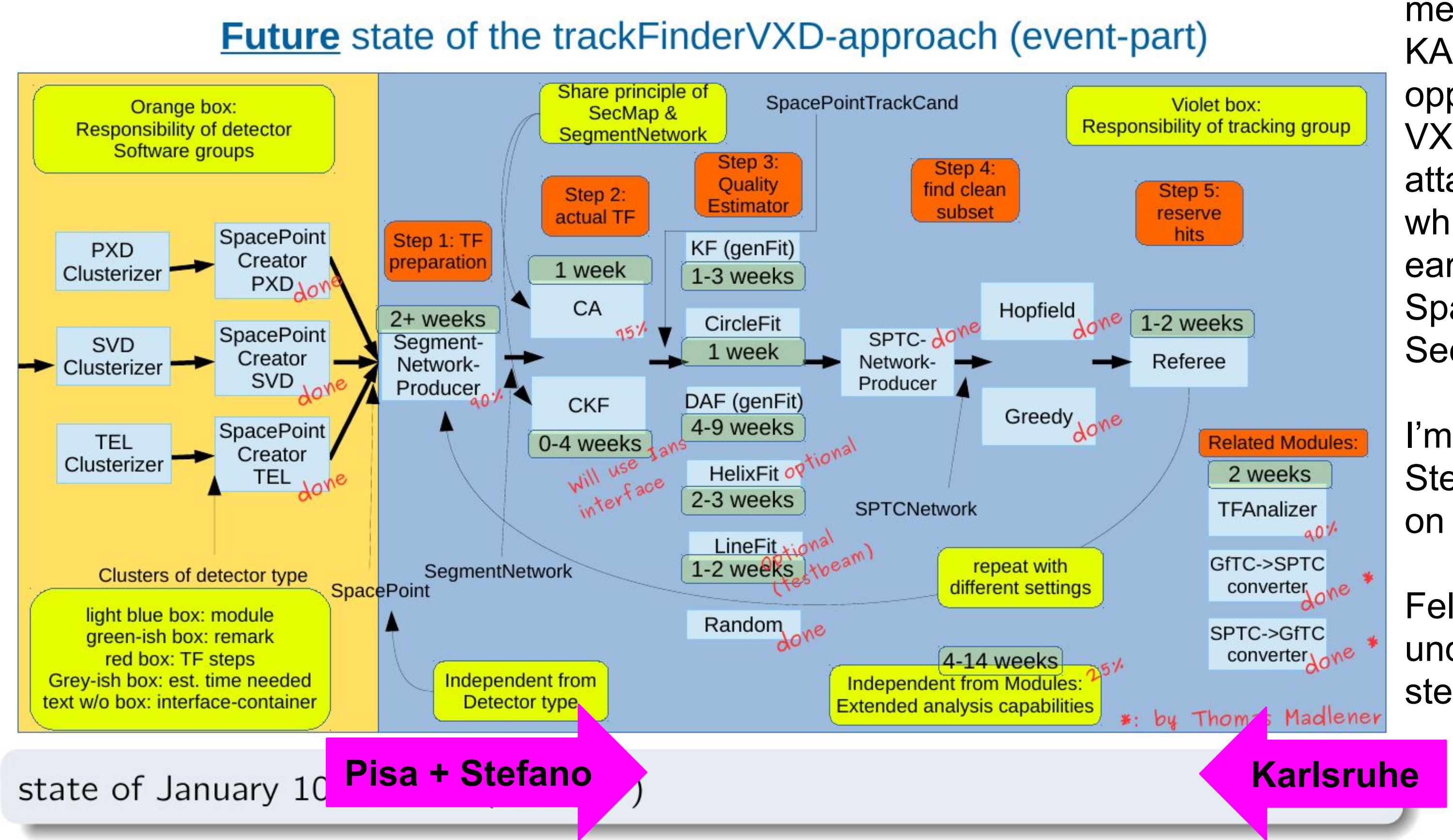
Status TrackFinderVXDCellOMat

Slide from last F2F by Jakob



Jakob is currently visiting us in Karlsruhe.

