The VXDTF SectorMap

Eugenio Paoloni, Thomas Lück
INFN Pisa

0) The Problem

- Given a set of 721 space points
  (average Y4s event with background)
  in how many way you can partition it
  in 11 non overlapping subsets
  (average multiplicity of a Y4s event)
  + one background remainder ?

StirlingS2 [721, 12] =

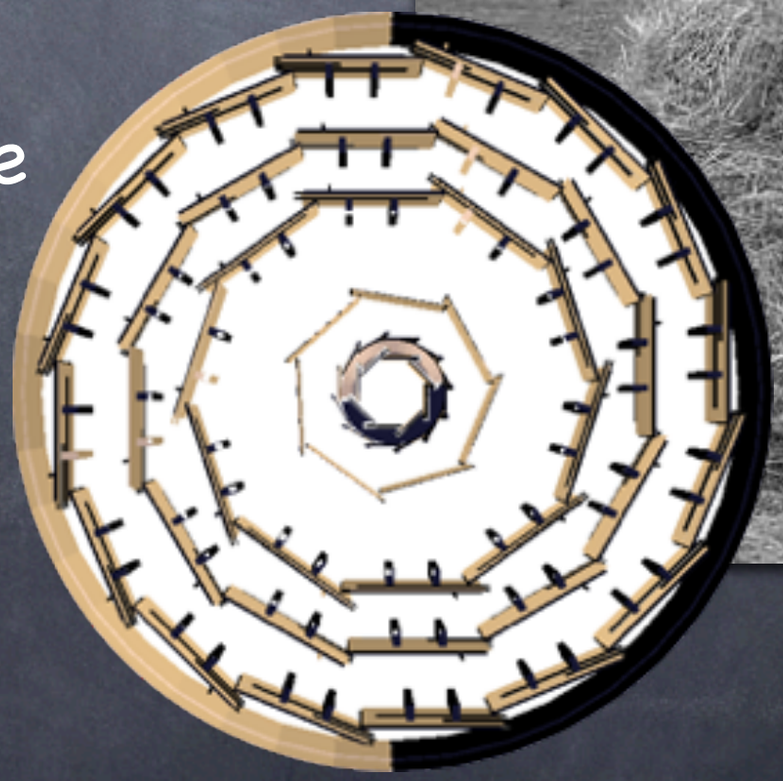
256650099078978006667088616772975496850875880893066055525716018292558910782038066916888719792944862718659274572536456502901088354377397379331397597931741424698428436073688718226102760722709547140898035523971203122114890537201700856294359272835759843127458619294579596618123081593424078154960794816665080592461741176485396425924599038084751287313745380635003292133223323698755748871712031303818431988121727536460644983909991904970952504738868295120710957006187050400042922233557417362282951807511286886448951985029002696379979138607226566468725745856779970945207291268251499135270932899959428010284943889368722198097544826206432893152763861321510372480943460251901002438991714914434737676893809223824395545479725193984726011271563610490296437306000659391007765022600071828

almost

$$2.57\ 10^{769}$$

And just a single one of these partitions is the correct one...

- Do not consider unlikely combinations
  - Tracks with much more than 1 Space Point per layer
  - Tracks jumping around erratically
- Just consider reasonable combinations



