	No	No	Yes	No	No	Yes	No
	7023	932	385	1321	301	279	38	46
	Merging Efficiency	CKF	Criteria?	CKF	Very bad!	VXDTF Must help	Criteria?	CDCTF Must help

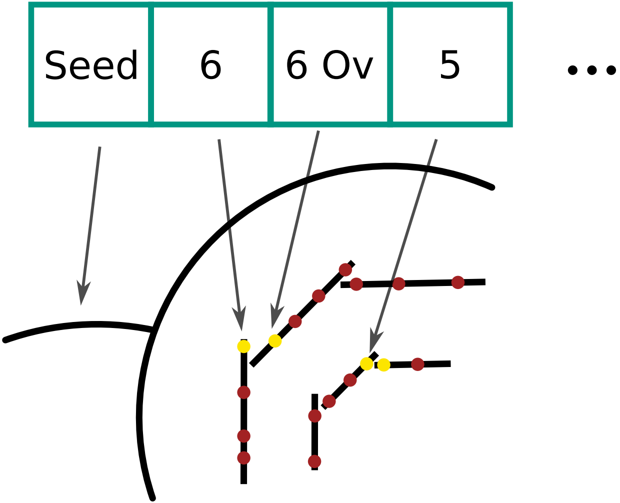
Problems/Challenges

- Extrapolation is slooooow...
- In a normal event, there are more than 10^6 possible combinations.
- You really have to care on memory layout, copying/duplications, fast access of hits etc.

Implementation: Data Flow

- `RecoTracks` are fetched from store array, only successfully fitted ones are used.
- `SpacePoints` are fetched from store array; sorted by layer, ladder, sensor; iterators of start and end of a layer are stored for caching.
- For each `RecoTrack` as seed: do tree search
- ~~*Search best non-overlapping set of tracks with added hits*~~
- Write back to another store array.
- Validation

Tree Search



TreeSearch implementation

```
void traverseTree(StateIterator currentState,
          std::vector<ResultPair>& resultsVector)
{
  StateIterator nextState = std::next(currentState);
  if (nextState == m_states.end()) {
    resultsVector.emplace_back(currentState->finalize());
    return;
  }
  const auto& matchingHits = getMatchingHits(currentState);
  for (const auto& hit : matchingHits) {
    *nextState = AStateObject(currentState, hit); // TODO
    if (useResult(nextState)) {
      traverseTree(nextState, resultsVector);
    }
  }
  *nextState = AStateObject(currentState, nullptr); // TODO
  if (useResult(nextState)) {
      traverseTree(nextState, resultsVector);
  }
}
```

State Object

```
RecoTrack* m_seedRecoTrack = nullptr;
const SpacePoint* m_spacePoint = nullptr;
unsigned int m_number = N;
CKFCDCToVXDStateObject* m_parent = nullptr;

double m_chi2 = 0;

bool m_isFitted = false;
bool m_isAdvanced = false;

genfit::MeasuredStateOnPlane m_measuredStateOnPlane;

bool m_hasCache = false;
genfit::MeasuredStateOnPlane m_cachedMeasuredStateOnPlane;
```

useResult

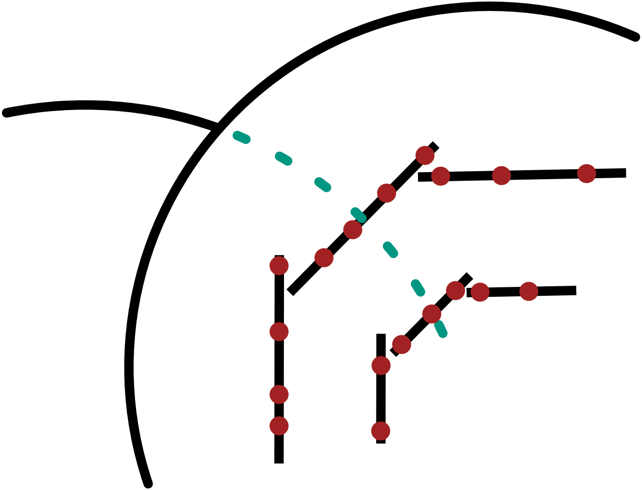
```
bool useResult(StateIterator currentState)
{
  Weight weight = m_firstFilter(*currentState);
  if (std::isnan(weight)) {
    return false;
  }
  if (not advance(currentState)) {
    return false;
  }
  weight = m_secondFilter(*currentState);
  if (std::isnan(weight)) {
    return false;
  }
  if (not fit(currentState)) {
    return false;
  }
  weight = m_thirdFilter(*currentState);
  return not std::isnan(weight);
}
```