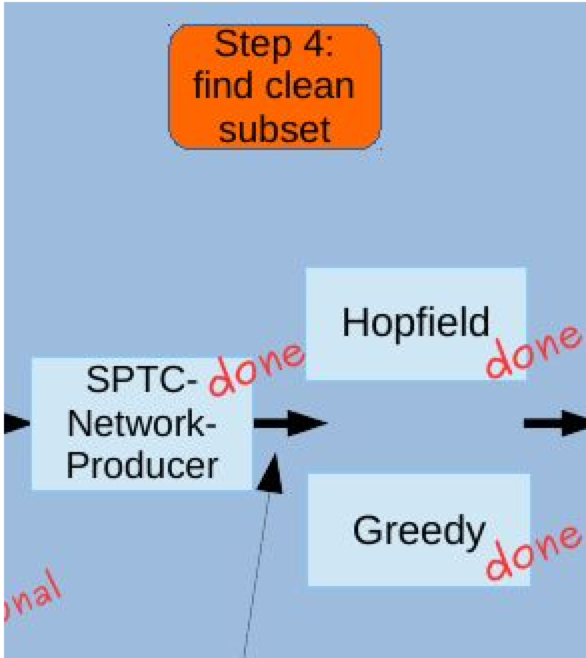
At the last F2F tracking meeting, we agreed, that KA helps opportunistically in the VXD tracking by attacking the later steps, while Pisa works on the earlier steps and Stefano Spataro works on the Sector Map set up.

I'm currently working on Step 4 (and plant to work on 5).

Felix is trying to understand the stuff in step 2, 3

Status

- There are working modules for step 4 in the Track Finder, but
  - it was difficult to find a good starting point to understand them,
  - their time consumption in a relevant setting was considerable.
- My impression was, that
  - part of the problem was the fact, that a very complicated Object, the SpTcNetwork (SpacePoint Track Candidate Network) was the communicating object on the DataStore,
  - enormous amounts of debug output was partially put in functions, of which it wasn't clear, that they are not doing anything with respect to the algorithm.



Example

This is the DataStore object for module to module communication.

The comment for this object gives only redundant information.

```
/** The SpTcNetwork class.
 *
 *  Is intended to be used as StoreObjPtr
 */
class SpTcNetwork : public RelationsObject {

protected:
  /** ************************* DATA MEMBERS ************************* */

  /** the actual network, packed here to be able to be stored as StoreArray-Compatible object */
  TCNetworkContainer<SPTCAvatar<TCCompetitorGuard>, TCCompetitorGuard > m_network;

  /** if true, overlaps are checked via SpacePoints. If false, overlaps are checked via clusters */
  bool m_compareSPs;

  ClassDef(SpTcNetwork, 1) // last member changed:  m_network
public:
  /** ************************* CONSTRUCTORS ************************* */

  /** standard constructor for Root IO */
  SpTcNetwork() :
    m_network(TCNetworkContainer<SPTCAvatar<TCCompetitorGuard>, TCCompetitorGuard >()),
    m_compareSPs(false) {}


  /** specific constructor allowing to set comparison mode */
  SpTcNetwork(bool compareSPs) :
    m_network(TCNetworkContainer<SPTCAvatar<TCCompetitorGuard>, TCCompetitorGuard >()),
    m_compareSPs(compareSPs) {}
```

Complicated Network Container object, that will use algorithm objects, that are given to it to perform stuff, e.g. the DataStore object is responsible for the execution of the algorithm.

→ Try putting simple stuff on the DataStore, so one can start somewhere to understand the code.

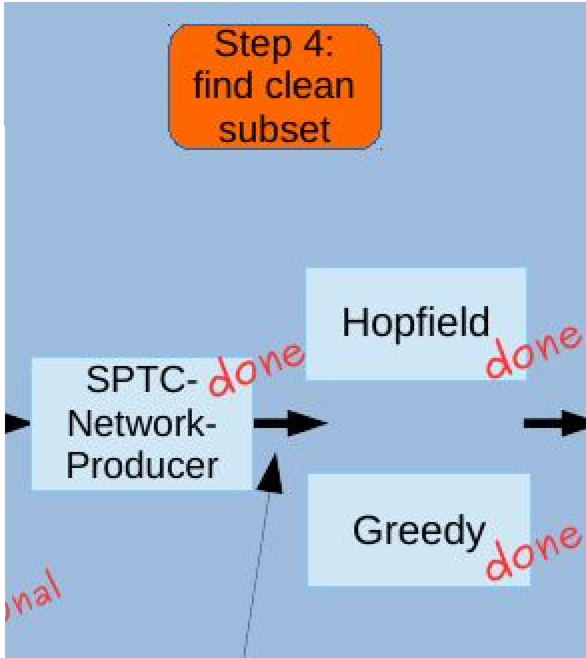
Rewrite of Step 4

- What is really needed by Hopfield/Greedy from the SPTCNetworkProduce?
  - A list of which track is in conflict with which other track!
  - Perhaps something as simple as the OverlapNetwork

```
namespace Belle2 {
  class OverlapNetwork : public RelationsObject {
  public:
    OverlapNetwork(const std::vector <std::vector <unsigned short> >& overlapMatrix) :
      m_overlapMatrix(overlapMatrix)
    {}

    std::vector<unsigned short>& getOverlapForTrackIndex(unsigned short trackIndex)
    {
      return m_overlapMatrix[trackIndex];
    }

  private:
    std::vector<std::vector <unsigned short> > m_overlapMatrix;
    ClassDef(OverlapNetwork, 1)
  };
}
```



How is the OverlapNetwork filled?