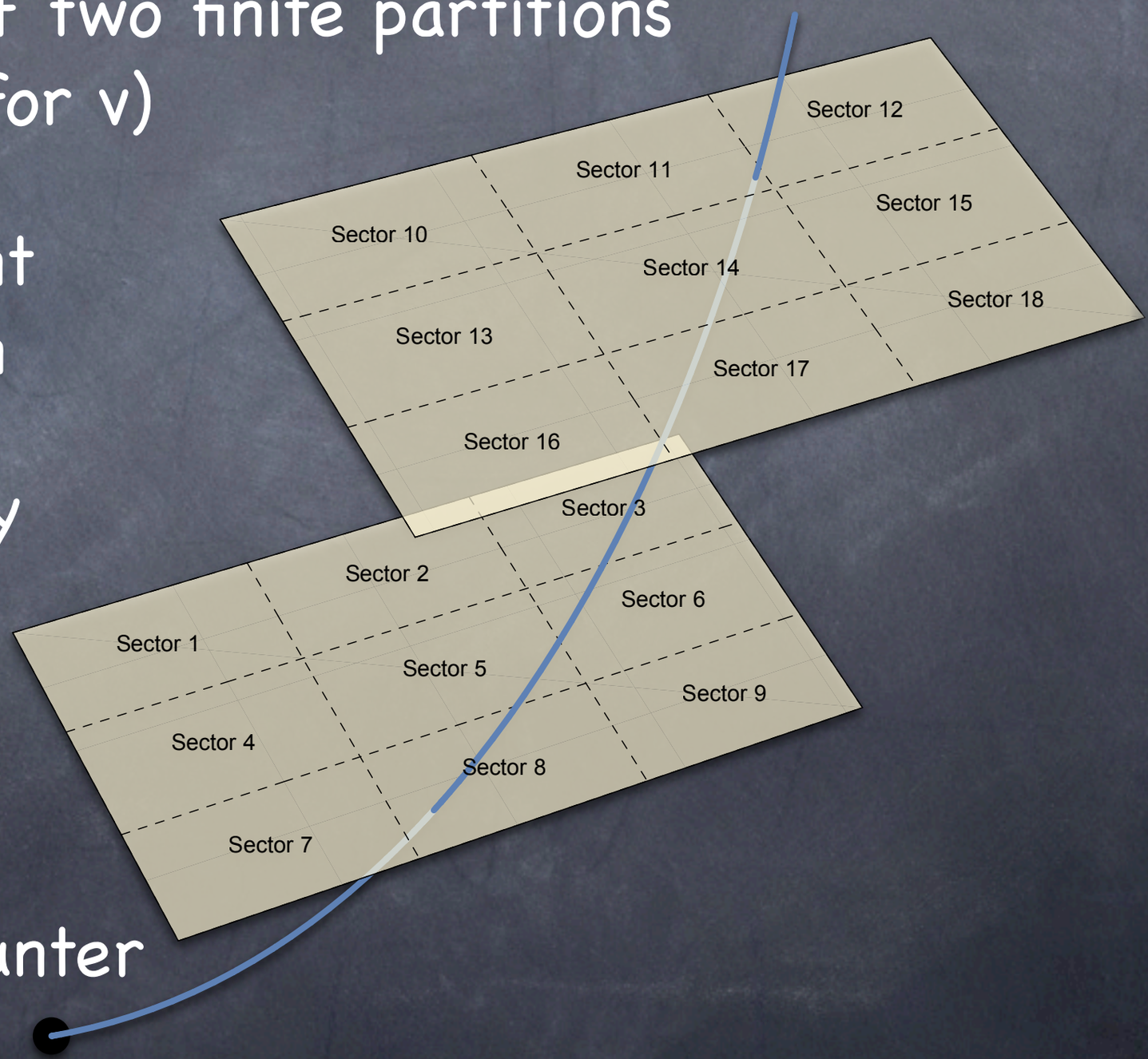
1) The Sector

- We define a map from the surface of each sensor to the unit square [0,1] ⊗ [0,1] (normalized local coordinates).

- For each sensor we define a partition of its surface by the cartesian product of two finite partitions of [0,1] (one for u, one for v)

- The Sector is an element of this surface partition

- The sectors are uniquely identified by their FullSecID defined in tracking/dataobjects. It is defined by: layer, ladder, sensor, counter



How we do define the sectors?

- The object VXDTFFilters in tracking/trackFindingVXD/environment take care of this task providing the method:

```
int
VXDTFFilters::addSectorsOnSensor(
 const std::vector<double>&                  normalizedUsups,
 const std::vector<double>&                  normalizedVsups,
 const std::vector< std::vector<FullSecID> >& sectorIds)
```

N.B.: All the sectorIds have to refer to the same sensor. The normalizedUsups and Vsups (sup stands for Supremum) can be in random order

For Example:

```
VXDTFFilters::addSectorsOnSensor(

{ 0.5 }, {0.5,0.75} ,

{
 {FullSecID(l,s,se,0 ), FullSecID(l,s,se,1 ), FullSecID(l,s,se,2 ) },
 {FullSecID(l,s,se,3 ), FullSecID(l,s,se,4 ), FullSecID(l,s,se,5 ) }
} );
```

Sensor: se
on layer l
sector s

Who takes care of the creation of the sectors?

- The SectorMapBootstrapModule in module/vxdtfRedesign
- At present all the sectors are partitioned in the very same way
  - We need to investigate how close to the optimum is this (trade off memory foot print / speed)

1.2) Filters, Friends, neighbours, next neighbours

Filters for SpacePoints combination

- We define "Friends" two sectors for wich there is a sizable probability for real tracks to leave two consecutive (in time) SpacePoint on them: the first one on the inner sector, the second on the outer one.
- We define a filter for each pair of friend sectors to select good SpacePoints combinations (aka Segments).

2 Space point Filters type

```
  template<class point_t>
  class VXDTFFilters {

...

  public:

    typedef decltype(
    (
      (0. <= Distance3DSquared<Belle2::SpacePoint>() <= 0.) &&
      (0. <= Distance2DXYSquared<Belle2::SpacePoint>() <= 0.)&&
      (0. <= Distance1DZ<Belle2::SpacePoint>() <= 0.)&&
      (0. <= SlopeRZ<Belle2::SpacePoint>() <= 0.)&&
      (0. <= Distance3DNormed<Belle2::SpacePoint>() <= 0.)
    ).observe(ObserverCheckMCPurity())
  ) twoHitFilter_t;
```

How to modify this type: define your smart variable

```
#include <tracking/trackFindingVXD/sectorMap/filterFramework/
SelectionVariable.h>

SELECTION_VARIABLE( MySmartVariable, SpacePoint,
                    static double value(const SpacePoint& p1,
                                        const SpacePoint& p2)
  { return Some Fancy expression of p1 and p2; };
);
```

- Preferably in trackFindingVXD/sectorMap/twoHitVariables

Then use it...

```
#include <trackFindingVXD/sectorMap/twoHitVariables/MySmartVariable.h>

template<class point_t>
  class VXDTFFilters {

...

  public:
    typedef decltype(
      (  MySmartVariable<Belle2::SpacePoint>() > 0 || (
        (0. <= Distance3DSquared<Belle2::SpacePoint>() <= 0.) &&
        (0. <= Distance2DXYSquared<Belle2::SpacePoint>() <= 0.)&&
        (0. <= Distance1DZ<Belle2::SpacePoint>() <= 0.)&&
        (0. <= SlopeRZ<Belle2::SpacePoint>() <= 0.)&&
        (0. <= Distance3DNormed<Belle2::SpacePoint>() <= 0.) )
      ).observe(ObserverCheckMCPurity())
    ) twoHitFilter_t;
```

Well... you have to train the new filter off course

- At present the implementation of the training is not yet very terse

- One module collect the data from simulated events: VXDTFTrainingDataCollector

- One module merges the data and trains the filters: RawSecMapMergerModule