Next steps

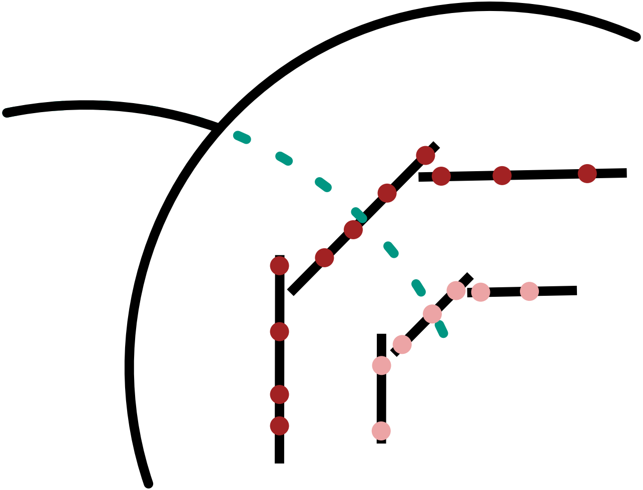
- Remove the state constructors (again).
- Refine the geometry overlap handling (fix bug; no need for fit and advance, no need for state?).
- Implement the quality estimation and the final selection (reuse VXD code).
- Present first (physical) results on F2F meeting.

Backup

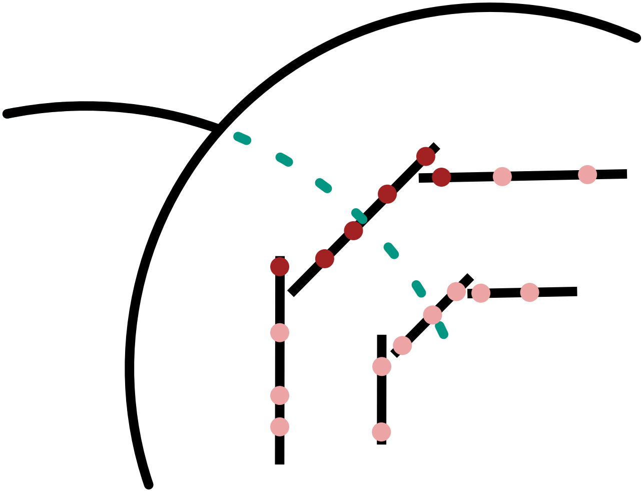
Principles



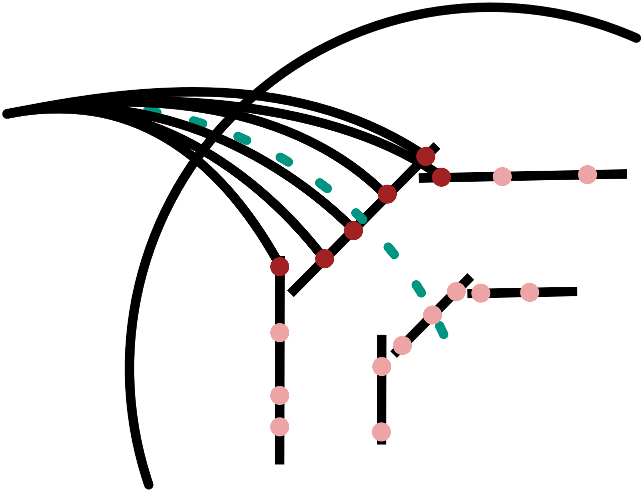
Principles



Principles



Principles



Principles