- With the SVDOverlapChecker module:

```
namespace Belle2 {

  /** Checks overlap of SpacePointTrackCandidates, e.g. multiple usage of the same 1D Cluster.
   *
   *  Expects StoreArray of SpacePointTrackCandidates;
   *  Produces OverlapNetwork, that can be asked, with which other ones a track candidate overlaps;
   *
   *  The algorithm idea is the following:<br>
   *  - Loop over all SpacePointTrackCandidates<br>
   *  - Loop over all SpacePoints of each candidate<br>
   *  - Fill for 1D SVD clusters a matrix[ClusterID][TrackIndex] <br>
   *  - Fill the OverlapNetwork (which really is an overlap matrix)
   */
  class SVDOverlapCheckerModule : public Module {

  public:
    /** Constructor of the module. */
    SVDOverlapCheckerModule();

    /** Initializes the Module. */
    void initialize() override final
    {
      //make requirements known to the framework;
      //TODO: Decide on either taking a parameter or rewrite the testing steering file to use the default names.
      m_spacePointTrackCands.isRequired("caSPTCs");
      m_svdClusters.isRequired();

      m_overlapNetwork.registerInDataStore();
    }

    /** Checks for overlaps and fills the OverlapNetwork. */
    void event() override final;
```

```
SVDOverlapCheckerModule::SVDOverlapCheckerModule() : Module()
{
  setDescription("Module checks for overlaps of SpacePointTrackCands\
      and stores them in an OverlapNetwork, which is basically a matrix of overlaps.");
}

void SVDOverlapCheckerModule::event()
{
  //Create matrix[svdCluster][track]
  unsigned short nHits = m_svdClusters.getEntries();
  vector<vector<unsigned short> > svdHitRelatedTracks(nHits);
  //TODO: Check if one saves time by reserving some space for each single of those vectors;

  //now fill the cluster/track matrix:
  unsigned short nSpacePointTrackCandidates = m_spacePointTrackCands.getEntries();
  for (int ii = 0; ii < nSpacePointTrackCandidates; ii++) {

    for (auto && spacePointPointer : m_spacePointTrackCands[ii]->getHits()) {
      //only SVD is handled with this algorithm
      if (spacePointPointer->getType() != VXD::SensorInfoBase::SensorType::SVD)  continue;

      //at the position of the svdCluster Index, the track index is pushed back;
      RelationVector<SVDCluster> svdClusterRelations = spacePointPointer->getRelationsTo<SVDCluster>();
      for (SVDCluster const& svdClusterPointer : svdClusterRelations) {
        svdHitRelatedTracks[svdClusterPointer.getArrayIndex()].push_back(ii);
      }
    }
  }

  //Create the overlap matrix and store it into the OverlapNetwork
  OverlapMatrixCreator overlapMatrixCreator(svdHitRelatedTracks, nSpacePointTrackCandidates);
  m_overlapNetwork.appendNew(OverlapNetwork(overlapMatrixCreator.getOverlapMatrix()));
}
```

I'm not sure, if this algorithm is reasonable for PXD based SpacePoints, as there are typically more PXD clusters and SpacePointTrackCandidates.

Creation of the actual overlapMatrix in a separate object.

From the OverlapMatrixCreator.

```
std::vector <std::vector<unsigned short> > getOverlapMatrix()
{
  //Loop over all the hits and make corresponding connections for the tracks
  for (auto && tracks : m_hitRelatedTracks) {
    for (unsigned short ii = 0; ii < tracks.size(); ii++) {
      for (unsigned short jj = ii + 1; jj < tracks.size(); jj++) {
        m_overlapMatrix[tracks[ii]].push_back(tracks[jj]);
        m_overlapMatrix[tracks[jj]].push_back(tracks[ii]);
      }
    }
  }

  //sort and erase overlaps
  //TODO: Check in realistic situation alternative approach:
  //see http://stackoverflow.com/questions/1041620/whats-the-most-efficient-way-to-erase-duplicates-and-sort-a-vector
  for (auto && overlapTracks : m_overlapMatrix) {
    std::sort(overlapTracks.begin(), overlapTracks.end());
    overlapTracks.erase(std::unique(overlapTracks.begin(), overlapTracks.end()), overlapTracks.end());
  }


  return m_overlapMatrix;
}
```

Main Goal is working Setup for “Round1”

- Use the TrackFinderVXDCellOMat for SVD only track finding;
- Do either/or
  - Extrapolate to PXD and add all hits in the proximity, e.g. “Region of Interest” to the track and let the DAF decide, which of the PXD hits should really be taken up by the track;
  - Use a Combinatorial Kalman Filter to extrapolate to the PXD;

In any case for this simple idea, we don’t need the XXXOverlapChecker (an OverlapCheckerModule exists already, checking for the whole detector, if there are any overlaps) to work with PXD hits from the start;

Performance?

If the output from the SVDOverlapChecker is as useful for the Hopfield Network as the SPTCNetwork, we have a substantial improvement in CPU resource usage.
Hopfield itself (not shown) is as well bad, but can fairly easily be improved (matrix library). Remaining to be attacked is the SegmentNetworkProducer, that depends strongly on the sector map, could be reduced perhaps with more passes... , Eugenio's, Stefano's,...(?) work

- Using a slightly modified tracking/vxdCaTracking/extendedExamples/secMapGen/testVXDTFRelatedModules.py I can get the following time consumptions for the modules:

  [
  20 muons per event;
  $0.1 < p < 0.145$
  $60 < \theta < 85$
  $0 < \phi < 90$
  ]
  (fairly harsh, see next slides)

Name	Calls	VMemory(MB)	Time(s)	Time(ms)/Call
RootInput	30	0	0.07	2.34 +- 0.44
EventInfoPrinter	30	0	0.00	0.00 +- 0.00
Gearbox	30	0	0.00	0.00 +- 0.00
Geometry	30	0	0.00	0.00 +- 0.00
EventCounter	30	0	0.00	0.13 +- 0.03
SectorMapBootstrap	30	0	0.00	0.00 +- 0.00
SpacePointCreatorSVD	30	0	0.03	1.10 +- 0.12
SpacePointCreatorPXD	30	0	0.01	0.18 +- 0.02
SpacePoint2TrueHitConnector	30	0	0.05	1.81 +- 0.18
GFTC2SPTCConverter	30	0	0.04	1.26 +- 0.14
SPTCReferee	30	0	0.01	0.20 +- 0.04
SpacePoint2TrueHitConnector	30	0	0.17	5.61 +- 0.70
RawSecMapMerger	30	0	0.00	0.00 +- 0.00
SegmentNetworkProducer	30	0	55.22	1840.61 +- 641.71
TrackFinderVXDCellOMat	30	0	0.31	10.21 +- 8.08
SPTCvirtualIPRemover	30	0	0.00	0.15 +- 0.13
QualityEstimatorVXDCircleFit	30	0	0.07	2.46 +- 1.90
SPTCNetworkProducer	30	0	29.20	973.21 +-2322.67
SVDOverlapChecker	30	0	3.73	124.19 +- 216.05
TrackFinderVXDAnalizer	30	0	0.82	27.40 +- 20.79
Total	30	0	89.82	2994.10 +-2662.37

```
SegmentNetworkProducerModule:event: event 29

[INFO] As no network (DirectedNodeNetworkContainer) was present, a new network was created
[DEBUG] Pass 0 is finished with 5 rounds (negative numbers indicate fail)! Of 819 cells total, their states were:
had 536 cells of state 0
had 208 cells of state 1
had 69 cells of state 2
had 6 cells of state 3
had 0 cells of state 4
  { module: TrackFinderVXDCellOMat @include/tracking/trackFindingVXD/algorithms/CellularAutomaton.h:126 }
[INFO] TrackFinderVXDAnalizer-Event 29: the tested TrackFinder found:  IDs (total/perfect/clean/contaminated/clone/tooShort/ghost: 20/20/0/0/170/0/156) within 346 TCs
 and lost (test/ref) 0/0 TCs. nBadCases: 0 refClones: 0
There are 20 referenceTCs, with mean of 0.000000/8.500000 PXD/SVD clusters
There are 346 testTCs, with mean of 0.000000/8.132948 PXD/SVD clusters
[INFO] TrackFinderVXDAnalizer-Event 29: the tested TrackFinder had an efficiency : total/perfect/clean/contaminated/clone/tooShort/ghost: 100.000000%/100.000000%/0.00
0000%/0.000000%/850.000000%/0.000000%/780.000000%
[INFO] TrackFinderVXDAnalizer-Event 29: totalCA|totalMC|ratio of pxdHits 0|0|-nan, svdHits 2814|170|16.552940 found by the two TFs
[INFO] EventCounterModule - Event: 30 having 43/167 pxd/svdClusters. Detailed info:
PXD:
L1: nClusters: 22, nPixels: 29
L2: nClusters: 21, nPixels: 31
PXD total: nClusters: 43, nPixels: 60
meanPXD per Layer: nClusters: 21.5, nPixels: 30
SVD:
L3: nClusters: 36, nClusterCombinations: 164, nUStrips: 51, nVStrips: 41, nStripsTotal: 92
L4: nClusters: 49, nClusterCombinations: 191, nUStrips: 55, nVStrips: 51, nStripsTotal: 106
L5: nClusters: 39, nClusterCombinations: 86, nUStrips: 42, nVStrips: 36, nStripsTotal: 78
L6: nClusters: 43, nClusterCombinations: 108, nUStrips: 41, nVStrips: 43, nStripsTotal: 84
SVD total: nClusters: 167, nClusterCombinations: 549, nUStrips: 189, nVStrips: 171, nStripsTotal: 360
meanSVD per Layer: nClusters: 41.75, nClusterCombinations: 137.25, nUStrips: 47.25, nVStrips: 42.75, nStripsTotal: 90
```

Random event, e.g.:
549 SpacePoints total;
1: 164
2: 191
3: 86
4: 108

Particle name	$\Upsilon(4S)$-only	BG-only	$\Upsilon(4S)+$BG	$\Upsilon(4S)+2\times$BG
L3 strips u/v	49.2/36.7	260.0/121.7	308.1/158.0	562.2/278.8
L3 occupancy u/v [%]	0.46/0.34	2.42/1.13	2.87/1.47	5.23/2.59
L3 clusters u/v	11.8/11.8	39.0/37.9	50.3/49.3	87.0/86.1
L3 SpacePoints	26.1	233.9	318.0	791.0
L4 strips u/v	39.4/29.1	120.3/61.2	159.1/90.1	277.8/150.6
L4 occupancy u/v [%]	0.17/0.19	0.52/0.40	0.69/0.59	1.21/0.98
L4 clusters u/v	12.7/12.6	29.9/26.7	42.5/39.2	71.8/65.3
L4 SpacePoints	22.5	100.5	143.1	320.4
L5 strips u/v	37.3/28.5	122.7/67.2	160.1/95.8	282.7/162.9
L5 occupancy u/v [%]	0.10/0.12	0.33/0.27	0.43/0.39	0.77/0.66
L5 clusters u/v	12.3/12.1	35.0/30.5	47.3/42.7	82.0/72.9
L5 SpacePoints	19.2	99.3	132.3	299.3
L6 strips u/v	38.3/28.6	134.6/76.8	172.9/105.4	307.1/182.0
L6 occupancy u/v [%]	0.06/0.07	0.22/0.19	0.28/0.26	0.50/0.44
L6 clusters u/v	12.4/12.2	42.1/36.3	54.4/48.5	96.2/84.5
L6 SpacePoints	17.0	100.8	127.9	283.1
Total strips u/v	164.3/122.8	637.6/326.8	800.3/449.3	1429.8/774.4
Total occupancy u/v [%]	0.12/0.14	0.48/0.37	0.61/0.51	1.08/0.88
Total clusters u/v	49.2/48.7	146.0/131.3	194.4/179.6	337.1/308.9
Total SpacePoints	84.8	534.6	721.3	1693.8

able 3.2.: Mean number of clusters and SpacePoints per Layer (L3-L6) and total of relevant ases - taken from a MC sample of 30,000 $\Upsilon(4S)$ events described in Section 11.1.

Looks like mostly BKG, but consider the fact, that the Y(4S) decays have probably a higher variance.

Layer 3 is worse than the others, but only ~ x 2.

Plot from Jakob's thesis draft.

Ghost hits dominate!
#Cluster u ~200, #SpacePoint ~800
But keep in mind, that this varies!
See next page

Only O(10%) of SpacePoints come from Signal.

Summary

- Given, that Felix and me started to work on the VXD tracking code only this week, I’m fairly happy about the amount of things we have already understood and worked on.